LECTURE NOTES IN COMPUTATIONAL SCIENCE AND ENGINEERING 136

Hans-Joachim Bungartz · Severin Reiz
Benjamin Uekermann · Philipp Neumann
Wolfgang E. Nagel *Editors*

Software for Exascale Computing SPPEXA 2016 – 2019

OPEN ACCESS

Springer

Hans-Joachim Bungartz • Severin Reiz •
Benjamin Uekermann • Philipp Neumann •
Wolfgang E. Nagel
Editors

Software for Exascale Computing - SPPEXA 2016-2019

Springer Open

Editors
Hans-Joachim Bungartz
Technische Universität München
Garching, Germany

Severin Reiz
Technische Universität München
Garching, Germany

Benjamin Uekermann
Department of Mechanical Engineering
Eindhoven University of Technology
Eindhoven, The Netherlands

Philipp Neumann
Helmut-Schmidt-Universität Hamburg
Hamburg, Germany

Wolfgang E. Nagel
Technische Universität Dresden
Dresden, Germany

ISSN 1439-7358 ISSN 2197-7100 (electronic)
Lecture Notes in Computational Science and Engineering
ISBN 978-3-030-47955-8 ISBN 978-3-030-47956-5 (eBook)
https://doi.org/10.1007/978-3-030-47956-5

Mathematics Subject Classification: 82-08

This book is an open access publication.

Preface

This volume summarizes the research done and results obtained in the second funding phase of the Priority Program 1648 "Software for Exascale Computing" (SPPEXA) of the German Research Foundation (DFG). In that respect, it both provides an overview of SPPEXA's achievements and represents a continuation of Vol. 113 in Springer's series "Lecture Notes in Computational Science and Engineering", the corresponding report of SPPEXA's first funding phase.

For some general remarks on the uniqueness of SPPEXA—as the first strategic, i.e. board-initiated Priority Program of DFG; as the first tri-national Priority Program with synchronized collaborative research in Germany, France, and Japan; as a multi-disciplinary endeavor involving informatics and mathematics, but also various fields from engineering, the sciences, and the life sciences; and as the first holistic approach to research on High-Performance Computing (HPC) software at the level of fundamental research—we refer to the overview contribution of Bungartz et al. (see chapter "Software for Exascale Computing: Some Remarks on the Priority Program SPPEXA") in this volume. There, also some statistics are provided.

The spirit of the international collaboration, whether in a bi-lateral (German–Japanese) or in a tri-lateral (French–Japanese–German) setting, can be found and felt in several of the reports of 16 out of 17 SPPEXA consortia. This structured and institutionalized collaboration was not easy to establish, and we are grateful for the shared enthusiasm, commitment, and support of the three involved funding agencies: the German Research Foundation (DFG), the Agence Nationale de la Recherche (ANR), and the Japan Science and Technology Agency (JST). The synergies emerging from bringing together the expertise of groups from three countries did not only boost the respective project work itself, it also prepared the ground for ongoing partnerships as well as for a topical extension towards the interplay of HPC and Artificial Intelligence—a field that both benefits tremendously from HPC and, at the same time, fosters HPC with new concepts.

As always, many people helped to make SPPEXA in general and this volume in particular a great success. Concerning the first, our thanks go to the agencies already mentioned and their responsible officers; then to all the SPPEXA researchers in 17 consortia who made SPPEXA such a wonderful and productive research experience; and finally to all helping hands that supported SPPEXA in terms of organizing and hosting events such as workshops, doctoral retreats, minisymposia, gender workshops, annual plenary meetings, and so forth. Moreover, concerning the preparation of this volume, we are grateful to Dr. Martin Peters and Leonie Kunz from Springer for their support—as in previous cases, it was again a pleasure to collaborate. Finally, we thank Mirco Troue, Tina Angerer, and Michael Obersteiner for their support in proofreading and compiling this book.

The first exascale systems are expected to be available in about one year. For sure, there is still a lot of work to be done to let cutting-edge science applications fully exploit their potential. However, we are fully convinced that SPPEXA contributed significantly to pave the way towards exascale computers and their usage.

Garching, Germany	Hans-Joachim Bungartz
Garching, Germany	Severin Reiz
Eindhoven, Netherlands	Benjamin Uekermann
Hamburg, Germany	Philipp Neumann
Dresden, Germany	Wolfgang E. Nagel

Contents

Acronyms

AABB	Axis-aligned bounding box
AMG	Algebraic multigrid
AoS	Array of structures
AST	Abstract syntax tree
BiCGSTAB	Biconjugate gradient stabilized
CFD	Computational fluid dynamics
CFL	Courant-Friedrichs-Lewy
CG	Conjugate gradient
CGS	Coarse-grid solver
CR	Conjugate residual
CSE	Common subexpression elimination
CST	Concrete syntax tree
CSV	Comma-separated values
DAG	Directed acyclic graph
DG	Discontinuous Galerkin
DSL	Domain-specific language
DTFT	Discrete-time Fourier transform
FAS	Full approximation scheme
FD	Finite differences
FDM	Finite difference method
FE	Finite elements
FEM	Finite element method
FFC	FEniCS form compiler
FFT	Fast Fourier transform
FPGA	Field-programmable gate array
FV	Finite volumes
FVM	Finite volume method
GA	Genetic algorithm
GCC	GNU compiler collection
GMG	Geometric multigrid
GP	Genetic programming

GPL	General-purpose language
HPC	High-performance computing
ICC	Intel C++ compiler
IR	Intermediate representation
ISL	Integer set library
IV	Internal variable
JIT	Just-in-time
LBM	Lattice Boltzmann method
LCCSE	Loop-carried common subexpression elimination
LFA	Local Fourier analysis
MINRES	Minimal residual
MSVC	Microsoft visual C++
PDE	Partial differential equation
PGAS	Partitioned global address space
RBGS	Red-black Gauss-Seidel
RSDFT	Real space density functional theory
SIMPLE	Semi-implicit method for pressure linked equations
SoA	Structure of arrays
SOR	Successive over-relaxation
SPDE	Stochastic partial differential equation
SPL	Software product line
SpMV	Sparse matrix-vector multiplication
SWE	Shallow water equations
TBB	Threading building blocks
TPDL	Target platform description language
TSC	Time stamp counter
TSFC	Two stage form compiler
UFC	Unified form-assembly code
UFL	Unified form language

Part I
SPPEXA: The Priority Program

Software for Exascale Computing: Some Remarks on the Priority Program SPPEXA

Hans-Joachim Bungartz, Wolfgang E. Nagel, Philipp Neumann, Severin Reiz, and Benjamin Uekermann

Abstract SPPEXA, the Priority Program 1648 "Software for Exa-scale Computing" of the German Research Foundation (DFG), was established in 2012. SPPEXA was DFG's first strategic Priority Program—strategic in the sense that it had been the initiative of DFG's board to suggest a larger and trans-disciplinary funding scheme to support the development of software at all levels that would be able to benefit from future exa-scale systems. A proposal had been formulated by a team of scientists representing domains across the STEM fields, evaluated in the standard format for Priority Programs, and financed via special funds. Operations started in January 2013, and after two 3-year funding phases and a cost-neutral extension, SPPEXA's activities will come to an end by end of April, 2020. A final international symposium took place on October 21–23, 2019, in Dresden, and this volume of Springer's Lecture Notes in Computational Science and Engineering—the second SPPEXA-related one after the corresponding report of Phase 1 (see Appendix 3 in [1])—contains reports of 16 out of 17 SPPEXA projects (the project ExaSolvers will deliver its report as a special issue of Springer's journal Computing and Visualization in Science) and is, thus, a comprehensive overview of research within SPPEXA.

While each single project report emphasizes the respective project's individual research outcomes and, thus, provides one perspective of research in SPPEXA, this contribution, co-authored by the two scientific coordinators—Hans-Joachim

H.-J. Bungartz · S. Reiz (✉)
Technical University of Munich, Garching, Germany
e-mail: bungartz@in.tum.de; reiz@in.tum.de

W. E. Nagel
Technical University of Dresden, Dresden, Germany

P. Neumann
Helmut-Schmidt-Universität Hamburg, Hamburg, Germany
e-mail: philipp.neumann@hsu-hh.de

B. Uekermann
Eindhoven University of Technology, Eindhoven, Netherlands

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_1

Bungartz and Wolfgang E. Nagel—and by three of the four researchers that have served as program coordinator over the years—Philipp Neumann, Benjamin Uekermann, and Severin Reiz—emphasizes the program SPPEXA itself. It provides an overview of the design and implementation of SPPEXA, it highlights its accompanying and supporting activities (internationalization, in particular with France and Japan; workshops; doctoral retreats; diversity-related measures), and it provides some statistics. It, thus, complements the papers from SPPEXA's research consortia collected in this volume.

1 Preparation

While super*computers* were recognized early as an important research infrastructure for German science and have been since then on the agenda (recommendations of the German Science Council (Wissenschaftsrat), introduction of the performance pyramid, Gauss Centre for Supercomputing, Gauss Alliance, NHR—Nationales Hochleistungsrechnen), the situation for super*computing* has always been quite different. First, the funds for HPC systems are typically limited to investments, i.e. the machinery; the current NHR initiative takes a more comprehensive view. Second, software development is frequently not considered as "science", which entails that neither typical projects in informatics or mathematics nor their counterparts in fields of application cover more than prototype development. Recently, BMBF's HPC software program and DFG's sustainable scientific software initiative, fortunately, have acknowledged the crucial role of software for HPC and support software development explicitly. Third, HPC software development has happened in Collaborative Research Centers or similar formats before, but mostly in an isolated way: an informatics initiative contained an HPC software project as an application, or a physics initiative contained a simulation- or HPC-oriented project. But all this hardly ever looked at more than one peculiar aspect at a time, and it was at most an interdisciplinary endeavor of two fields.

However, when Moore's law at least gets exhausted a bit and performance gains are more and more achievable through a more and more massive parallelism only, it is obvious that software and its performance and scalability play an increasingly crucial part. Therefore, the challenges at the eve of the exa-scale era required more—and that's actually what happened elsewhere, for example in the U.S. or in Japan: a significant, concerted initiative, bringing together informatics, mathematics, and several domains of application, comprising all relevant aspects of HPC software. That's where SPPEXA entered the stage.

2 Design Principles

SPPEXA was designed to provide a holistic approach to HPC software, comprising the aspects most relevant for ensuring the efficient use of current and upcoming high-end supercomputers, and to do this via exploring both evolutionary and disruptive research threads. Six research directions were identified as crucial ones: (1) Computational Algorithms, (2) Application Software, (3) System Software and Runtime Libraries, (4) Programming, (5) Software Tools, and (6) Data Management. Computational algorithms, such as fast linear solvers or eigensolvers, are a core numerical component of many large-scale application codes—both classical simulation-driven and recent data analytics-oriented ones. If scalability cannot be ensured here, the battle is already almost lost. Application software is the "user" of HPC systems, typically appearing as legacy codes that have been developed over many years. Increasing their performance via a co-design that addresses both the "systems—algorithms" and the "algorithms—applications/models" interfaces and combines algorithm and performance engineering is vital. Performance engineering can't succeed without progress in compilers, monitoring, code optimization, verification support, and parallelization support (such as auto-tuning)—which underlines the importance of system software and runtime libraries as well as of tools. Programming, including programming models, is probably the topic where the need for a balance of evolutionary research (improve and extend existing programming models, e.g.) and revolutionary approaches (explore new programming models, new language concepts such as Domain-Specific Languages) gets most obvious. Data management, finally, has always been HPC-relevant in terms of I/O or post-processing and visualization, and it is of ever-increasing importance since more and more HPC applications are on the data side.

To ensure the impact of this holistic idea, it was clear that having a set of projects in our Priority Program where some address this issue and others that one, and where they may collaborate or not, would not suffice. Therefore, SPPEXA's concept was to have a set of larger projects, or project consortia (research units—Forschergruppen), that would all have to address at least two of the six big topics with their research agenda; and that would all have to combine a relevant large-scale application with HPC-methodical advancements. This means that neither a merely domain-driven research ("improve my code, and this is a contribution to HPC in itself"), as we see it frequently in domain-driven research initiatives (Collaborative Research Centers in physics, life sciences, or engineering, e.g.), nor a generic purely algorithmic research ("if I improve my solver, this will help everyone"), as we see it frequently in mathematics- or informatics-driven research initiatives, would be allowed to find their place in SPPEXA. This was somewhat challenging, since we had to communicate this concept clearly and to convince potential applicants and reviewers that everyone should really comply with this agenda.

Furthermore, there is one property better known from Collaborative Research Centers than from Priority Programs: program-wide joint activities. For example, we wanted to have a vivid collaboration framework of cross-project workshops; networking with the big international programs; a focus on education also through fostering novel teaching formats or coding weeks and doctoral retreats for the doctoral candidates; gender-related activities to understand, evaluate and work towards a more gender-balanced research community; etc. This allowed for sharing mutual best practices in HPC for the mathematics- or informatics- or application-driven areas. Therefore, there was more coordination than we see in typical Priority Programs.

3 Funded Projects and Internal Structure

In the first funding phase, the following thirteen projects or project consortia were funded:[1]

CATWALK—A Quick Development Path for Performance Models. Felix Wolf (Darmstadt), Christian Bischof (Darmstadt), Torsten Hoefler (Zürich), Bernd Mohr (Jülich), and Gabriel Wittum (Frankfurt)

ESSEX—Equipping Sparse Solvers for Exa-scale. Gerhard Wellein (Erlangen), Achim Basermann (Köln), Holger Fehske (Greifswald), Georg Hager (Erlangen), and Bruno Lang (Wuppertal)

Exa-Dune—Flexible PDE Solvers, Numerical Methods, and Applications. Peter Bastian (Heidelberg), Olaf Ippisch (Clausthal), Mario Ohlberger (Münster), Christian Engwer (Münster), Stefan Turek (Dortmund), Dominik Göddeke (Stuttgart), and Oleg Iliev (Kaiserslautern)

ExaFSA—Exa-scale Simulation of Fluid-Structure-Acoustics Interactions. Miriam Mehl (Stuttgart), Hester Bijl (Delft), Sabine Roller (Siegen), Dörte Sternel (Darmstadt), and Thomas Ertl (Stuttgart)

EXAHD—An Exa-Scalable 2-Level Sparse Grid Approach for Higher-Dimensional Problems in Plasma Physics and Beyond. Dirk Pflüger (Stuttgart), Hans-Joachim Bungartz (München), Michael Griebel (Bonn), Markus Hegland (Canberra), Frank Jenko (Garching), and Hermann Lederer (Garching)

EXAMAG—Exa-scale Simulations of the Evolution of the Universe Including Magnetic Fields. Volker Springel (Heidelberg) and Christian Klingenberg (Würzburg)

ExaSolvers—Extreme-scale Solvers for Coupled Problems. Lars Grasedyck (Aachen), Wolfgang Hackbusch (Leipzig), Rolf Krause (Lugano), Michael Resch (Stuttgart), Volker Schulz (Trier), and Gabriel Wittum (Frankfurt)

EXASTEEL—Bridging Scales for Multiphase Steels. Daniel Balzani (Bochum), Axel Klawonn (Köln), Oliver Rheinbach (Freiberg), Jörg Schröder (Duisburg-Essen), and Gerhard Wellein (Erlangen)

ExaStencils—Advanced Stencil-Code Engineering. Christian Lengauer (Passau), Armin Größlinger (Passau), Ulrich Rüde (Erlangen), Harald Köstler (Erlangen), Sven Apel (Saarbrücken), Jürgen Teich (Erlangen), Frank Hannig (Erlangen), and Matthias Bolten (Wuppertal)

FFMK—A Fast and Fault-tolerant Microkernel-Based System for Exa-scale Computing. Hermann Härtig (Dresden), Alexander Reinefeld (Berlin), Amnon Barak (Jerusalem), and Wolfgang E. Nagel (Dresden)

GROMEX—Unified Long-range Electrostatics and Dynamic Protonation for Realistic Biomolecular Simulations on the Exa-scale. Helmut Grubmüller (Göttingen), Holger Dachsel (Jülich), and Berk Hess (Stockholm)

DASH—Smart Data Structures and Algorithms with Support for Hierarchical Locality. Karl Fürlinger (München), Colin W. Glass (Stuttgart), José Gracia (Stuttgart), and Andreas Knüpfer (Dresden)

Terra-Neo—Integrated Co-Design of an Exa-scale Earth Mantle Modeling Framework. Hans-Peter Bunge (München), Ulrich Rüde (Erlangen), Gerhard Wellein (Erlangen), and Barbara Wohlmuth (München)

[1]Some Principal Investigators have changed affiliation during the SPPEXA program. We specified the most recent main affiliation here.

After 3 years, twelve of those got a prolongation for the second funding phase, some with an "international extension" (bi-national with Japanese partners or tri-national with French and Japanese partners):

ESSEX-2—Equipping Sparse Solvers for Exa-scale. Gerhard Wellein (Erlangen), Achim Basermann (Köln), Holger Fehske (Greifswald), Georg Hager (Erlangen), Bruno Lang (Wuppertal), Tetsuya Sakurai (Tsukuba; Japanese partner), and Kengo Nakajima (Tokyo; Japanese partner)

Exa-Dune—Flexible PDE Solvers, Numerical Methods, and Applications. Peter Bastian (Heidelberg), Olaf Ippisch (Clausthal), Mario Ohlberger (Münster), Christian Engwer (Münster), Stefan Turek (Dortmund), Dominik Göddeke (Stuttgart), and Oleg Iliev (Kaiserslautern)

ExaFSA—Exa-scale Simulation of Fluid-Structure-Acoustics Interactions. Miriam Mehl (Stuttgart), Alexander van Zuijlen (Delft), Thomas Ertl (Stuttgart), Sabine Roller (Siegen), Dörte Sternel (Darmstadt), and Hiroyuki Takizawa (Tohoku; Japanese partner)

EXAHD—An Exa-Scalable 2-Level Sparse Grid Approach for Higher-Dimensional Problems in Plasma Physics and Beyond. Dirk Pflüger (Stuttgart), Hans-Joachim Bungartz (München), Michael Griebel (Bonn), Markus Hegland (Canberra), Frank Jenko (Garching), and Tilman Dannert (Garching)

EXAMAG—Exa-scale Simulations of the Magnetic Universe. Volker Springel (Heidelberg), Christian Klingenberg (Würzburg), Naoki Yoshida (Tokyo; Japanese partner), and Philippe Helluy (Strasbourg; French partner)

ExaSolvers—Extreme-scale Solvers for Coupeld Problems. Lars Grasedyck (Aachen), Rolf Krause (Lugano), Michael Resch (Stuttgart), Volker Schulz (Trier), Gabriel Wittum (Frankfurt), Arne Nägel (Frankfurt), Hiroshi Kawai (Tokyo; Japanese partner), and Ryuji Shioya (Toyo; Japanese partner)

EXASTEEL-2—Dual Phase Steels—From Micro to Macro Properties. Daniel Balzani (Bochum), Axel Klawonn (Köln), Oliver Rheinbach (Freiberg), Jörg Schröder (Duisburg-Essen), Olaf Schenk (Lugano), and Gerhard Wellein (Erlangen)

ExaStencils—Advanced Stencil-Code Engineering. Christian Lengauer (Passau), Ulrich Rüde (Erlangen), Harald Köstler (Erlangen), Sven Apel (Saarbrücken), Jürgen Teich (Erlangen), Frank Hannig (Erlangen), Matthias Bolten (Wuppertal), and Shigeru Chiba (Tokyo; Japanese partner)

FFMK—A Fast and Fault-tolerant Microkernel-Based System for Exa-scale Computing. Hermann Härtig (Dresden), Alexander Reinefeld (Berlin), Amnon Barak (Jerusalem), and Wolfgang E. Nagel (Dresden)

GROMEX—Unified Long-range Electrostatics and Dynamic Protonation for Realistic Biomolecular Simulations on the Exa-scale. Helmut Grubmüller (Göttingen), Holger Dachsel (Jülich), and Berk Hess (Stockholm)

DASH—Smart Data Structures and Algorithms with Support for Hierarchical Locality. Karl Fürlinger (München), Colin W. Glass (Stuttgart), José Gracia (Stuttgart), and Andreas Knüpfer (Dresden)

Terra-Neo—Integrated Co-Design of an Exa-scale Earth Mantle Modeling Framework. Hans-Peter Bunge (München), Ulrich Rüde (Erlangen), and Barbara Wohlmuth (München)

Furthermore, four new project consortia joined SPPEXA:

ADA-FS—Advanced Data Placement via Ad-hoc File Systems at Extreme Scales. Wolfgang E. Nagel (Dresden), André Brinkmann (Mainz), and Achim Streit (Karlsruhe)

AIMES—Advanced Computation and I/O Methods for Earth-System Simulations. Thomas Ludwig (Hamburg), Thomas Dubos (Versailles; French partner), Naoya Maruyama (RIKEN; Japanese partner), and Takayuki Aoki (Tokyo; Japanese partner)

ExaDG—High-order Discontinuous Galerkin for the Exa-scale. Guido Kanschat (Heidelberg), Katharina Kormann (München), Martin Kronbichler (München), and Wolfgang A. Wall (München)

MYX—MUST Correctness Checking for YML and XMP Programs. Matthias S. Müller (Aachen), Serge Petiton (Lille; French partner), Nahid Emad (Versailles; French partner), Taisuke Boku (Tsukuba; Japanese partner), and Hitoshi Murai (RIKEN; Japanese partner)

Finally, 1 year later, a seventeenth project joined SPPEXA as associated project:

ExtraPeak—Automatic Performance Modeling of HPC Applications. Felix Wolf (Darmstadt) and Torsten Hoefler (Zürich)

Hence, overall, there have been four Japanese-German and three French-Japanese-German consortia within SPPEXA. On the German side, an overall sum of 57 principal investigators from 39 institutions have been involved, representing informatics (25), mathematics (19), engineering (8), natural sciences (4), and life sciences (1).

Concerning governance, SPPEXA was headed by its two Spokespersons Hans-Joachim Bungartz (Technical University of Munich—TUM) and Wolfgang E. Nagel (Technical University of Dresden). For the everyday organization, a Program Coordinator (in chronological order: Benjamin Peherstorfer, now professor at New York University; Philipp Neumann, now professor at Helmut-Schmidt-University Hamburg; Benjamin Uekermann, now with Eindhoven University of Technology; and Severin Reiz, TUM) as well as an Office were established (both at TUM). Strategic decisions in SPPEXA were taken by the Steering Committee, consisting of H.-J. Bungartz, W. E. Nagel, as well as Sabine Roller (Siegen), Christian Lengauer (Passau), Hans-Peter Bunge (München), Dörte Sternel (Darmstadt), and—in the second funding phase—Nahid Emad (France) and Takayuki Aoki (Japan). Finally, a Scientific Advisory Board supported our activities and planning: George Biros (University of Texas at Austin), Rupak Biswas (NASA), Klaus Becker (Airbus), Rob Schreiber (at that time HP Labs), and Craig Stewart (University of Indiana at Bloomington).

4 SPPEXA Goes International

Extreme-scale HPC has always been an international endeavor. In 2010, as the first call in the framework of the G8 Research Councils' Initiative on Multilateral Research Funding, the topic Application Software towards Exa-scale Computing for Global Scale Issues had been selected. In the sequel of that initiative, the idea arose to give SPPEXA in its second funding phase a more international flavor, beyond the individual international partners present in some of the consortia. DFG's head office contacted several of their partner institutions in other countries. While it turned out to be complicated to synchronize activities with the National Science Foundation (NSF) in the U.S., the discussions with the French Agence Nationale de la Recherche (ANR) and the Japan Science and Technology Agency (JST) became very concrete. Finally, for the first time, a funding phase of a complete DFG Priority Program was linked to funding formats from two other countries, and the three agencies combined their forces in a joint call run by DFG. Due to formal restrictions, two new types of SPPEXA consortia were open for application: bi-national Japanese-German or tri-national French-Japanese-German ones.

Overall, the following French institutions participated in SPPEXA projects: Université de Versailles, Université de Strasbourg, and Maison de la Simulation, Saclay. From the Japanese side, the involved partner institutions involved were RIKEN, Tokyo University of Technology, University of Tsukuba, University of Tokyo, Tohoku University, Tokyo University of Science, and Toyo University. Beyond research in the single consortia, one SPPEXA doctoral retreat was held in France, and SPPEXA co-organized three French-Japanese-German workshops—the first one 2017 in the French embassy in Tokyo, the second one in 2018 in the German embassy in Tokyo, and the third one in 2019, again in the French embassy. The first two focused on exa-scale computing, while the third one did a move towards artificial intelligence (AI) and, in particular, addressed the convergence of AI and HPC.

Further internationalization measures were the SPPEXA guest program, the research stays for doctoral candidates (up to 3 months; overall 25 taken in funding phase 2), and our PR activities at the big international meetings. For example, SPPEXA organized panels or sessions at the Supercomputing Conference (SC) and the International Supercomputing Conference (ISC HPC) and participated in the session and poster exhibition on DFG-funded collaborative research at DATE 2019.

5 Joint Coordinated Activities

As mentioned above, SPPEXA featured a rich program of joint cross-consortium activities (the following numbers refer to funding phase 2, 2016–2019):

Guests Overall, more than 85 guest researchers visited one or more SPPEXA projects.

Workshops Workshops were a particular format to foster exchange and collaboration across project consortia. Central funds had been established for that, and each SPPEXA PI could hand in proposals (two calls per year). The proposal had to depict how the cross-consortium effect was to be ensured (more than one organizing consortium, etc.). Overall, 41 SPPEXA workshops, held at conferences or standalone, were supported via this channel.

Doctoral Retreats The SPPEXA Doctoral Retreat had two main goals—first, to offer an additional educational component to our doctoral candidates; second, to overcome the sometimes narrow borders of research by connecting with international researchers on a doctoral level (guest lectures, own contributions, hands-on sessions, . . .). Overall, three doctoral retreats were organized: Strasbourg (2016), Dresden (2017), and Wuppertal (2018).

Doctoral Research Stays Following the successful model of TUM Graduate School, where each doctoral candidate university-wide can get funds for an international research stay of up to 3 months, we encouraged our doctoral candidates SPPEXA-wide to enrich their PhD phase with such an international component. Overall, 25 such research interns were funded, examples for destinations being ETH Zurich, NORCE Bergen, or University of Tennessee.

Gender Activities Looking at the gender situation in HPC, it is obvious that the presence of women is even worse than in general in informatics. To improve that situation and to provide a more open atmosphere, a couple of measures were taken. At every Annual Plenary Meeting (2016, 2017, 2018, and 2019), we organized gender trainings by external coaches to raise awareness of gender biases in academia, each with 25 participants. Additionally, SPPEXA members organized workshop-like events such as student MINT mentoring days (2016–2018) and women's networking events in 2019. Moreover, we connected to industry (Bosch and IBM) via gender bias discussion days called "Equality at Exascale". Exceptional at this event was that not only women participated, but we had an ideal gender-parity in participants.

Impact on Education As a side effect, HPC education also got a boost by SPPEXA. Numerous lectures and lab courses were updated, and a lot of student theses had topics directly related to SPPEXA projects.

Prizes During the second phase of SPPEXA, every year, the best student and doctoral theses SPPEXA-wide were awarded a prize. Over the years, the winners were:

2016: Klaudius Scheufele (Stuttgart, master's thesis) and Benjamin Uekermann (Munich, PhD thesis);
2017: Sebastian Schweikl (Passau, bachelor's thesis), Simon Schwitanski (Aachen, master's thesis), and Moritz Kreutzer (Erlangen, PhD thesis);
2018/2019: Piet Jarmatz (Munich/Hamburg, master's thesis) and Sebastian Kuckuk and Christian Schmitt (Erlangen, PhD thesis).

Support of Young Researchers For sustainability in academia, supporting young aspiring researchers is indispensable. We took measures by funding research stays for doctoral candidates and awarding prizes for exceptional theses. Additionally, we also supported bachelor and master students for the student cluster competition at the (international) supercomputing conferences SC and ISC HPC 2016–2019.

Public Relations Dissemination of research becomes more and more important. Continuing efforts from the first phase, SPPEXA featured articles in the InSiDE magazine, published by the GAUSS Center for Supercomputing, twice per year in 2016, 2017, and 2018 introducing one project each time. Furthermore, starting 2018, SPPEXA contributed five articles to the online platform Science Node.[2] Last, in 2018, SPPEXA also featured an article in the EU Research magazine.

Internationalisation See previous Sect. 4.

6 HPC Goes Data

The computational revolution goes on! Computers and sophisticated computational methods have shaped the "third paradigm", the third path to insight in science, complementing the classical approaches, theory and experiment, but also building a bridge and providing the missing link between those two. An early incarnation of "computational" were numerical simulations, later expanded by so-called "outer-loop scenarios", in which repeated simulations allow for enhanced results: optimization, parameter identification, stochastics, or uncertainty quantification. All of this, basically, was model-driven, following a deductive regime of model hypotheses and derivations from them. The latest appearance of "computational" can be characterized by the focus on data: data-enhanced simulation, data analytics, machine learning, or artificial intelligence. Instead of being based on models, this approach is much more data-driven, following an inductive regime of collecting data and drawing conclusions from them. In simplified words, the "data from models" turned into, or was complemented by, a "models from data". Despite that shift of focus, the basic underlying principle did not change: state-of-the-art computer systems and state-of-the-art computational methods are combined and used to advance the frontier of science. Something new is maybe the fact that the club of scientific domains that benefit from the "third paradigm" has become bigger: While numerical simulation was, more or less, driven by natural, engineering, and life sciences, the data-centered approach comprises all domains, including social sciences and humanities.

Of course, this development has a huge impact on HPC. In particular, new fields and new types of applications popped up, as well as new lines of architectures and systems. For example, in 2018, the majority of finalists for the Gordon Bell

[2] https://sciencenode.org/feature/the-race-to-exascale.php.

Award, the most renowned prize in HPC, already had a significant amount of machine learning in their papers. World-wide, HPC centers observe an increasing share of data-driven jobs on their machines. This is not surprising: as science and science methodology evolve, the kind of studies done in that context also does. Despite all those changes, the role of HPC is astonishingly stable: HPC is a core enabling technology of "computational". It was and still is an enabler of numerical simulation, and it has become a crucial enabler of data analytics and artificial intelligence. If artificial intelligence, machine learning, or deep learning have become so popular recently, this is much more due to the fact that established methodology can succeed due to HPC, than due to new AI/ML/DL methodology itself.

These developments are also visible at the end of SPPEXA. Several consortia already are on that "data-driven track", as, for example, our third French-Japanese-German workshop in Tokyo showed.

7 Shaping the Landscape

When SPPEXA started in 2013, the core idea was to significantly improve algorithms, software, and tools, in order to be prepared for the exa-scale age. In the meantime, we are at the eve of exa-scale systems, as the co-design developments in the U.S. and in Japan (Fugaku) or the discussions in the European Union on exa-scale and pre-exa-scale systems show. And research in SPPEXA has definitely contributed to the application landscape in Germany being much closer to "exa-scale-readiness" than before. Several leading application software packages were involved, and significant progress in terms of scalability and parallel efficiency could be achieved. Furthermore, and maybe even more important, the SPPEXA consortia showed the advantages of the multi-disciplinary engagement, brought together a lot of groups and ideas disconnected before, and, thus, justified the concept of larger, cross-institutional, and cross-disciplinary teams instead of single-PI projects.

The visibility SPPEXA got is stunning. SPPEXA was present at the leading international conferences (Euro-Par, Supercomputing, ISC HPC)—through individual presentations and special events, such as minisymposia or panels. But also at "neighboring" events, such as the DATE 2019 (Design, Automation, and Test in Europe), SPPEXA had a presentation slot and a booth. SPPEXA was involved in the activities (workshops, white papers, etc.) of the BDEC Community (Big Data and Extreme-Scale Computing) as well as in the organization of the Long Program "Science at Extreme Scales: Where Big Data Meets Large-scale Computing" at the Institute for Pure and Applied Mathematics (IPAM) in Los Angeles, and it co-organized a French-Japanese-German workshop series in Tokyo (cf. the section on internationalization). Thus, at an international scale, SPPEXA was generally perceived as the "German player" in the HPC software concert.

8 Concluding Remarks

Without any doubt, SPPEXA has written a success story: in terms of its research, concerning the innovative funding format, with its multi-disciplinary approach, its multi-national facets, and—last, but not least—its huge visibility. We are grateful for all the support we got from the German Research Foundation (DFG): the funding, but also for the encouragement during the preparation of SPPEXA and the continued advice during its runtime.

Appendix 1: Qualification

The following achievements have been completed in the SPPEXA program within 1.1.2016 and 30.04.2020:

Projects	Completed PhD theses	Completed habilitations	Calls to professorship
AIMES	0	0	1
ADA-FS	0	0	0
DASH	1	0	1
ESSEX	1	1	0
ExaDG	4	0	1
Exa-Dune	4	0	1
ExaFSA	2	0	0
EXAHD	3	0	0
EXAMAG	9	0	0
ExaSolvers	1	0	1
EXASTEEL	2	0	1
ExaStencils	5	2	4
ExtraPeak	3	0	0
FFMK	1	0	0
GROMEX	2	0	0
MYX	0	0	0
Terra-Neo	3	0	0
Coordination	1	1	2
Overall	43	3	12

The previous table follows the DFG requirements for final reports in priority programs. At least 25 additional PhD candidates are close to being finished; however, due to the lengthy defense procedure they are not counted here.

Also, please take into account that project consortia vary in size (regarding Principal Investigators and PhD candidates) and their start/end date.

Appendix 2: Software from Project Consortia

In the following, a table with links to software that has been developed by the project consortia in SPPEXA Phase-II is given.

Project	Software developed
AIMES	SCIL
	github.com/JulianKunkel/scil
ADA-FS	GekkoFS
	tu-dresden.de/zih/forschung/projekte/ada-fs
DASH	DASH
	www.dash-project.org/
ESSEX	PHIST, GHOST, CRAFT, RACE, ScaMaC
	bitbucket.org/essex/{`PHIST, ..., RACE, matrixcollection`}
ExaDG	deal.II
	github.com/dealii/dealii
EXA-Dune	DUNE
	gitlab.dune-project.org/exadune
ExaFSA	preCICE
	github.com/precice/precice
	Ateles
	apes.osdn.io/pages/ateles
EXAHD	SG++
	github.com/SGpp/SGpp
EXAMAG	AREPO
	arepo-code.org/
ExaSolvers	utopia
	bitbucket.org/zulianp/utopia/src/master/
EXASTEEL	FE2TI
	www.numerik.uni-koeln.de/14079.html
ExaStencils	LFA Lab
	hrittich.github.io/lfa-lab/
	ExaSlang
	i10git.cs.fau.de/exastencils/release
ExtraPeak	Extra-P
	www.scalasca.org/scalasca/software/extra-p/
FFMK	FFMK
	ffmk.tudos.org/
GROMEX	GROMACS
	www.gromacs.org

MYX	MUST
	doc.itc.rwth-aachen.de/display/CCP/Project+MUST
Terra-Neo	HyTeG
	i10git.cs.fau.de/hyteg/hyteg
	waLBerla
	www.walberla.net/
	TerraNeo
	terraneo.fau.de/

Appendix 3: Project Consortia Key Publications

This volume represents a continuation of the corresponding report in SPPEXA Phase-I, which is referenced several times in the text above:

1. Bungartz, H.-J., Neumann, P., Nagel, W.E.: *Software for Exascale Computing-SPPEXA 2013–2015*, vol. 113. Springer, Berlin (2016)

SPPEXA Phase-II showed visibility in the research community with numerous publications. In the following we provide a list of two key publications for each project consortium:[3]

AIMES

1. Jum'ah, N., Kunkel, J.: Performance portability of earth system models with user-controlled GGDML code translation. In: International Conference on High Performance Computing, pp. 693–710. Springer, Berlin (2018)
2. Kunkel, J., Novikova, A., Betke, E., Schaare, A.: Toward decoupling the selection of compression algorithms from quality constraints. In: International Conference on High Performance Computing, pp. 3–14. Springer, Berlin (2017)

ADA-FS

1. Vef, M.A., Moti, N., Süß, T., Tocci, T., Nou, R., Miranda, A., Cortes, T., Brinkmann, A.: GekkoFS—a temporary distributed file system for HPC applications. In: 2018 IEEE International Conference on Cluster Computing (CLUSTER), pp. 319–324. IEEE, Piscataway (2018)
2. Soysal, M., Berghoff, M., Klusáček, D., Streit, A.: On the quality of wall time estimates for resource allocation prediction. In: Proceedings of the 48th International Conference on Parallel Processing: Workshops, pp. 1–8. ACM, New York (2019)

DASH

1. Kowalewski, R., Jungblut, P., Fürlinger, K.: Engineering a distributed histogram sort. In: 2019 IEEE International Conference on Cluster Computing (CLUSTER), pp. 1–11. IEEE, Piscataway (2019)

[3]Following the DFG requirements for final reports in priority programs.

2. Fürlinger, K., Glass, C., Gracia, J., Knüpfer, A., Tao, J., HHünichnich, D., Idrees, K., Maiterth, M., Mhedheb, Y., Zhou, H.: DASH: data structures and algorithms with support for hierarchical locality. In: European Conference on Parallel Processing, pp. 542–552. Springer, Berlin (2014)

ESSEX

1. Pieper, A., Kreutzer, M., Alvermann, A., Galgon, M., Fehske, H., Hager, G., Lang, B., Wellein, G.: High-performance implementation of Chebyshev filter diagonalization for interior eigenvalue computations. J. Comput. Phys. **325**, 226–243 (2016)
2. Röhrig-Zöllner, M., Thies, J., Kreutzer, M., Alvermann, A., Pieper, A., Basermann, A., Hager, G., Wellein, G., Fehske, H.: Increasing the performance of the Jacobi–Davidson method by blocking. SIAM J. Sci. Comput. **37**(6), C697–C722 (2015)

ExaDG

1. Kronbichler, M., Kormann, K.: Fast matrix-free evaluation of discontinuous Galerkin finite element operators. ACM Trans. Math. Softw. **45**(3), 1–40 (2019)
2. Fehn, N., Wall, W.A., Kronbichler, M.: Efficiency of high-performance discontinuous Galerkin spectral element methods for under-resolved turbulent incompressible flows. Int. J. Numer. Methods Fluids **88**(1), 32–54 (2018)

EXA-Dune

1. Bastian, P., Engwer, C., Göddeke, D., Iliev, O., Ippisch, O., Ohlberger, M., Turek, S., Fahlke, J., Kaulmann, S., Steffen Müthing, S., et al.: EXA-DUNE: flexible PDE solvers, numerical methods and applications. In: European Conference on Parallel Processing, pp. 530–541. Springer, Berlin (2014)
2. Engwer, C., Altenbernd, M., Dreier, N.A., Göddeke, D.: A high-level C++ approach to manage local errors, asynchrony and faults in an MPI application. In: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), pp. 714–721. IEEE, Piscataway (2018)

ExaFSA

1. Mehl, M., Uekermann, B., Bijl, H., Blom, D., Gatzhammer, B., Van Zuijlen, A.: Parallel coupling numerics for partitioned fluid–structure interaction simulations. Comput. Math. Appl. **71**(4), 869–891 (2016)
2. Totounferoush, A., Pour, N.E., Schröder, J., Roller, S., Mehl, M.: A new load balancing approach for coupled multi-physics simulations. In: 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 676–682. IEEE, Piscataway (2019)

EXAHD

1. Obersteiner, M., Hinojosa, A.P., Heene, M., Bungartz, H.J., Pflüger, D.: A highly scalable, algorithm-based fault-tolerant solver for gyrokinetic plasma simulations. In: Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, pp. 1–8 (2017)

2. Hupp, P., Heene, M., Jacob, R., Pflüger, D.: Global communication schemes for the numerical solution of high-dimensional PDEs. Parallel Comput. **52**, 78–105 (2016)

ExaSolvers

1. Benedusi, P., Garoni, C., Krause, R., Li, X., Serra-Capizzano, S.: Space-time FE-DG Discretization of the anisotropic diffusion equation in any dimension: the spectral symbol. SIAM J. Matrix Anal. Appl. **39**(3), 1383–1420 (2018)
2. Kreienbuehl, A., Benedusi, P., Ruprecht, D., Krause, R.: Time-parallel gravitational collapse simulation. Commun. Appl. Math. Comput. Sci. **12**(1), 109–128 (2015)

ExaStencils

1. Köstler, H., Schmitt, C., Kuckuk, S., Kronawitter, S., Hannig, F., Teich, J., Rüde, U., Lengauer, C.: A scala prototype to generate multigrid solver implementations for different problems and target multi-core platforms. Int. J. Comput. Sci. Eng. **14**(2), 150–163 (2017). https://doi.org/10.1504/IJCSE.2017.082879
2. Schmitt, C., Kronawitter, S., Hannig, F., Teich, J., Lengauer, C.: Automating the development of high-performance multigrid solvers. Proc. IEEE **106**(11), 1969–1984 (2018)

ExtraPeak

1. Shudler, S., Calotoiu, A., Hoefler, T., Wolf, F.: Isoefficiency in practice: configuring and understanding the performance of task-based applications. In: ACM SIGPLAN Notices, vol. 52, pp. 131–143. ACM, New York (2017)
2. Calotoiu, A., Hoefler, T., Poke, M., Wolf, F.: Using automated performance modeling to find scalability bugs in complex codes. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, p. 45. IEEE, Piscataway (2013)

FFMK

1. Weinhold, C., Lackorzynski, A., Härtig, H.: FFMK: an HPC OS based on the L4Re microkernel. In: Operating Systems for Supercomputers and High Performance Computing, pp. 335–357. Springer, Berlin (2019)
2. Gholami, M., Schintke, F.: Multilevel checkpoint/restart for large computational jobs on distributed computing resources. In: IEEE 38th Symposium on Reliable Distributed System (SRDS) (2019)

GROMEX

1. Beckmann, A., Kabadshow, I.: Portable node-level performance optimization for the fast multipole method. In: Recent Trends in Computational Engineering-CE2014, pp. 29–46. Springer, Berlin (2015)
2. Kutzner, C., Páll, S., Fechner, M., Esztermann, A., de Groot, B.L., Grubmüller, H.: More bang for your buck: Improved use of GPU nodes for GROMACS 2018. J. comput. chem. **40**(27), 2418–2431 (2019)

MYX

1. Protze, J., Tsuji, M., Terboven, C., Dufaud, T., Murai, H., Petiton, S., Emad, N., Müller, M., Boku, T.: Myx—runtime correctness analysis for multi-level parallel programming paradigms. In: Software for Exascale Computing: SPPEXA 2016–2019. Lecture Notes in Computational Science and Engineering. Springer, Berlin (2020)
2. Protze, J., Schulz, M., Ahn, D.H., Müller, M.S.: Thread-local concurrency: a technique to handle data race detection at programming model abstraction. In: Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing, pp. 144–155 (2018)

Terra-Neo

1. Bauer, S., Huber, M., Ghelichkhan, S., Mohr, M., Rüde, U., Wohlmuth, B.: Large-scale simulation of mantle convection based on a new matrix-free approach. J. Comput. Sci. **31**, 60–76 (2019)
2. Huber, M., Gmeiner, B., Rüde, U., Wohlmuth, B.: Resilience for massively parallel multigrid solvers. SIAM J. Sci. Comput. **38**(5), S217–S239 (2016)

A Perspective on the SPPEXA Collaboration from France

Nahid Emad

As the French member of the Steering Committee of SPPEXA, it is my great pleasure to give a short address to this volume from the perspective of the French partners in this German-French-Japanese cooperation. To highlight the types of software supported by SPPEXA, we first present a classification of high-performance software types. We then take a look at the recent activities of HPC software in France under the SPPEXA umbrella. Next, some local impacts of the SPPEXA collaboration on the French HPC community is provided, and lastly, an outlook to future collaborations.

1 HPC Software in Three Phases

High-performance numerical software targets at obtaining relevant scalability in space and time for large-size applications by using a large number of cores/processors/nodes of powerful computers. They can be classified into three phases: pre-treatment, treatment, and post-treatment. Obviously, such a software often belongs to more than one of the categories mentioned.

Pre-processing Software The precise definition of these phases depends on the context, but the main role of pre-treatment software is the preparation of the input data for the treatment phase. This preparation sometimes consists of a rather complex parallel algorithmic and programming processing. Big data compression, uniform data formatting, and conditioning improvement of data matrices are some examples of the pre-treatment phase.

N. Emad (✉)
University of Paris Saclay, UVSQ, LI PARAD, Maison de la Simulation, Versailles, France
e-mail: nahid.emad@uvsq.fr

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_2

Treatment Software The HPC software in the processing phase concerns mainly high-performance numerical simulation of physical phenomena, social networks, etc. These softwares could be classified as (1) "standard" libraries and (2) ad-hoc libraries done by application field programmers, which implement their application by making use of building blocks (partially or totally) of the libraries of class (1). The latter implement numerical methods with the main parallel/distributed programming methodologies, such as ScaLAPACK, PETSc, SLEPc, ATLAS, etc. In the fields of application , HPC ad-hoc software targets epidemiology, electromagnetism, gamma astronomy, safety, or health and nutrition. Some ad-hoc software examples are CEDRE which targets simulation for energy and propulsion, CELESTIA and STELLARIUM which is a space simulator and a planetarium for observing the solar system and the rest of the universe in real time and in 3D, GAUSS, which is a flexible platform for data analysis, and AREPO, which is a cosmological hydrodynamical simulation code on a dynamic unstructured mesh.

Post-processing Software Post-processing software essentially involves the analysis, visualization, and performance evaluation of the treatment phase results. Some examples of such software are ParaView (a multi-platform data analysis and visualization application), VisIt (an interactive platform for visualization, animation and data analysis), MAQAO (sets of software tools for code optimization in the core or node level of a parallel architecture), and Maya (a software for modeling, simulation, and 3D animation).

All these software packages generally translate a physical phenomenon, social behaviour, etc. into mathematical equations. Their high-performance implementation on parallel and/or distributed systems is a delicate task and requires a huge ecosystem with people having interdisciplinary skills. This makes the existence and use of accompanying software necessary, which provides the logistics of high-performance computing. These software frameworks provide the environment for high-performance programming and often conceal the complexity of underlying parallel and/or distributed architectures. As a consequence this allows the users to focus on main objectives. MPI, OpenMP, Globus, Condor, XMP, YML, MUST, etc. are very few examples of this kind of software.

2 Trilateral Projects in SPPEXA and Their Impact

SPPEXA targeted fundamental research on different aspects of HPC software and covered software categories cited before with a co-design approach. Thanks to ANR, DFG, and JST, the trilateral French/German/Japanese projects have been funded within SPPEXA. Some of these projects, such as MYX, have benefited from pre-existing bilateral collaborations. This allowed a dynamical and productive work from the beginning and for a rapid progress towards the objectives set. In addition to project meetings, the cross-project SPPEXA workshops have given a new dynamic to the trilateral collaborations paving the way for the organization of other conferences, workshops or seminars.

The EXAMAG (Exascale Simulations of the Magnetic Universe) project is an example of an SPPEXA trilateral French/German/Japanese project with the aim of improving the astrophysical moving-mesh code AREPO and extending its range of applicability for high scale computing platforms. EXAMAG is an ad-hoc HPC software of the processing category of the classification given in the previous section.

MYX (MUST Correctness Checking for YML and XMP Programs) also is a trilateral French/German/Japanese project which aims to offer a guideline how to limit the risk to introduce errors and how to best express the parallelism to catch errors at runtime. From a practical viewpoint, MYX aims at the design and the application of a scalable correctness checking tool MUST to YML and XMP. In the MYX project, the main developed software packages (YML, XMP and MUST) belong to the last category of HPC software; the ones providing the logistics of high-performance application programs. However, in order to validate the design and development of these softwares, many other benchmarks and/or real applications are developed. Among them are the multiple restarted Krylov methods/HPCS ad-hoc, matrix generator/pretreatment , epidemic HPCS ad-hoc, etc.

The SPPEXA funding of workshops with several projects involved added an extra dimension of interdisciplinarity. In collaboration with the DASH and ESSEX-II projects, MYX members organized four trilateral (German/Japanese/French) workshops. Two of them have been hold at university of Paris Saclay/Versailles in France. With the prominent invited speakers and the talks of SPPEXA-involved project members, these workshops have been very attractive (40 and 60 attendees, respectively). An important number of indirect outcomes of SPPEXA activities (workshops, "open" trilateral meetings, doctoral retreats, etc.) generated new connections between German and French colleagues and students. A few examples are ▷ the review of the PhD dissertation of a non-SPPEXA funded French student by Sabine Roller, professor at Siegen University, Germany, ▷ Xinzhe Wu, who finished recently his PhD, funded by ANR part of ANR/DFG/JST MYX project and, who is currently in a post-doctoral position at *Jülich Research Centre* in Germany, ▷ M.A. Diop, currently PhD student, funded by a French CIFRE followship (with ATOS/EVIDIAN company), who participated in SPPEXA doctoral retreats as well as several SPPEXA workshops, ▷ a workhop organised by Sabine Roller and Nahid Emad at HPC Asia, or a large number of BSc and MSc students benefiting from the collaboration.

3 What Will Be Next?

The on-going convergence between machine learning, data analysis, and high-performance computing is creating new algorithmic and co-design approaches that need to be taken into account for the future. With the three tri-national workshops in Tokyo, SPPEXA has contributed to this on-going development, and we are all looking forward to a continued collaboration in the future.

A Perspective on the SPPEXA Collaboration from Japan

Takayuki Aoki

A national research project was running in Japan from 2010 to 2017 named "Development of System Software Technologies for Post-Peta Scheme High Performance Computing" (so called Post-Peta CREST) . It was supported by JST (Japan Science and Technology Agency), which is the Japanese counterpart to DFG ("Deutsche Forschungsgemeinschaft"). The Post-Peta CREST project was similar to the first funding phase of SPPEXA in the sense that it had a primarily national scope. Then, the Post-Peta CREST project opened up to international collaboration, and some projects were extended for two more years, where they formed collaborative research groups with SPPEXA phase-II projects. Projects with contributions from Japan are *ExaFSA*, *ExaStencils*, *EXAMAG*, *ESSEX-II*, *EXASOLVERS*, *AIMES*, and *MYX*, with more than 10 researchers in the second phase of SPPEXA. To highlight the success of the Japanese collaboration with SPPEXA, we have a brief look at two working groups.

ppOpen-HPC and ESSEX-II

As a part of Post-Peta CREST projects between 2011 and 2015, a group at the University of Tokyo developed ppOpen-HPC, which is an open source infrastructure for the development and execution of optimized and reliable simulation code on post-peta-scale (pp) parallel computers based on many-core architectures. The framework covers various types of procedures for scientific computations in various types of computational models, such as FEM, FDM, FVM, BEM, and DEM.

T. Aoki (✉)
Tokyo Institute of Technology, Tokyo, Japan
e-mail: taoki@gsic.titech.ac.jp

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_3

Automatic tuning (AT) technology enables automatic generation of optimized libraries and applications under various types of environments. The most updated version of ppOpen-HPC was released as open source software, which is available at https://github.com/Post-Peta-Crest/ppOpenHPC.

In 2016, the team of ppOpen-HPC joined the SPPEXA phase-II project ESSEX-II including members from the University of Erlangen-Nuremberg, which is funded by JST-CREST and SPPEXA under Japan (JST)-Germany (DFG) collaboration until 2018. ESSEX-II developed pK-Open-HPC (extended version of ppOpen-HPC, a framework for exa-feasible applications), such as preconditioned iterative solvers for quantum science.

Sparse coefficient matrices derived from applications in quantum science have generally relatively very small diagonal components, and they are generally ill-conditioned. Therefore, it is difficult to apply preconditioned iterative methods developed in ppOpen-HPC directly to such applications. The ESSEX-II team developed a regularization method for robustness based on blocking and diagonal shifting, which provide efficient and robust convergence of ill-conditioned problems in quantum science. Preconditioning methods with the regularization method are implemented in GHOST/PHIST libraries for solving matrices, which integrates all linear solvers and related methods developed in ESSEX/ESSEX-II projects. Moreover, they proposed a new method for global parallel reordering, which provides robust and efficient convergence of parallel iterative solvers with ILU-based preconditioning for very ill-conditioned problems. The developed method kept iteration number constant in strong scaling cases up to O(104) MPI processes for very ill-conditioned problems. This is the first method for global parallel reordering.

In the ESSEX-II project, CRAFT (A library for application-level Check-point/Restart and Automatic Fault Tolerance) has been developed for fault resilience on exascale systems by checkpointing. ESSEX-II integrated the dynamic load balancing function and CRAFT, and developed a prototype of a fault-resilient framework for parallel FEM applications. Parallel FEM codes can continue computations by this framework, when some of the computing nodes fail. This framework does not need spare nodes for fault resilience. This idea can be extended to various types of procedures for dynamic scheduling on exascale systems.

Collaborations in ESSEX-II project have been continuing in the JHPCN projects (“Numerical Library with High-Performance/Adaptive-Precision/High-Reliability” (starting in 2018), “Innovative Multigrid Methods” (starting in 2018)), and in “Innovative Methods for Scientific Computing in the Exascale Era by Integrations of (Simulation+Data+Learning)” funded by “Grant-in-Aid for Scientific Research (S) (KAKENHI S)” (2019-2023)

Xevolver and ExaFSA

The so-called Xevolver project is one of the Post-Peta CREST projects from 2011 to 2017. A group at the Tohoku University discussed how they could help in legacy code migration to future-generation extreme-scale computing systems that will be massively parallel and heterogeneous. Even today an HPC application code is likely optimized assuming a particular system configuration, and hence specialized only for its target system. In general, such an application is not performance-portable at all. As the HPC system architectures are now diverging and also getting more complicated in terms of accelerators, it will require more time and effort to migrate or re-optimize the code to another system in the future. To make matters worse, system-specific code optimizations are tightly interwoven with the computation and thereby degrade the code readability and maintainability, even though HPC applications need to evolve not only for achieving high performance, but also for advancing computational science. Therefore, in the project, our team has developed a code transformation framework, Xevolver, so that users can define their own code transformations and thus express system-specific code optimizations as code transformation rules. Since code transformation rules can be defined separately from application codes themselves, the Xevolver framework can contribute to separation of system-specific performance concerns from application codes, and hence prevent overcomplicating the codes.

In 2016, core members of the Xevolver research team joined the second phase of the ExaFSA project in order to demonstrate that the Xevolver approach is effective for optimizing real-world applications in practice. The Xevolver approach assumes that an HPC application is developed by a team work of at least two kinds of programmers. One is application developers and the other is performance engineers. Application developers are interested in simulation results rather than performance, while performance engineers are mainly focusing on sustained simulation performance. Therefore, Japanese researchers have worked as performance engineers using Xevolver by considering German research groups as application developers.

The ExaFSA project focused on engineering two solvers, FASTEST and Ateles, which have been developed in the ExaFSA project as primary building blocks of a practical coupled simulation. An incompressible flow solver, FASTEST, has a long history of development and was once optimized for classic vector machines. Thus, some of important kernels still have two versions, default version and its vector-optimized version. In the ExaFSA project, hence, they used the Xevolver framework to express the differences between the two versions, and demonstrated that the vector-optimized version can be generated by transforming the default version. That is, the Xevolver approach can express the system-specific code optimizations as code transformation rules, and thus even simplify the code while achieving high performance and portability. Ateles is based on based on Discontinuous Galerkin (DG) discretization method, and a part of the simulation framework, APES, was developed at the University of Siegen in Germany. Unlike FASTEST, Ateles is written using modern Fortran language features to hide the implementation details. However, the kernel loops still need to be optimized in different ways for individual system architectures to achieve high performance. For example, some loop optimizations with compiler directives are mandatory for the NEC SX-ACE vector computing system to properly vectorize and thus efficiently execute the loops. In this project, Xevolver is used to apply the loop optimizations without major modifications of the original code. Accordingly, the ExaFSA project was a very good opportunity for us to demonstrate that the Xevolver approach can help an appropriate division of labor between application developers and performance engineers by achieving separation of concerns. This clarification of role-sharing will be very helpful for long-term application development especially in an upcoming extreme-scale computing era.

The Role of Japan in HPC Collaborations

The SPPEXA program was unique in the sense that it established sustainable connections in the field of HPC between France, Germany, and Japan. With the supercomputing infrastructure in Japan (and its upcoming flagship supercomputer Fugaku), the three countries are suitable partners for portability and methodology comparisons and, thus, synergistic research developments (such as within SPPEXA connection).

A new field of interest in Japan, Germany, and France is data science and its connection to HPC. SPPEXA participated in the tri-lateral workshop in Tokyo "Convergence of HPC and Data Science for Future Extreme Scale Intelligent Applications", where we discussed new possible collaborations in the fields of HPC and Big Data. Looking back at SPPEXA, we see many success stories, and hope for a lot of continuing collaborations.

Part II
SPPEXA Project Consortia Reports

ADA-FS—Advanced Data Placement via Ad hoc File Systems at Extreme Scales

Sebastian Oeste, Marc-André Vef, Mehmet Soysal, Wolfgang E. Nagel, André Brinkmann, and Achim Streit

Abstract Today's High-Performance Computing (HPC) environments increasingly have to manage relatively new access patterns (e.g., large numbers of metadata operations) which general-purpose parallel file systems (PFS) were not optimized for. Burst-buffer file systems aim to solve that challenge by spanning an ad hoc file system across node-local flash storage at compute nodes to relief the PFS from such access patterns. However, existing burst-buffer file systems still support many of the traditional file system features, which are often not required in HPC applications, at the cost of file system performance.

The ADA-FS project aims to solve that challenge by providing a temporary burst-buffer file system—GekkoFS—which relaxes POSIX, based on previous usage studies of how HPC applications use file systems. Due to a highly distributed and decentralized design GekkoFS reaches scalable data and metadata performance with tens of millions of metadata operations per second on a 512 node cluster. The ADA-FS project further investigated the benefits of using ad hoc file systems and how they can be integrated into the workflow of supercomputing environments. In addition, we explored how to gather application-specific information to optimize the file system for an individual application.

S. Oeste (✉) · W. E. Nagel
TU Dresden, Dresden, Germany
e-mail: sebastian.oeste@tu-dresden.de; wolfgang.nagel@tu-dresden.de

M.-A. Vef · A. Brinkmann
Johannes Gutenberg University Mainz, Mainz, Germany
e-mail: vef@uni-mainz.de; brinkman@uni-mainz.de

M. Soysal · A. Streit
KIT Karlsruhe, Karlsruhe, Germany
e-mail: mehmet.soysal@kit.edu; achim.streit@kit.edu

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_4

1 Introduction

Application-imposed workloads on *High-Performance computing* (HPC) environments have considerably changed in the past decade. While traditional HPC applications have been compute-bound, large-scale simulations, today's HPC applications are also generating, processing, and analyzing massive amounts of experimental data—known as *data-driven science* applications—affecting several scientific fields. Some of which have already made significant progress in previously unaddressable challenges due to newly discovered techniques [27, 55].

Many data-driven workloads are based on new algorithms and data structures which impose new requirements on HPC file systems [45, 77]. Particularly, large numbers of metadata operations, data synchronization, non-contiguous and random access patterns, and small I/O requests [14, 45], used in data-driven science applications, are challenging for today's general-parallel file systems (PFSs) to handle since past workloads mostly perform sequential I/O operations on large files. Not only are such applications disruptive to the shared storage system but also heavily interfere with other applications which access the same shared storage system [18, 68]. As a result, many workloads which impose these new types of I/O operations suffer from prolonged I/O latencies, reduced file system performance, and occasional long wait times.

Software-based approaches, e.g., application modifications or middleware and high-level libraries [21, 39], and hardware-based approaches, moving from magnetic disks to NAND-based solid-state drives (SSDs) within PFSs, are attempts to mitigate the impact of these new access patterns on the HPC system. However, software-based approaches often suffer from time-consuming adaptations within applications and are sometimes (based on the underlying algorithms) even impossible to adapt to. One of the hardware-based approaches leverages on, nowadays, existing SSDs, installed within a compute node, in order to use them as *node-local* burst buffers. To achieve high metadata performance, they can be deployed in combination with a dynamic *burst buffer file system* [5, 78]. Nonetheless, existing burst buffer file systems have been mostly POSIX compliant which can severely reduce a file system's peak performance [75].

The ADA-FS project, funded by the German Research Foundation (DFG) through the Priority Programme 1648 "Software for Exascale Computing", aims to further explore the possibilities of burst buffer file systems in this context while investigating how they can be used in a modern HPC system. The developed burst buffer file system—*GekkoFS*—acts as ADA-FS' main component. GekkoFS is a temporarily deployed, highly-scalable distributed file system for HPC applications which aims to accelerate I/O operations of common HPC workloads that are challenging for modern PFSs. As such, it can be used in several temporary use cases, such as the lifetime of a compute job or in longer-term use cases, e.g., campaigns. Unlike previous works on burst buffer file systems, it relaxes POSIX by removing some of the semantics that most impair I/O performance in a distributed context and takes previous studies on the behavior of HPC applications into account [37] to optimize the most used file system operations. As a result, GekkoFS reaches scalable data and metadata performance with tens of millions of metadata operations per second on a 512 node cluster while still providing strong consistency for file system operations that target a specific file or directory. In fact, due to its highly distributed and decentralized file system design, GekkoFS is built to perform on even bigger supercomputers, as exascale environments are right around the corner.

While GekkoFS provides the core building block within ADA-FS, it relies and benefits from further information of the application it is used with. Application-specific information that we gather can then further optimize the file system (e.g., the used file system block size) and therefore may increase the file system's performance in terms of latency and throughput. In addition, the ADA-FS project investigated how such a temporary and *on demand* burst buffer file system can be integrated into the workflow of batch systems in supercomputing environments. Although it is hard to reliably predict when compute jobs finish to prematurely deploy GekkoFS for a following ADA-FS job, for instance, we investigated and showed the benefits of on demand burst buffer file systems concerning both application performance and the reduction of the PFS load as a result of using such a file system.

The article is structured as follows: first, we describe GekkoFS' design and its evaluation of nowadays common and challenging HPC workloads on a 512 node cluster in Sect. 2. Section 3 discusses the existing challenges when data is staged in advance and how we solved the challenge, through implementing a plugin for the batch system. Section 4 discusses how we can detect system resources like the amount of node local storage or the NUMA configuration of a node which can be used for the deployment of the GekkoFS file system even on heterogenous compute nodes. In Sect. 5 we show how the option for an on demand file system, can be added to an HPC system. We follow with an evaluation of the performance of GekkoFS for new NVME based storage systems in Sect. 6. Finally, we conclude in Sect. 7.

2 GekkoFS—A Temporary Burst Buffer File System for HPC

In this section, we present the main component of ADA-FS—GekkoFS. GekkoFS is a temporarily deployed, highly-scalable burst buffer file system for HPC applications. In general, the goal of GekkoFS is to accelerate I/O operations in common HPC workloads that are challenging for modern PFSs while offering the combined storage capabilities of node-local storage devices. Further, it does not only aim for providing scalable I/O performance, but, in particular, focuses on offering scalable metadata performance by departing from traditional ways of handling metadata in distributed file systems. To provide a single, global namespace, accessible to all file system nodes, the file system pools together fast node-local storage resources of all participating file system nodes.

Based on previous studies [37] on the behavior of HPC applications, GekkoFS relaxes or removes some of the POSIX semantics, known to heavily impact I/O performance in a distributed environment. As a result, it is able to optimize for the most used file system operations, achieving tens of millions of metadata operations per second on a 512 node cluster. At the same time, GekkoFS is able to run complex applications, such as OpenFOAM solvers [32], and since the file system runs in user-space and it can be easily deployed in under 20 s on a 512 node cluster, it is usable by any user. Consequently, GekkoFS can be used for several use cases which require an ephemeral distributed file system, such as during the lifetime of a compute job or campaigns where data is simultaneously accessed by many nodes in short bursts.

Parts of this section's contents is build on the conference paper by the authors M.-A. Vef et al. [72] and the journal article by the authors M.-A. Vef et al. [71] which both discuss each of the system components of GekkoFS in more detail and provide an in-depth investigation into the performance of GekkoFS compared to other file systems in various HPC environments. First, Sect. 2.1 provides a background on parallel and distributed file systems and discusses some of the related work in the context of burst buffer file systems. Section 2.2 presents the file system's core architecture and design to achieve scalable data and metadata performance in a distributed environment. Finally, in Sect. 2.3 we demonstrate GekkoFS data and metadata performances.

2.1 Related Work

In this section, we give an overview over existing HPC file systems and discuss the differences to GekkoFS.

2.1.1 General-Purpose Parallel File Systems

Most HPC systems are equipped with a backend storage system which is globally accessible using a parallel file system (e.g., GPFS [57], Lustre [7, 53], BeeGFS [26], or PVFS [56]). These file systems offer a POSIX-like interface and focus on data consistency and long-term storage. However, due to the nature of the file system being globally accessible, single applications can disrupt the I/O performance of other applications as well. In addition, these file systems are not well suited for small file accesses, in particular on shared files, often found in scientific applications [45].

The design of GekkoFS does not focus on long-term storage and aims for temporary use cases, such as in the context of compute jobs or campaigns. In addition, since GekkoFS relaxes POSIX semantics, it is able to provide a significant increase in metadata performance.

2.1.2 Node-Local Burst Buffers

Burst buffers are fast, intermediate storage systems that aim to reduce the load on the global file system and on reducing an applications' I/O overhead [38]. Such burst buffers can be categorized into two groups [78]: remote-shared and node-local. Remote-shared burst buffers are generally dedicated I/O nodes to forward application I/O to the underlying PFS, e.g., DDN's IME[1] and Cray's DataWarp.[2]

Node-local burst buffers, on the other hand, are collocated with compute nodes, using existing node-local storage. This node-local storage is then used to create a (distributed) file system which spans over a number of nodes for the lifetime of a compute job, for example. Node-local burst buffers can also be dependent on the PFS (e.g., PLFS [5]) or are sometimes even managed directly by the PFS [49].

BurstFS [78], perhaps the most related work to ours, is a standalone burst buffer file system which does not require a centralized instance as well. However, GekkoFS is not limited to writing data locally like BurstFS. Instead, all data is distributed across all participating file system nodes to balance data workloads for write and read operations without sacrificing scalability. BeeOND [26] can create a job-temporal file system on a number of nodes similar to GekkoFS. BeeOND is, in contrast to our file system, POSIX compliant and our GekkoFS measurements show a much higher metadata throughput than offered by BeeOND [69, 71].

2.1.3 Metadata Scalability

The management of inodes (containing a file's metadata) and related directory blocks (containing data about which files belong to the directory) are the main scalability limitations of file systems in a distributed environment [73]. Typically, general-purpose PFSs distribute data across all available storage targets. While this technique works well for data, it does not achieve the same throughput when handling metadata [11, 54], although the file system community presented various techniques to tackle this challenge [5, 22, 50, 51, 79, 80]. The performance limitation can be attributed to the sequentialization enforced by underlying POSIX semantics which is particularly degrading throughput when an extremely large number of files is created in a single directory from multiple processes. This workload, common to HPC environments [5, 49, 50, 74], can become an even bigger challenge for upcoming data-science applications. GekkoFS handles directories and replaces directory entries by objects, stored within a strongly consistent key-value store which helps to achieve tens of millions of metadata operations for billions of files.

[1]IME: https://www.ddn.com/products/ime-flash-native-data-cache/.

[2]Datawarp: https://www.cray.com/datawarp.

2.2 Design

In this section, we present goals, architecture, and general design of GekkoFS which allows scalable data and metadata performance. In general, any user without administrative access should be able to deploy GekkoFS. The user dictates on how many compute nodes and at which path the mountpoint of GekkoFS and its metadata and data is stored. The user is then presented with a single global namespace, consisting of the aggregated node-local storage of each node. To provide this functionality GekkoFS aims to achieve four core goals:

Scalability: GekkoFS should be able to scale with an arbitrary number of nodes and efficiently use available hardware.

Consistency model: GekkoFS should provide the same strong consistency as POSIX for common file system operations that access a specific data file. However, the consistency of directory operations, for example, can be relaxed.

Fast deployment: To avoid wasting valuable and expensive resources in HPC environments, the file system should startup within a minute and be ready for usage immediately by applications after the startup succeeds.

Hardware independence: GekkoFS should be able to support networking hardware that is commonly used in HPC environments, e.g., Omni-Path or Infiniband. The file system should be able to use the native networking protocols to efficiently move data between file system nodes. Finally, GekkoFS should work with modern and future storage technologies that are accessible to a user at an existing file system path.

2.2.1 POSIX Semantics

Similarly to PVFS [12] and OrangeFS [42], GekkoFS does not provide complex global locking mechanisms. In this sense, applications should be responsible to ensure that no conflicts occur, in particular, concerning overlapping file regions. However, the lack of distributed locking has consequences for operations where the number of affected file system objects is unknown beforehand, e.g., `readdir()` called by the `ls -l` command. In these *indirect file system operations*, GekkoFS does not guarantee to return the current state of the directory and follows the eventual-consistency model. Furthermore, each file system operation is synchronous without any form of caching to reduce file system complexity and to allow for an evaluation of its raw performance capabilities.

Further, GekkoFS does not support move or rename operations or linking functionality as HPC application studies have shown that these features are rarely or not used at all during the execution of a parallel job [37]. Such unsupported file system operations then trigger an I/O error to notify an application. Finally, security management in the form of access permissions is not maintained by GekkoFS since it already implicitly follows the security protocols of the node-local file system.

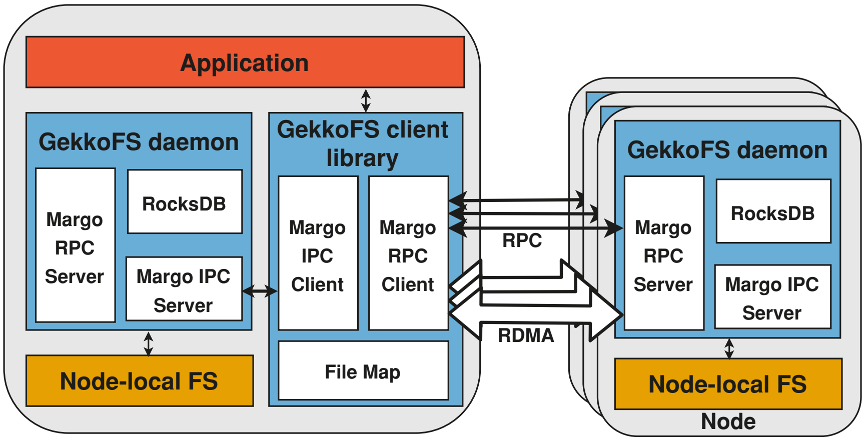


Fig. 1 GekkoFS architecture

2.2.2 Architecture

The architecture of GekkoFS (see Fig. 1) consists of two main components: a client library and a server process. An application that uses GekkoFS must first preload the client interposition library which intercepts all file system operations and forwards them to a server (*GekkoFS daemon*), if necessary. The GekkoFS daemon, which runs on each file system node, receives forwarded file system operations from clients and processes them independently, sending a response when finished. In the following paragraphs, we describe the client and daemon in more detail.

2.2.3 GekkoFS Client

The client consists of three components: (1) An interception interface that catches relevant calls to GekkoFS and forwards unrelated calls to the node-local file system; (2) a file map that manages the file descriptors of open files and directories, independently of the kernel; and (3) an RPC-based communication layer that forwards file system requests to local/remote GekkoFS daemons.

Each file system operation is forwarded via an RPC message to a specific daemon (determined by hashing of the file's path, similar to Lustre DNE 2[3]) where it is directly executed. In other words, GekkoFS uses a pseudo-random distribution to spread data and metadata across all nodes, also known as *wide-striping*. Because each client is able to independently resolve the responsible node for a file system operation, GekkoFS does not require central data structures that keep track of where metadata or data is located. To achieve a balanced data distribution for large files,

[3] https://lustre.ornl.gov/ecosystem-2016/documents/papers/LustreEco2016-Simmons-DNE.pdf.

data requests are split into equally sized chunks before they are distributed across file system nodes (or GekkoFS daemons). The GekkoFS daemons then store each received chunk in a separate file (so-called *chunk files*) in its underlying node-local storage. If supported by the underlying network fabric protocol, the client exposes the relevant chunk memory region to the daemon, accessed via *remote-direct-memory-access* (RDMA).

2.2.4 GekkoFS Daemon

GekkoFS daemons consist of three parts: (1) A key-value store (KV store) used for storing metadata; (2) an I/O persistence layer that reads/writes data from/to the underlying local storage system; and (3) an RPC-based communication layer that accepts local and remote connections to handle file system operations.

Each daemon operates a single local RocksDB KV store [17]. RocksDB is optimized for NAND storage technologies with low latencies and fits GekkoFS' needs as SSDs are primarily used as node-local storage in today's HPC clusters. While RocksDB fits this use case well, the component is replaceable by other software or hardware solutions. Therefore, GekkoFS may introduce various choices for backends in the future to, for example, support recent key-value SSDs[4]

For the communication layer, we leverage on the *Mercury* RPC framework [62]. It allows GekkoFS to be network-independent and to efficiently transfer large data within the file system. Within GekkoFS, Mercury is interfaced indirectly through the *Margo* library which provides *Argobots*-aware wrappers to Mercury's API with the goal to provide a simple multi-threaded execution model [13, 58]. Using Margo allows GekkoFS daemons to minimize resource consumption of Margo's progress threads and handlers which accept and handle RPC requests [13].

Further, as indicated in Sect. 2.1.3, GekkoFS does not use a global locking manager. Therefore, when multiple processes write to the same file region concurrently, they may cause a shared write conflict with resulting undefined behavior with regards to which data is written to the underlying node-local storage. Such conflicts can, however, be handled locally by any GekkoFS daemon because it is using a POSIX-compliant node-local file system to store the corresponding data chunks, serializing access to the same chunk file. Note that such conflicts in a single file only affect one chunk at a time since the file's data is spread across many chunk files in the file system. As a result, chunks of that file are not disrupted during such a potential shared write conflict.

[4]https://www.samsung.com/semiconductor/global.semi.static/Samsung_Key_Value_SSD_enables_High_Performance_Scaling-0.pdf.

2.3 Evaluation

In this section, we evaluate the performance of GekkoFS based on various unmodified microbenchmarks which catch access patterns that are common in HPC applications. First, we describe the experimental setup and introduce the workloads that we simulate with microbenchmark applications. Then, we investigate the startup time of GekkoFS and compare metadata performance against a Lustre parallel file system. Although GekkoFS and Lustre have different goals, we point out the performances that can be gained by using GekkoFS as a burst buffer file system. Finally, we evaluate the data performance of GekkoFS and discuss the measured results.

2.3.1 Experimental Setup

We evaluated the performance of GekkoFS based on various unmodified microbenchmarks which catch access patterns that are common in HPC applications. Our experiments were conducted on the *MOGON II* supercomputer, located at the Johannes Gutenberg University Mainz in Germany. All experiments were performed on Intel 2630v4 Intel Broadwell processors (two sockets each). The node main memory capacity ranges from 64 GiB up to 512 GiB. MOGON II uses 100 Gbit/s Intel Omni-Path to establish a fat-tree network between all compute nodes. In addition, each node provides a data center Intel SATA SSD DC S3700 Series as scratch-space (*XFS* formatted) usable within a compute job. We used these SSDs for storing data and metadata of GekkoFS which uses an internal chunk size of 512 KiB. All Lustre experiments were performed on a Lustre scratch file system with 12 Object Storage Targets (OSTs), 2 Object Storage Servers (OSSs), and 1 Metadata Service (MDS) with a total of 1.2 PiB of storage.

Before each experiment iteration, GekkoFS daemons are restarted (requiring less than 20 s for 512 nodes), all SSD content is removed, and kernel buffer, inode, and dentry caches are flushed. The GekkoFS daemon and the application under test are pinned to separate processor sockets to ensure that file system and application do not interfere with each other.

2.3.2 Metadata Performance

We simulated common metadata intensive HPC workloads using the unmodified *mdtest* microbenchmark [41] to evaluate GekkoFS' metadata performance and compare it against a Lustre parallel file system. Although GekkoFS and Lustre have different goals, we point out the performances that can be gained by using GekkoFS as a burst buffer file system. In our experiments, mdtest performs *create*, *stat*, and *remove* operations in parallel in a single directory—an important workload in many

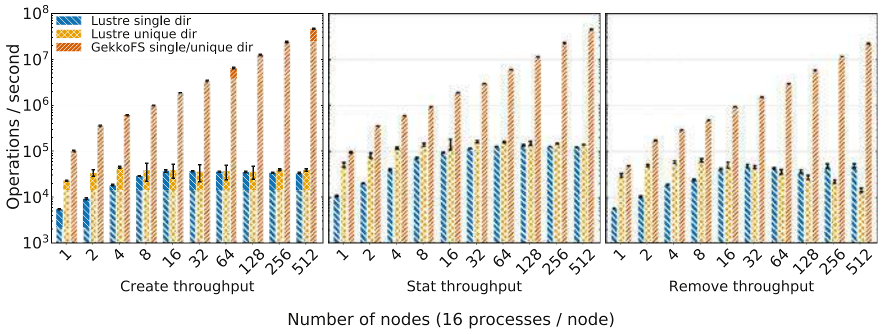


Fig. 2 GekkoFS' file create, stat, and remove throughput for an increasing number of nodes compared to a Lustre file system

HPC applications and among the most difficult workloads for a general-purpose PFS [74].

Each operation on GekkoFS was performed using 100,000 zero-byte files per process (16 processes per node). From the user application's perspective, all created files are stored within a single directory. However, due to GekkoFS' internally kept flat namespace, there is conceptually no difference in which directory files are created. This is in contrast to a traditional PFS that may perform better if the workload is distributed among many directories instead of in a single directory.

Figure 2 compares GekkoFS with Lustre in three scenarios with up to 512 nodes: file creation, file stat, and file removal. The y-axis depicts the corresponding operations per second that were achieved for a particular workload on a logarithmic scale. Each experiment was run at least five times with each data point representing the mean of all iterations. GekkoFS' workload scaled with 100,000 files per process, while Lustre's workload was fixed to four million files for all experiments. We fixed the number of files for Lustre's metadata experiments because Lustre was otherwise detecting hanging nodes when scaling to too many files.

Lustre experiments were run in two configurations: All processes operated in a single directory (`single dir`) or each process worked in its own directory (`unique dir`). Moreover, Lustre's metadata performance was evaluated while the system was accessible by other applications as well.

As seen in Fig. 2, GekkoFS outperforms Lustre by a large margin in all scenarios and shows close to linear scaling, regardless of whether Lustre processes operated in a single or in an isolated directory. Compared to Lustre, GekkoFS achieved around 46 million creates/s ($\sim$1405$\times$), 44 million stats/s ($\sim$359$\times$), and 22 million removes/s ($\sim$453$\times$) on 512 nodes. The standard deviation was less than 3.5% which was computed as the percentage of the mean. Therefore, we achieve our scalability goal, demonstrating the performance benefits of distributing metadata and decoupling directory entries from non-scalable directory blocks (see Sect. 2.2).

Additional GekkoFS experiments were also run while Mogon II was used by other users during production, revealing network interference within the cluster.

With up to 128 nodes we were unable to measure a difference in metadata operation throughput outside of the margin for error compared to the experiments in an undisturbed environment (see Fig. 2). For 256 and 512, we measured a reduced metadata operation throughput between 10 and 20% for create and stat operations. Remove operation throughput remained unaffected.

Lustre's metadata performance did not scale beyond approximately 32 nodes, demonstrating the aforementioned metadata scalability challenges in such a general-purpose PFS. Moreover, processes in Lustre experiments that operated within their own directory achieved a higher performance in most cases, except for the remove case where Lustre's `unique dir` remove throughput is reduced by over 70% at 512 nodes compared to Lustre's `single dir` throughput. This is because the time required to remove the directory of each process (in which it creates its workload) is included in the remove throughput and the number of created `unique` directories increases with the number of used processes in an experiment. Similarly, the time to create the process directories is also included in the create throughput but does not show similar behavior to the case of the remove throughput, indicating optimizations towards create operations.

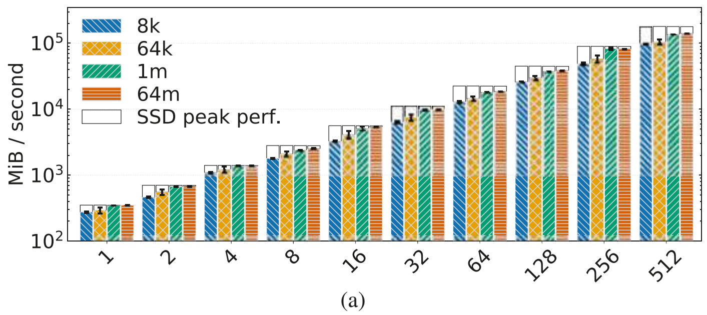
2.3.3 Data Performance

We used the unmodified *IOR* [31] microbenchmark to evaluate GekkoFS' I/O performance for sequential and random access patterns in two scenarios: Each process is accessing its own file (file-per-process) and all processes access a single file (shared file). We used 8 KiB, 64 KiB, 1 MiB, and 64 MiB *transfer sizes* to assess the performances for many small I/O accesses and for few large I/O requests. We ran 16 processes on each client, each process writing and reading 4 GiB in total.

GekkoFS data performance is not compared with the Lustre scratch file system as the peak performance of the used Lustre partition, around 12 GiB/s, is already reached for $\leq$10 nodes for sequential I/O patterns. Moreover, Lustre has shown to scale linearly in larger deployments with more OSSs and OSTs being available [48].

Figure 3 shows GekkoFS' sequential I/O throughput in MiB/s, representing the mean of at least five iterations, for an increasing number of nodes for different transfer sizes. In addition, each data point is compared to the peak performance that all aggregated SSDs could deliver for a given node configuration, visualized as a white rectangle, indicating GekkoFS' SSD usage efficiency. In general, every result demonstrates GekkoFS' close to linear scalability, achieving about 141 GiB/s ($\sim$80% of the aggregated SSD peak bandwidth) and 204 GiB/s ($\sim$70% of the aggregated SSD peak bandwidth) for write and read operations for a transfer size of 64 MiB for 512 nodes.

Figure 4 shows GekkoFS' throughput for random accesses for an increasing number of nodes, showing close to linear scalability in all cases. The file system achieved up to 141 GiB/s write throughput and up to 204 GiB/s read throughput for 64 MiB transfer sizes at 512 nodes.



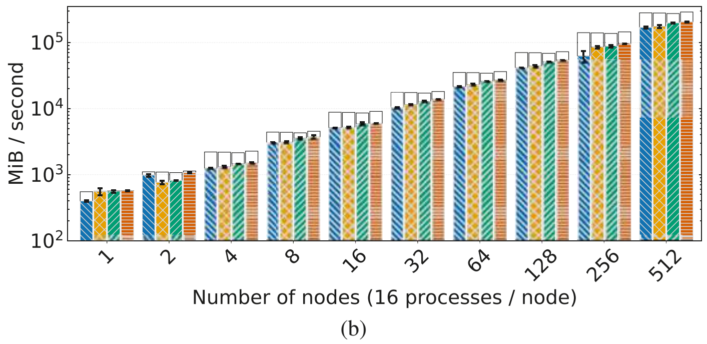
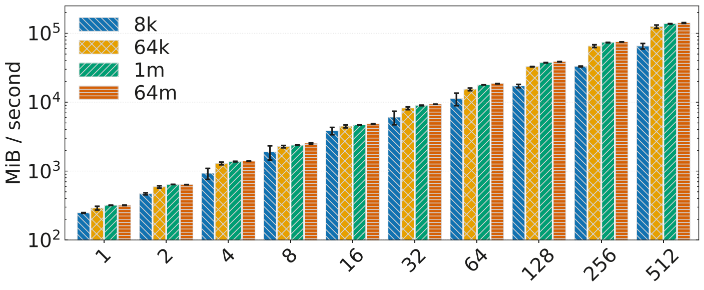


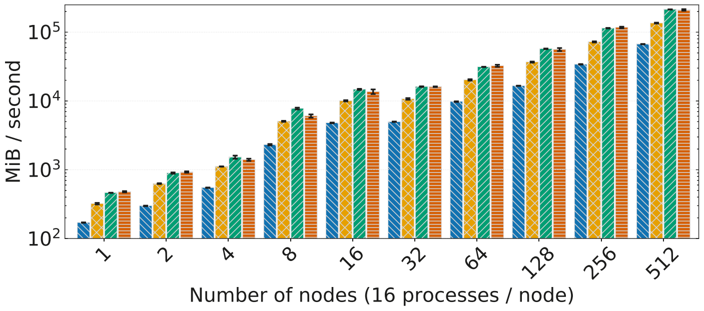
Fig. 3 GekkoFS' sequential throughput for each process operating on its own file compared to the plain SSD peak throughput. (**a**) Write throughput. (**b**) Read throughput

For the file-per-process cases, sequential and random access I/O throughput are similar for transfer sizes larger than the file system's chunk size (512 KiB). This is due to transfer sizes larger than the chunk size internally access whole chunk files while smaller transfer sizes access one chunk at a random offset. Consequently, random accesses for large transfer sizes are conceptually the same as sequential accesses. For smaller transfer sizes, e.g., 8 KiB, random write and read throughput decreased by approximately 33 and 60%, respectively, for 512 nodes owing to the resulting random access to positions within the chunks.

For the shared file cases, a drawback of GekkoFS' synchronous and cacheless design becomes visible. No more than approximately 150 K write operations per second were achieved. This was due to network contention on the daemon which maintains the shared file's metadata whose size needs to be constantly updated. To overcome this limitation, we added a rudimentary client cache to locally buffer size updates of a number of write operations before they are sent to the node that manages the file's metadata. As a result, shared file I/O throughput for sequential and random access were similar to file-per-process performances since chunk management on the daemon is then conceptually indifferent in both cases.



(a)



(b)

Fig. 4 GekkoFS' random throughput for each process operating on its own file. (**a**) Write throughput. (**b**) Read throughput

3 Scheduling and Deployment

In order to transfer the data to a previously generated on demand file system in time, the nodes that will be allocated to a job must be known in advance. Today's schedulers plan the resources of a supercomputer. The schedule is based on user requested wall times. Reality shows that the users requested wall times are very inaccurate, and thus the scheduler's predictions are unreliable.

Here two investigations were made and published. In the first work, we have shown that we can improve wall time estimates based on simple job metadata. We also used unconsidered metadata that is usually not publicly available [65].

Predicting the run times of jobs is only one aspect of the challenge. However, the essential factor is the prediction of node allocation to a job. In this second investigation, we have determined the influence of the wall-time on the node prediction [64]. The question we wanted to answer—How good do wall time predictions have to be to predict the allocated nodes accurately?

3.1 Walltime Prediction

One of the challenges is to know which nodes are going to be allocated to a queued job. The HPC scheduler predicts these nodes based on the user given wall times. Therefore, we have decided to evaluate whether there is an easy way to predict such wall time automatically. Our proposed approach for wall time prediction is to train an individual model for every user. For this, we used methods from the machine learning domain and added job metadata, which was in previous work unconsidered. As historical data, we used workloads from two HPC-systems at the Karlsruhe Institute for Technology/Steinbuch Centre for Computing [66], the ForHLR I + II [34, 35] clusters. To train the model, we used automatic machine learning (AUTOML). AUTOML automates the process of hyperparameter optimization and selecting the correct model. We have chosen the auto ML library auto-sklearn [20], which is based on scikit-learn [9, 52].

In Fig. 5 the comparison of the user given wall times and the wall time prediction is shown. As a metric, the median absolute error (medAE) in hours is depicted as cumulative distribution. A model trained with AUTOML shows for 60% of the users a medAE of approximately 1 h on the ForHLR I and 1.4 h for the ForHLR II. The user estimations show a medAE deviation of about 7.4 h on both clusters. So we are able to reduce the median absolute deviation from 7.4 down to 1.4 h in average. Considering the fact that simple methods were used and no insight was provided into the job payload, this result is very good.

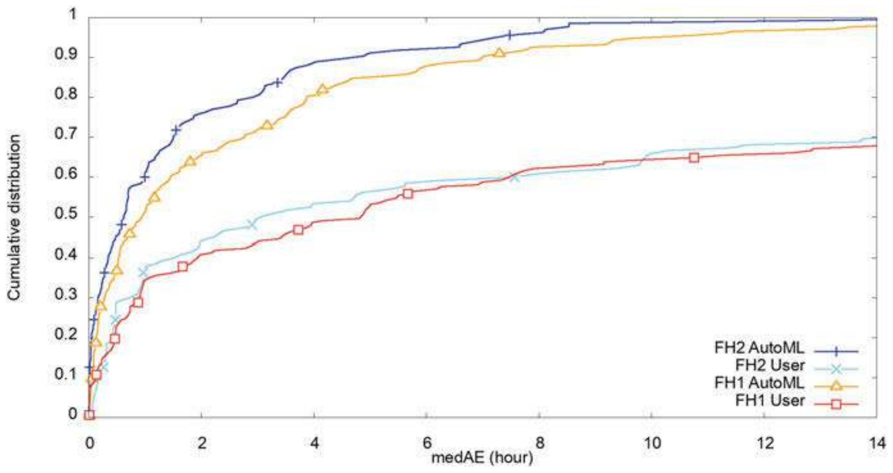


Fig. 5 Comparison of median absolute error (medAE) for ForHLR I+II. X-axis Median absolute error in hours, Y-Axis cumulative distribution

3.2 Node Prediction

As mentioned before predicting the run times of jobs is only one aspect of the challenge. However, the decisive factor is the accuracy of node allocation prediction. In this subsequent investigation, we have determined the impact on the node allocation accuracy with improved wall times. Therefore the ALEA Simulator [2] has been extended to simulate the time of the node allocation list [64].

We have conducted several simulations with subsequently improved job run time estimates, from inaccurate wall times as provided by users to fully accurate job run time estimates. For this purpose, we introduce $\tilde{T}_{\text{Req}}$, the "refined" requested wall time,

$$\tilde{T}_{\text{Req}} = T_{\text{Run}} + \lambda(T_{\text{Req}} - T_{\text{Run}}) \quad \text{with} \quad \lambda \in [0, 1], \tag{1}$$

where T_{Req} is the user requested wall time and T_{Run} is the run time of the job. To effectively simulate different precision of requested wall times, each job in the workload is modified by the same λ.

The result of the simulation is shown in Fig. 6, each bar represents a simulation with a different λ value. The bars are categorized into four groups based on the valid node allocation prediction (T_{NAP}). The blue part represents the jobs that are started immediately (instant) execution after the job is submitted to the batch system. These instantly started jobs offer of course no time to create a file system or even stage data. The orange part represents queued jobs with a T_{NAP} between 0 and 1 s. The green part shows jobs with a T_{NAP} from one second up to 10 min and red indicates long term prediction with a valid node allocation prediction over 10 min. The class of jobs with long-term predictions (red) is in our focus. This long-term predictions

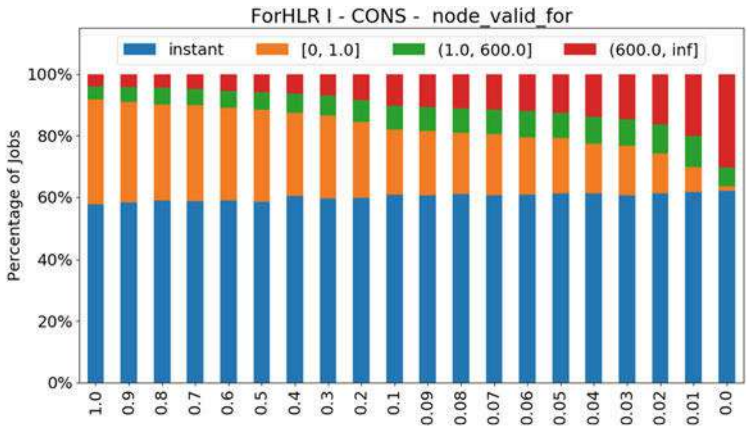


Fig. 6 Job distributions of ForHLR II workload with back-filling (CONS). Blue color denotes instant jobs, orange color means job having prediction $\leq$1 s, green color denotes jobs with 1 and 600 s and red color denotes long-term predictions($<$600 s)

increase significantly only at very small $\lambda \leq 0.1$ which proves that very good run time estimates are needed.

3.3 On Demand Burst Buffer Plugin

From both evaluations, it is clear, that advanced data staging based on the scheduler prediction is not possible. Also, by using state-of-the-art methods such as machine learning, the accuracy is not sufficient. Therefore we decided to extend the functionality of the SLURM [15] scheduler. Slurm has a feature to manage burst buffers [16]. However, the current implementation status only includes support for the Cray DataWarp solution. Management of burst buffers using other storage technologies is documented, but not yet implemented. With the developed plugin, we extend the functionality of SLURM to create a file system on demand. For the prototype implementation, we also developed tools which deploy BeeOND (BeeGFS On Demand) as an on demand file system per job. Other parallel file systems,e.g. Lustre [7] or GekkoFS, can be added easily. The user requests an on demand file system by a job flag. He can also specify if data should be staged in and out. The SLURM controller marks the jobs and then does the corresponding operations [76].

3.4 Related Work

The requested wall times are unfortunately far away from the real used wall time. Gibbons [23, 24], and Downey [19] used historical workloads to predict the wall times of parallel applications. They predict wall times based on templates. These templates are created by analyzing previously collected metadata and grouped according to similarities. However, both approaches are restricted to simple definitions.

In the recent years, machine learning algorithms have been used to predict resource consumption in several studies [33, 40, 43, 44, 61, 70].

Predicting the run-time of jobs is also important in different topics, like for energy aware scheduling [3]. Here the applications' power and performance characteristics are considered to provide an optimized trade off between energy savings and job execution time.

However, all of the above mentioned studies do not try to evaluate the accuracy of the node allocation predictions. Most of the publications focus on observing the utilization of the HPC system and the reliability of the scheduler estimated job start times. In our work, we focus on the node allocation prediction and how good wall time estimates have to be. This directly affects, whether a cross-node, on demand, independent parallel FS can be deployed, and data can be pre-staged, or not.

4 Resource and Topology Detection

Compute nodes of modern HPC systems tend to get more heterogeneous. To plan a proper deployment of the GekkoFS file system on the compute nodes, knowledge of the underlying storage components are vital. This section describes what kind of resource information is of interest and shows possible ways to gather this information. Further, we discuss the architecture of the sysmap tool that we build to collect relevant information.

When thinking about the resources of a compute node, we distinguish between static and dynamic resource usage. Static resource information describes components that do not change frequently and are often similar between nodes. This includes the number of CPU cores, the amount of main memory, the number and capacity of node-local storage devices, or the type of file system. It is unlikely that this kind of hardware is replaced frequently. Otherwise different parts of a Cluster may have different configurations, e.g., one island of a Cluster may have more RAM than another island. On the other hand, dynamic resource usage describes the available resources at a certain point in time.

The goal is to hold a map of the resources available on a system. On the one hand, this can be used as an input for the data staging. On the other hand, such information is useful for the deployment of the file system. When the job scheduler has decided on which set of nodes a job will run, available hardware resources can be queried and an appropriate configuration to deploy the file system can be selected.

The sysmap tool can utilize existing hardware discovery libraries such as *hwloc* [8, 25] by using their interface. While *hwloc* does an excellent job for computing-related artifacts like the number of CPUs or cache sizes, it does not focus on the storage subsystem. Therefore, we use information from the `/proc` and `/sys` pseudo file systems to get information about the system. By reading the system configuration, the sysmap tool gathers information about partitions, mountpoints, file systems but also about available kernel modules and I/O-scheduler configuration. Moreover, we gather network information for InfiniBand networks by utilizing the well-known `ibnetdiscover` tool from the OFED distribution [47].

4.1 Design and Implementation

We have designed an extensible architecture for our sysmap tool. Each resource of interest is captured by a so-called *extractor*. Figure 7 shows a schematic UML-diagram of two extractors. Each extractor module consists of an abstract part, which defines the structure of the data that will be gathered and a specialized part which implements the logic to read the data from a specific source, by overriding the abstract interface. In Fig. 7, the `Filesystem_Extractor` and the `Disk_Extractor` are examples of the abstract parts. The `Linux::Filesystem_Extractor` and the

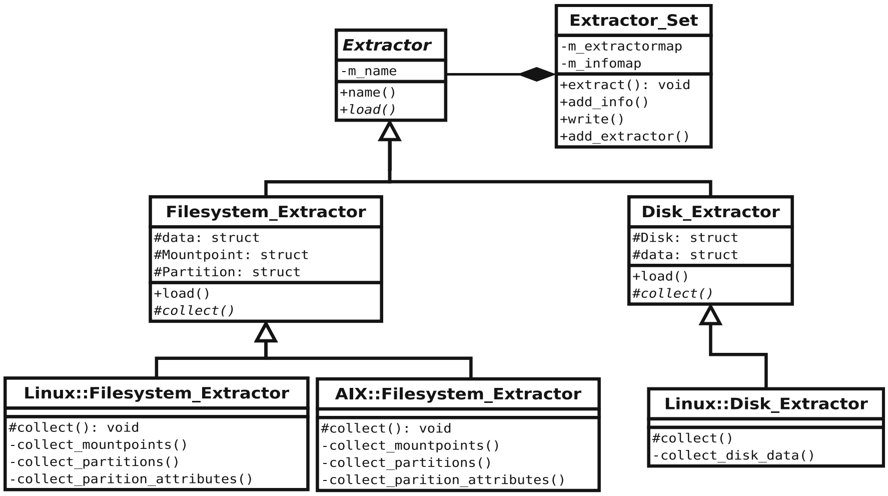


Fig. 7 Simple UML-Diagram for two example *extractor* modules of our system-map tool

`AIX::Filesystem_Extractor` are the specialized parts for extracting information of mountpoints and partitions of a specific system [46]. This is useful because the same information may be available on different systems through different sources. On the one hand, the user has to define the abstract extractor he wants, and the sysmap tool selects the source depending on what is available on the target system. On the other hand, we can implement specialized extractor modules for different sources resulting in an equivalent representation of the data for our tool. After gathering the data, the sysmap tool provides a wide variety of output formats presenting the data to the user. Since the tool is mentioned to be executed on multiple compute nodes, the recommended way is to store the results in a central database. Figure 8 depicts an overview of the general workflow of the resource discovery process. The sysmap tool runs on the compute nodes and gathers the resource information. Afterwards, the collected data is stored in a central resource database. For our working prototype, we use a *sqlite*[5] database. The information can be queried by the sysquery tool, which queries the resource database and outputs the selected data in JSON format. This way the querying component gets a machine-readable section of the required data which can be easy post-processed for they need. Further, the particular database query remains hidden from the user inside the sysquery tool. The datamodel of the resource database is shown in Fig. 9 and consists of four simple tables. The *HostTable* and *ExtractorTable* are to map the hostname or the extractor name to a numerical ID. Information extracted by an extractor is stored as JSON string in the *DataTable*. Further, a *DataID* is maintained to reference the data from an extractor. In the *Host2Data* table, the *DataID* is

[5]https://www.sqlite.org/index.html.

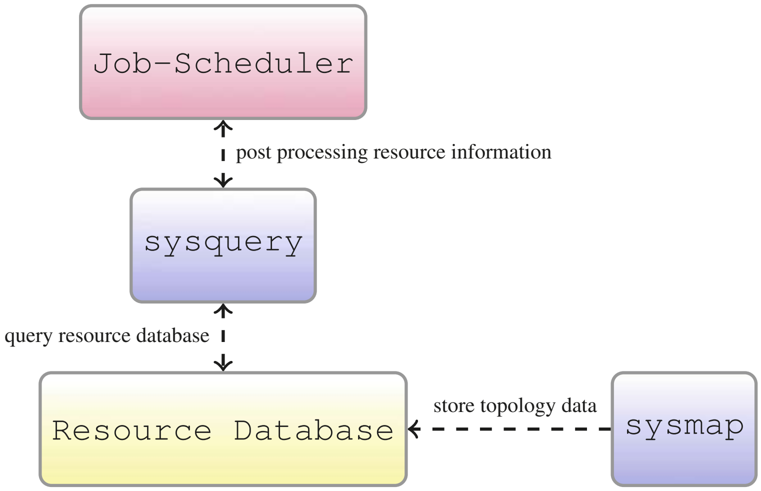


Fig. 8 Overview of resource discovery components, the blue components are part of the sysmap tool suite, the resource database is highlighted as the yellow box, the red box represents the querying component, in this case the Job-Scheduler

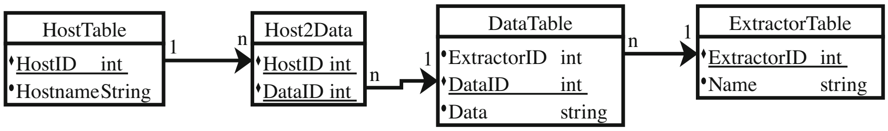


Fig. 9 The datamodel of the resource database

mapped with the corresponding *HostID*. This way, data that is equal across multiple nodes do not need to be stored multiple times but are easy to query. Since, the output of a query is a JSON string, it makes further processing and output easy for the calling script.

5 On Demand File System in HPC Environment

When using on demand file systems in HPC environments, the premise is that the normal operation should not be affected by the use of on demand file systems. The interference on other jobs should be avoided or even reduced. There should be also no modifications, that have a negative impact on the performance or utilization, of the system.

5.1 Deploying on Demand File System

Usually HPC systems use a *batch system*, such as SLURM [60], MOAB [1], or LSF [30]. The batch system manages the resources of the cluster and starts the user jobs on allocated nodes. Before a job is started, a prologue script may be started on one or all allocated nodes and, if necessary, an epilogue script at the end of a job. These scripts are used to clean, prepare, or test the full functionality of the nodes. We modified these scripts to start the on demand file system upon request. During job submission, a user can request an on demand file system for the job. This solution has minimal impact on the HPC system operation. Users without the need for an on demand file system are not affected. An alternative way of deploying a on demand file system we have described in Sect. 3.3

5.2 Benchmarks

As initial benchmarks we tested the startup and shutdown time of the on demand file system (cf. Table 1). Comparing the startup time of BeeGFS on demand to the startup time of GekkoFS (512 Nodes under 20 s) it is clear, that BeeGFS takes too much time for startup and shutdown at larger scales. BeeGFS has a serial section in its startup where a status file is created on every node sequentially. This was also discussed on the mailing list [63] with a possible solution to improve the behavior in future releases.

In Fig. 10 we show the IoZone [10] benchmark to measure the read and write throughput of the on demand file system (solid line). The figure shows that performance increases linearly with the number of used compute nodes. The limiting factor here is the aggregate throughout of the used SATA-SSDs. A small throughput variation can be observed due to normal performance scattering of SSDs [36]. The dotted line indicates the theoretical throughput with NVMe devices. Here we assumed the performance for today's common PCIe $\times 4$ NVM devices [29] with a throughput of 3500/2000 MB/s of read/write performance.

In a further test, we evaluated the storage pooling feature of BeeGFS [4]. We created a storage pool for each switch according to the network topology.In other words, when writing to a storage pool, the data is distributed via the stripe count and chunk size, but remains within the storage pool and thus on a switch. Figure 11 shows the write throughput for three scenarios. Each scenario uses a different number of core switches with six being the full network capacity. In the first

Table 1 BeeGFS startup and shutwdown

Nodes	8	16	32	64	128	256
Startup (s)	10.2	16.7	29.3	56.5	152.1	222.4
Shutdown (s)	11.9	12.1	9.4	15.9	36.1	81.0

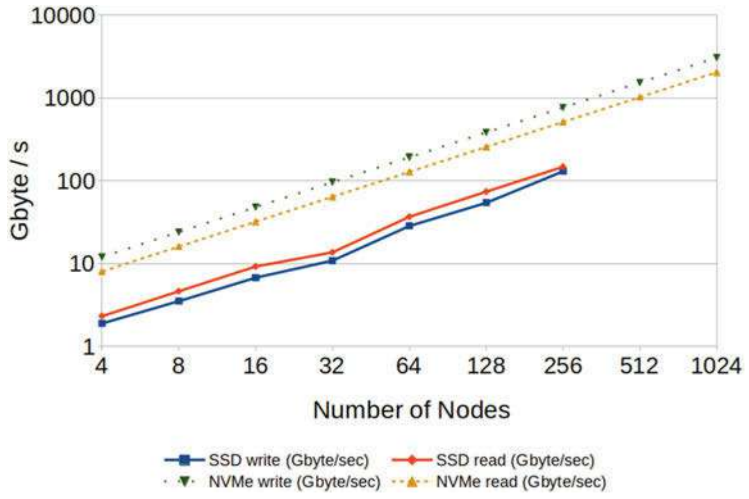


Fig. 10 Solid line: Read/write throughput. Dashed line: extrapolation with the theoretical peak of NVMe-SSDs

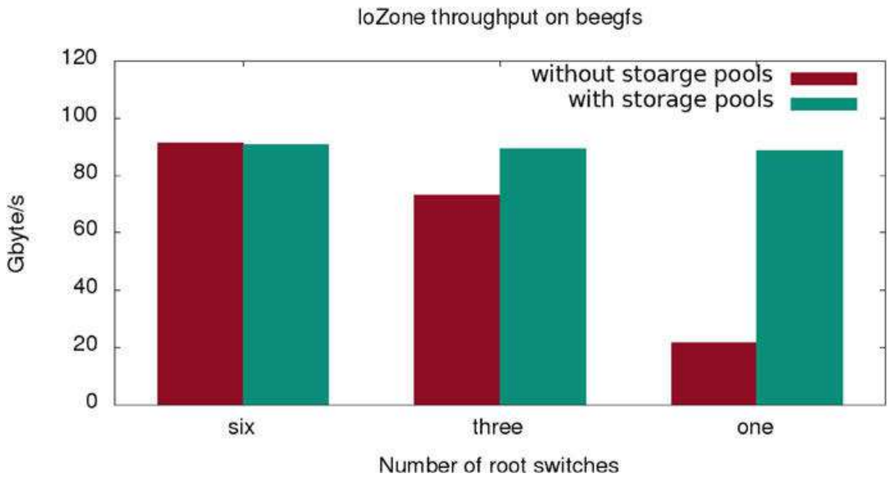


Fig. 11 IoZone write throughput with reduced number of core switches on 240 nodes

experiment, with all six core switches, there is only a minimal performance loss, which indicates a small overhead when using storage pools. In the second case we turned off three switches, and in the last case we turned off five switches. With reduced number of core switches the write throughput drops due to the reduced network capacity. If storage pools are created according to the topology, it is possible to achieve the same performance as with all six switches.

5.3 Concurrent Data Staging

We also considered the case of copying data back to the PFS while an application is running. For this purpose, we evaluated NAStJA [6] with concurrent data staging. To stage the data, during the NAStJA execution, we used the parallel copy tool dcp [59]. The configuration for this use-case:

- We used 24 nodes with 20 cores per node.
- NAStJA was executed on 23 nodes with 20 tasks per node.
- BeeOND was started on all 24 nodes using the idle node as metadata server.
- Three different scenarios were evaluated during the application execution:
  - without data staging,
  - data staging using every node with one task per node for data staging,
  - data staging using the node, where only the meta-data server is running, with 4 tasks executed on this node.

Figure 12 shows the average execution time per time-step of five runs in our different scenarios. In the beginning, the slowdown is significant (orange line) due to the high amount of metadata operations. In this case, a portion of the data is indexed on every node. This indexing is causing interference with the application. When using only the MDS-server to copy the data (green line), the indexing is done only on the MDS-server.

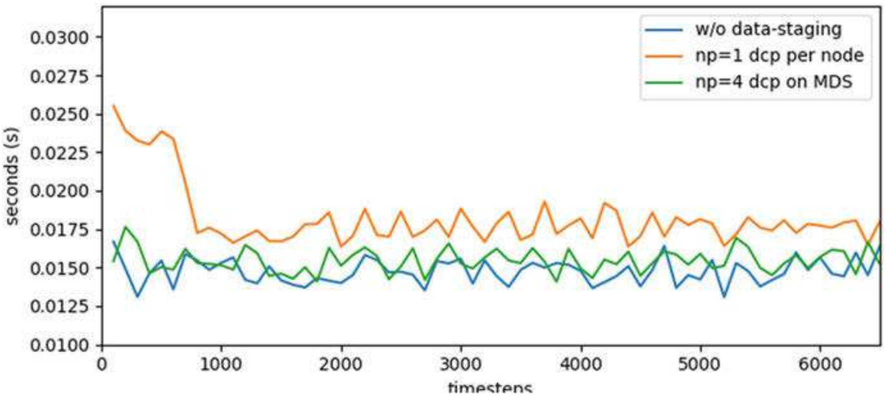


Fig. 12 Average execution time per time-step (5 runs). Without data staging(blue). Concurrent data staging using the meta-data node (green) and using every node (orange)

6 GekkoFS on NVME Based Storage Systems

Recently, new storage technologies such as NVME SSDs have been introduced to modern HPC systems. To evaluate the GekkoFS file system, for future systems, we performed some benchmarks using NVME SSDs. For demonstration we installed GekkoFS on the cluster Taurus [67] of TU Dresden. Taurus consists of ca. 47,000 cores of different architectures. For the demonstration, we use 8 NVME nodes of the HPC-DA [28] extension of Taurus. This extension consists of 90 nodes, and a single node has 8 Intel SSD DC P4610 Series NVME storage with 3.2 TB capacity and a peak bandwidth of 3.2 GB/s. Each node has 2 sockets Intel Xeon E5-2620 v4 with 32 cores and 64 GB main memory. Further, the NVME nodes are equipped with two 100 Gbit/s EDR Infiniband links with a peak bandwidth of 25 GB/s each. This experiment aims to investigate how well GekkoFS performs on new storage architectures.

We installed GekkoFS on Taurus using the Infiniband network provider. For our demonstration, we use 8 NVME nodes in this setup. The nodes are client and server in one. We assign one NVME card per node as backing storage to the GekkoFS daemon. This results in a distributed file system with a total capacity of 25.6 TB and a theoretical maximum bandwidth of 8×3.2 GB/s $=$ 25.6 GB/s for this configuration. To measure the data throughput of GekkoFS and investigate the impact of different access patterns to the file system we utilize the IOR benchmark.

We perform strong scaling tests with 8, 16, 32 and 64 processes writing and reading 1 TB of data. Therefore, we adjust the block size and transfer size for a different number of processes. To avoid interference, we pin the IOR processes to one socket while the GekkoFS daemon is pinned to the other. Before the creation of the GekkoFS file system the NVME devices were cleared, and a new Ext4 file system was created as an underlying file system on the block device. We measure different access patterns, file per process with sequential and random accesses and shared file with sequential access. To avoid measuring cache effects, we flush the page, inode and dentry caches of the operating system before each run.

Figure 13 shows the sequential access pattern. In the figure, one can see that the write bandwidth is stable at around 22 GB/s for all runs. The variation is small, and the values are near to the peak bandwidth of 25 GB/s for this setup. The suitable write bandwidths came from the relatively large transfer sizes of 64 MB to benefit from RDMA. For the read bandwidth, we get values between 13 and 17 GB/s. Also the read bandwidth first decreases when more processes are used and then increases again at the 64 processes. Such a poor read bandwidth is a behavior which could not be observed for the other measurements on MOGON II, where read and write bandwidth are almost equal, and is certainly a point of further investigation.

Figure 14 depicts the random access case. The results are similar to the sequential access pattern, which was expected because the internal handling of GekkoFS makes no difference for these cases. The write bandwidth is stable between 22 and 23 GB/s

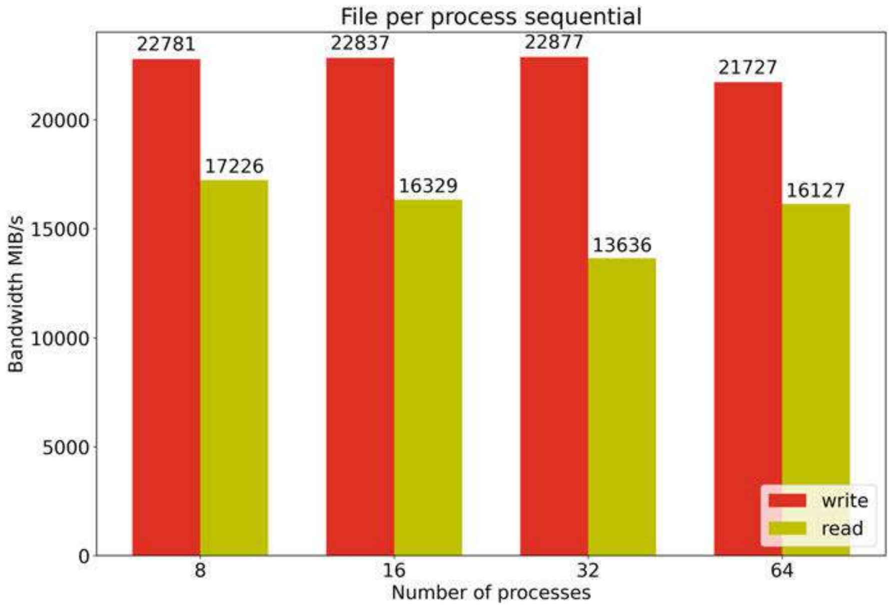


Fig. 13 IOR on GekkoFS on 8 NVME nodes performing a sequential file per process access pattern

and saturates the NVME SSD quite well. For the read, the achieved bandwidth is around 14 GB/s, the values are more stable than for the sequential case, which might be some cache effects.

In Fig. 15 we can see, that even for shared access pattern the results are similar to the file per process access pattern. The write bandwidth is again stable at 22 GB/s and the read bandwidth is around 16 GB/s except for the configuration with 32 processes where the read bandwidth is lower. This is also similar to the sequential file per process configuration in Fig. 13. As a result, we can see that GekkoFS can utilize NVME SSDs and is, therefore, ready for the next generation of storage systems. We could figure out that the different access patterns make no difference for the write bandwidth. For the read bandwidth, there is some bottleneck which needs further investigation. At the time of writing, multiple causes are imaginable; for example, the network layer for Infiniband might be an issue. This could also explain why this problem does not occur for the tests in Mainz because they have other network types.

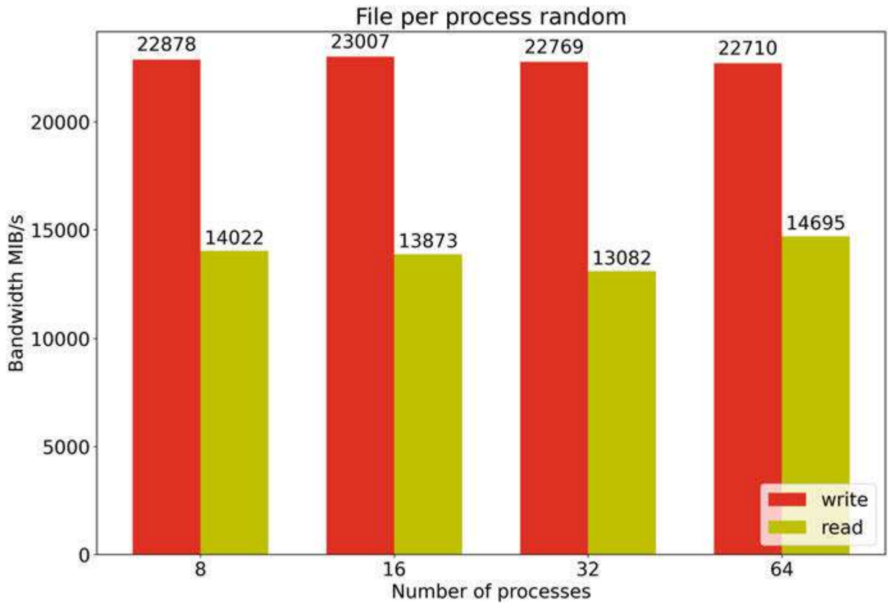


Fig. 14 IOR on GekkoFS on 8 NVME nodes performing a random file per process access pattern

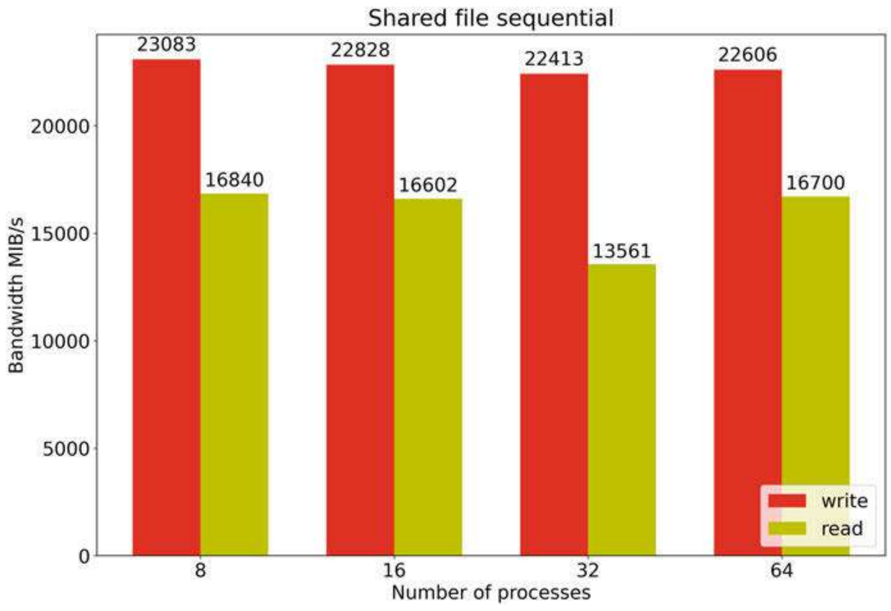


Fig. 15 IOR on GekkoFS on 8 NVME nodes performing a sequential shared file access pattern

7 Conclusion

The goal of the ADA-FS project was to improve I/O performance for parallel applications. Therefore, a distributed burst buffer file system, and several components for deployment and data management were developed. The GekkoFS distributed burst buffer file system as the central part of the project was presented as a scalable and very flexible alternative to handle the challenging I/O patterns of scientific applications. Primarily through the innovative metadata management, it beats conservative shared parallel file systems for metadata intensive workloads by a margin. Thanks to its flexibility, GekkoFS offers the user an exclusive file system for his applications and eliminates several bottlenecks caused by the contention of a shared resource. In addition, GekkoFS has become a basis in the EU-funded *Next Generation I/O for Exascale* (NEXTGenIO) project where it will be continuously and collaboratively developed to support future storage technologies as well, such as persistent memory.

For successful data staging, investigations about the precision of the user-provided wallclock time of jobs were made. We could show how to improve wallclock estimates by considering the metadata of a job, and show a way to integrate the process of deployment and data staging into the job scheduler. Further, we present a tool suite to collect information about hardware resources of a compute node to support the deployment in a flexible manner.

Another topic that was not covered here is the analysis of the required POSIX semantics of parallel applications. These insights show during the design of the file system which operations are required to run scientific workloads. Further, its results can help the user to decide for a storage system that fits his needs best.

The evaluations showed that GekkoFS provides close to linear data and metadata scalability up to 512 nodes with tens of millions of metadata operations. Due to the decentralized and distributed design, the file system is set to be used in even larger environments as exascale environments are in close reach. Even on the latest storage infrastructure, GekkoFS can operate out of the box at the peak bandwidth at least for write operations.

Following this project, we plan further improvements on GekkoFS, for example, caching offers possibilities to gain even more performance. Another topic that we want to keep working on is the integration of GekkoFS into the job schedulers of the systems and the workflows of the user.

Conclusively, the project reached its goals by improving I/O performance of parallel applications, especially in the field of metadata intensive workloads where traditional parallel file systems are lacking performance.

Acknowledgments This work as part of the project ADA-FS is funded by the DFG Priority Program "Software for Exascale Computing" (SPPEXA, SPP 1648), which is gratefully acknowledged.

This research was conducted using the supercomputer ForHLR II and services offered by Karlsruher Insitute of Technology and the Steinbuch Centre for Computing. The authors gratefully acknowledge the time and granted access to ForHLR II.

Parts of this research were conducted using the supercomputer Mogon II and services offered by Johannes Gutenberg University Mainz. The authors gratefully acknowledge the computing time granted on Mogon II.

The authors gratefully acknowledge the GWK support for funding this project by providing computing time through the Center for Information Services and HPC (ZIH) at TU Dresden on the HPC-DA.

References

1. Adaptive Computing. http://www.adaptivecomputing.com
2. Alea 4: Job scheduling simulator (2019). https://github.com/aleasimulator
3. Auweter, A., Bode, A., Brehm, M., Brochard, L., Hammer, N., Huber, H., Panda, R., Thomas, F., Wilde, T.: A case study of energy aware scheduling on supermuc. In: Proceedings of the 29th International Conference on Supercomputing, ISC 2014, vol. 8488, pp. 394–409. Springer, New York (2014). https://doi.org/10.1007/978-3-319-07518-1_25
4. BeeGFS: BeeGFS Storage Pool. https://www.beegfs.io/wiki/StoragePools (2018). Accessed: August 1 201
5. Bent, J., Gibson, G.A., Grider, G., McClelland, B., Nowoczynski, P., Nunez, J., Polte, M., Wingate, M.: PLFS: a checkpoint filesystem for parallel applications. In: Proceedings of the ACM/IEEE Conference on High Performance Computing (SC), November 14–20, Portland, (2009)
6. Berghoff, M., Kondov, I., Hötzer, J.: Massively parallel stencil code solver with autonomous adaptive block distribution. IEEE Trans. Parallel Distrib. Syst. **29**(10), 2282–2296 (2018). https://doi.org/10.1109/TPDS.2018.2819672
7. Braam, P.J., Schwan, P.: Lustre: The intergalactic file system. In: Ottawa Linux Symposium, p. 50 (2002)
8. Broquedis, F., Clet-Ortega, J., Moreaud, S., Furmento, N., Goglin, B., Mercier, G., Thibault, S., Namyst, R.: hwloc: A generic framework for managing hardware affinities in HPC applications. In: Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing, PDP 2010 (2010)
9. Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., Varoquaux, G.: API design for machine learning software: experiences from the scikit-learn project. In: ECML PKDD Workshop: Languages for Data Mining and Machine Learning, pp. 108–122 (2013)
10. Capps, D., Norcott, W.: Iozone filesystem benchmark (2008). http://iozone.org/
11. Carns, P., Yao, Y., Harms, K., Latham, R., Ross, R., Antypas, K.: Production I/O characterization on the Cray XE6. In: Proceedings of the Cray User Group meeting, vol. 2013 (2013)
12. Carns, P.H., III, W.B.L., Ross, R.B., Thakur, R.: PVFS: A parallel file system for Linux clusters. In: 4th Annual Linux Showcase and Conference 2000. Atlanta (2000)
13. Carns, P.H., Jenkins, J., Cranor, C.D., Atchley, S., Seo, S., Snyder, S., Ross, R.B.: Enabling NVM for data-intensive scientific services. In: 4th Workshop on Interactions of NVM/Flash with Operating Systems and Workloads, INFLOW@OSDI 2016. Savannah (2016)
14. Crandall, P., Aydt, R.A., Chien, A.A., Reed, D.A.: Input/output characteristics of scalable parallel applications. In: Proceedings Supercomputing ’95, San Diego, p. 59 (1995)
15. Documentation, S.: Slurm workload manager—overview. https://slurm.schedmd.com/overview.html
16. Documentation, S.: Slurm workload manager—-slurm burst buffer guide. https://slurm.schedmd.com/burst_buffer.html

17. Dong, S., Callaghan, M., Galanis, L., Borthakur, D., Savor, T., Strum, M.: Optimizing space amplification in rocksdb. In: Proceedings of the 8th Biennial Conference on Innovative Data Systems Research, CIDR 2017. Chaminade (2017)
18. Dorier, M., Antoniu, G., Ross, R.B., Kimpe, D., Ibrahim, S.: Calciom: Mitigating I/O interference in HPC systems through cross-application coordination. In: Processing Symposium on 2014 IEEE 28th International Parallel and Distributed, pp. 155–164. Phoenix (2014)
19. Downey, A.B.: Predicting queue times on space-sharing parallel computers. In: Proceedings of the 11th International Parallel Processing Symposium, pp. 209–218. IEEE, Piscataway (1997)
20. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 28, pp. 2962–2970. Curran Associates, Red Hook (2015). http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf
21. Folk, M., Cheng, A., Yates, K.: HDF5: a file format and I/O library for high performance computing applications. In: Proceedings of the Supercomputing, vol. 99, pp. 5–33 (1999)
22. Frings, W., Wolf, F., Petkov, V.: Scalable massively parallel I/O to task-local files. In: Proceedings of the ACM/IEEE Conference on High Performance Computing (SC), Portland (2009)
23. Gibbons, R.: A historical application profiler for use by parallel schedulers. In: Job Scheduling Strategies for Parallel Processing, pp. 58–77. Springer, Berlin (1997)
24. Gibbons, R.: A historical profiler for use by parallel schedulers. Master's thesis, University of Toronto (1997)
25. Goglin, B.: Managing the topology of heterogeneous cluster nodes with hardware locality (hwloc). In: 2014 International Conference on High Performance Computing and Simulation (HPCS), pp. 74–81. IEEE, Piscataway (2014)
26. Herold, F., Breuner, S., Heichler, J.: An introduction to beegfs (2014). https://www.beegfs.io/docs/whitepapers/Introduction_to_BeeGFS_by_ThinkParQ.pdf
27. Hey, T., Tansley, S., Tolle, K.M. (eds.): The Fourth Paradigm: Data-Intensive Scientific Discovery. Microsoft Research (2009)
28. HPC for Data Analytics (2019). https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/HPCDA. Accessed 25 July 2019
29. Hung, J.J., Bu, K., Sun, Z.L., Diao, J.T., Liu, J.B.: PCI express-based NVMe solid state disk. In: Applied Mechanics and Materials, vol. 464, pp. 365–368. Trans Tech Publications (2014)
30. IBM—platform computing. http://www.ibm.com/systems/platformcomputing/products/lsf/
31. Ior data benchmark (2018). https://github.com/hpc/ior
32. Jasak, H., Jemcov, A., Tukovic, Z., et al.: Openfoam: a C++ library for complex physics simulations. In: International Workshop on Coupled Methods in Numerical Dynamics, vol. 1000, pp. 1–20 (2007)
33. Kapadia, N.H., Fortes, J.A.: On the design of a demand-based network-computing system: The purdue university network-computing hubs. In: Proceedings of the Seventh International Symposium on High Performance Distributed Computing, pp. 71–80. IEEE, Piscataway (1998)
34. Forschungshochleistungsrechner ForHLR 1 (2018). www.scc.kit.edu/dienste/forhlr1.php
35. Forschungshochleistungsrechner ForHLR 2 (2018). www.scc.kit.edu/dienste/forhlr2.php
36. Kim, E.: SSD performance-a primer. In: Solid State Storage Initiative (2013)
37. Lensing, P.H., Cortes, T., Hughes, J., Brinkmann, A.: File system scalability with highly decentralized metadata on independent storage devices. In: IEEE/ACM 16th International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, pp. 366–375 (2016)
38. Liu, N., Cope, J., Carns, P.H., Carothers, C.D., Ross, R.B., Grider, G., Crume, A., Maltzahn, C.: On the role of burst buffers in leadership-class storage systems. In: IEEE 28th Symposium on Mass Storage Systems and Technologies, MSST 2012, Pacific Grove, pp. 1–11 (2012)
39. Lofstead, J.F., Klasky, S., Schwan, K., Podhorszki, N., Jin, C.: Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In: Proceedings of the Sixth International Workshop on Challenges of Large Applications in Distributed Environments, CLADE@HPDC 2008, Boston, pp. 15–24 (2008)

40. Matsunaga, A., Fortes, J.A.: On the use of machine learning to predict the time and resources consumed by applications. In: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp. 495–504. IEEE Computer Society, Washington (2010)
41. Mdtest metadata benchmark (2018). https://github.com/hpc/ior
42. Moore, M., Bonnie, D., Ligon, B., Marshall, M., Ligon, W., Mills, N., Quarles, E., Sampson, S., Yang, S., Wilson, B.: Orangefs: Advancing PVFs. FAST poster session (2011)
43. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel Distrib. Syst. **12**(6), 529–543 (2001)
44. Nadeem, F., Fahringer, T.: Using templates to predict execution time of scientific workflow applications in the grid. In: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 316–323. IEEE Computer Society, Washington (2009)
45. Nieuwejaar, N., Kotz, D., Purakayastha, A., Ellis, C.S., Best, M.L.: File-access characteristics of parallel scientific workloads. IEEE Trans. Parallel Distrib. Syst. **7**(10), 1075–1089 (1996)
46. Oeste, S., Kluge, M., Soysal, M., Streit, A., Vef, M., Brinkmann, A.: Exploring opportunities for job-temporal file systems with ADA-FS. In: Proceedings of the First Joint International Workshop on Parallel Data Storage and Data Intensive Scalable Computing Systems (2016)
47. OpenFabrics Alliance: Open fabric enterprise distribution (2016). https://www.openfabrics.org/. Accessed 16 July 2016
48. Oral, S., Dillow, D.A., Fuller, D., Hill, J., Leverman, D., Vazhkudai, S.S., Wang, F., Kim, Y., Rogers, J., Simmons, J., et al.: OLCFs 1 TB/s, next-generation lustre file system. In: Proceedings of the Cray User Group Conference (CUG 2013), pp. 1–12 (2013)
49. Oral, S., Shah, G.: Spectrum scale enhancements for coral. Paper presentation slides at supercomputing'16. (2016). http://files.gpfsug.org/presentations/2016/SC16/11_Sarp_Oral_Gautam_Shah_Spectrum_Scale_Enhancements_for_CORAL_v2.pdf
50. Patil, S., Gibson, G.A.: Scale and concurrency of GIGA+: file system directories with millions of files. In: Proceedings of the Ninth USENIX Conference on File and Storage Technologies, San Jose, pp. 177–190 (2011)
51. Patil, S., Ren, K., Gibson, G.: A case for scaling HPC metadata performance through de-specialization. In: 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, Salt Lake City, pp. 30–35 (2012)
52. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
53. Qian, Y., Li, X., Ihara, S., Zeng, L., Kaiser, J., Süß, T., Brinkmann, A.: A configurable rule based classful token bucket filter network request scheduler for the lustre file system. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Denver, pp. 6:1–6:12 (2017)
54. Ren, K., Zheng, Q., Patil, S., Gibson, G.A.: IndexFS: scaling file system metadata performance with stateless caching and bulk insertion. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014, New Orleans, pp. 237–248 (2014)
55. Ross, R., Thakur, R., Choudhary, A.: Achievements and challenges for I/O in computational science. In: Journal of Physics: Conference Series, vol. 16, p. 501 (2005)
56. Ross, R.B., Latham, R.: PVFS–PVFS: a parallel file system. In: Proceedings of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing, Tampa, p. 34 (2006)
57. Schmuck, F.B., Haskin, R.L.: GPFS: a shared-disk file system for large computing clusters. In: Proceedings of the FAST '02 Conference on File and Storage Technologies, Monterey, pp. 231–244 (2002)
58. Seo, S., Amer, A., Balaji, P., Bordage, C., Bosilca, G., Brooks, A., Carns, P.H., Castelló, A., Genet, D., Hérault, T., Iwasaki, S., Jindal, P., Kalé, L.V., Krishnamoorthy, S., Lifflander, J., Lu, H., Meneses, E., Snir, M., Sun, Y., Taura, K., Beckman, P.H.: Argobots: A lightweight low-level threading and tasking framework. IEEE Trans. Parallel Distrib. Syst. **29**(3), 512–526 (2018)

59. Sikich, D., Di Natale, G., LeGendre, M., Moody, A.: mpiFileUtils: A Parallel and Distributed Toolset for Managing Large Datasets. Tech. rep., Lawrence Livermore National Lab.(LLNL), Livermore (2017)
60. Slurm—schedmd. http://www.schedmd.com
61. Smith, W.: Prediction services for distributed computing. In: IEEE International Parallel and Distributed Processing Symposium, pp. 1–10. IEEE, Piscataway (2007)
62. Soumagne, J., Kimpe, D., Zounmevo, J.A., Chaarawi, M., Koziol, Q., Afsahi, A., Ross, R.B.: Mercury: enabling remote procedure call for high-performance computing. In: 2013 IEEE International Conference on Cluster Computing, CLUSTER 2013, Indianapolis, pp. 1–8 (2013)
63. Soysal, M.: fhfgs mailing list discussion (2017). https://groups.google.com/d/msg/fhgfs-user/g8ysFS35Ucs/E-RtxKyiCAAJ
64. Soysal, M., Berghoff, M., Klusáček, D., Streit, A.: On the quality of wall time estimates for resource allocation prediction. In: Proceedings of the 48th International Conference on Parallel Processing: Workshops, ICPP 2019, pp. 23:1–23:8. ACM, New York, (2019). https://doi.org/10.1145/3339186.3339204
65. Soysal, M., Berghoff, M., Streit, A.: Analysis of job metadata for enhanced wall time prediction. In: Proceedings of the 22nd International Workshop on Job Scheduling Strategies for Parallel Processing, JSSPP 2018, Vancouver. Revised selected papers, Klusáček, D (ed.) Lecture Notes in Computer Science, vol. 11332, p. 14 S. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-10632-4_1. 46.11.01; LK 01
66. Steinbuch Center for Computing (SCC). http://www.scc.kit.edu (2016). Accessed 16 August 2016
67. Bull HPC-Cluster Taurus (2019). https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/SystemTaurus. Accessed 25 July 2019
68. Thapaliya, S., Bangalore, P., Lofstead, J.F., Mohror, K., Moody, A.: Managing I/O interference in a shared burst buffer system. In: 45th International Conference on Parallel Processing, ICPP 2016, Philadelphia, pp. 416–425 (2016)
69. Thinkparq, BeeGFS: Beegfs the leading parallel cluster file system (2018). https://www.beegfs.io/docs/BeeGFS_Flyer.pdf
70. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Backfilling using system-generated predictions rather than user runtime estimates. IEEE Trans. Parallel Distrib. Syst. **18**(6), 789–803 (2007)
71. Vef, M., Moti, N., Süß, T., Tacke, M., Tocci, T., Nou, R., Miranda, A., Cortes, T., Brinkmann, A.: Gekkofs—a temporary burst buffer file system for HPC applications. J. Comput. Sci. Technol. **35**(1), 72–91 (2020)
72. Vef, M., Moti, N., Süß, T., Tocci, T., Nou, R., Miranda, A., Cortes, T., Brinkmann, A.: Gekkofs—a temporary distributed file system for HPC applications. In: IEEE International Conference on Cluster Computing, CLUSTER 2018, Belfast, pp. 319–324 (2018)
73. Vef, M.A.: Analyzing file create performance in IBM spectrum scale. Master's thesis, Johannes Gutenberg University Mainz (2016). http://www.staff.uni-mainz.de/vef/pubs/vef2016thesis.pdf
74. Vef, M.A., Tarasov, V., Hildebrand, D., Brinkmann, A.: Challenges and solutions for tracing storage systems: a case study with spectrum scale. ACM Trans. Storage **14**(2), 18:1–18:24 (2018)
75. Vilayannur, M., Nath, P., Sivasubramaniam, A.: Providing tunable consistency for a parallel file store. In: Proceedings of the FAST '05 Conference on File and Storage Technologies, San Francisco (2005)
76. Voigt, V.: Entwicklung eines on-demand burst-buffer-plugins fuer HPC-batch-systeme
77. Wang, F., Xin, Q., Hong, B., Brandt, S.A., Miller, E., Long, D., McLarty, T.: File system workload analysis for large scale scientific computing applications. Tech. rep., Lawrence Livermore National Laboratory (LLNL), Livermore (2004)
78. Wang, T., Mohror, K., Moody, A., Sato, K., Yu, W.: An ephemeral burst-buffer file system for scientific applications. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2016, Salt Lake City, pp. 807–818 (2016)

79. Xing, J., Xiong, J., Sun, N., Ma, J.: Adaptive and scalable metadata management to support a trillion files. In: Proceedings of the ACM/IEEE Conference on High Performance Computing, SC 2009, Portland (2009)
80. Yang, S., Ligon III, W.B., Quarles, E.C.: Scalable distributed directory implementation on orange file system. In: Proceedings of the IEEE International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI) (2011)

AIMES: Advanced Computation and I/O Methods for Earth-System Simulations

Julian Kunkel, Nabeeh Jumah, Anastasiia Novikova, Thomas Ludwig, Hisashi Yashiro, Naoya Maruyama, Mohamed Wahib, and John Thuburn

Abstract Dealing with extreme scale earth system models is challenging from the computer science perspective, as the required computing power and storage capacity are steadily increasing. Scientists perform runs with growing resolution or aggregate results from many similar smaller-scale runs with slightly different initial conditions (the so-called ensemble runs). In the fifth Coupled Model Intercomparison Project (CMIP5), the produced datasets require more than three Petabytes of storage and the compute and storage requirements are increasing significantly for CMIP6. Climate scientists across the globe are developing next-generation models based on improved numerical formulation leading to grids that are discretized in alternative forms such as an icosahedral (geodesic) grid. The developers of these models face similar problems in scaling, maintaining and optimizing code. Performance portability and the maintainability of code are key concerns of scientists as, compared to industry projects, model code is continuously revised and extended to incorporate further levels of detail. This leads to a rapidly growing code base that is rarely refactored. However, code modernization is important to maintain productivity

J. Kunkel (✉)
University of Reading, Reading, England
e-mail: j.m.kunkel@reading.ac.uk

N. Jumah
Universität Hamburg, Hamburg, Germany

A. Novikova
Fraunhofer IGD, Rostock, Germany

T. Ludwig
Deutsches Klimarechenzentrum (DKRZ), Hamburg, Germany

H. Yashiro · N. Maruyama
RIKEN, Tokyo, Japan

M. Wahib
AIST Tokyo, Tokyo, Japan

J. Thuburn
University of Exeter, Exeter, England

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_5

of the scientist working with the code and for utilizing performance provided by modern and future architectures. The need for performance optimization is motivated by the evolution of the parallel architecture landscape from homogeneous flat machines to heterogeneous combinations of processors with deep memory hierarchy. Notably, the rise of many-core, throughput-oriented accelerators, such as GPUs, requires non-trivial code changes at minimum and, even worse, may necessitate a substantial rewrite of the existing codebase. At the same time, the code complexity increases the difficulty for computer scientists and vendors to understand and optimize the code for a given system. Storing the products of climate predictions requires a large storage and archival system which is expensive. Often, scientists restrict the number of scientific variables and write interval to keep the costs balanced. Compression algorithms can reduce the costs significantly but can also increase the scientific yield of simulation runs. In the AIMES project, we addressed the key issues of programmability, computational efficiency and I/O limitations that are common in next-generation icosahedral earth-system models. The project focused on the separation of concerns between domain scientist, computational scientists, and computer scientists.The key outcomes of the project described in this article are the design of a model-independent Domain-Specific Language (DSL) to formulate scientific codes that can then be mapped to architecture specific code and the integration of a compression library for lossy compression schemes that allow scientists to specify the acceptable level of loss in precision according to various metrics. Additional research covered the exploration of third-party DSL solutions and the development of joint benchmarks (mini-applications) that represent the icosahedral models. The resulting prototypes were run on several architectures at different data centers.

1 Introduction

The problems on the frontier of science requires extreme computational resources and data volumes across the disciplines. Examples of processes include the understanding of the earth mantle [10], plasma fusion [24], properties of steel [5], and the simulation of weather and climate. The simulation of weather and climate requires to model many physical processes such as the influence of radiation from the sun and the transport of air and water in atmosphere and ocean [8]. As these processes are complex, scientists from different fields collaborate to develop models for climate and weather simulations.

The mathematical model of such processes is discretized and encoded as computer model using numerical methods [53]. Different numerical methods can be used to approximate the mathematical models. A range of different numerical methods are used, including finite differences, finite volumes, and finite elements. All of these methods partition the domain of interest into small regions and apply stencil computations to approximate operations such as derivatives.

The necessary computations include variables (fields) like temperature or pressure distributed spatially over some surface or space—the problem domain. Simple techniques divide a surface into rectangular smaller regions covering the whole domain. Such rectangular grids have a simple regular structure. Those grids fit computations well as the grid structure simply corresponds to the array notation of the programming languages. However, applying this grid to the globe leads to variable sizes of grid cells, e.g., the equator region has a coarse grid while the polar regions are a singularity. With such a shortcoming, rectangular grids are well suited for regional models but not for a global model.

Therefore, recent models targeting global simulations are developed using different grids. Moving to such alternative grids allows to solve the cell area problem for global models, but the formulation of the models is more complicated. Icosahedral grids are examples of such alternatives. An icosahedral grid results from projecting an icosahedron onto the surface of the globe. The surface of the globe is then divided into twenty spherical triangles with equal areas. Grid refinement is achieved with recursive division of the spherical arcs into halves. The resulting points of the division form four smaller spherical triangles within each spherical triangle. Such refinement is repeated until the needed resolution is reached. Icosahedral grids can be used with the triangles as the basic cell, but also hexagons can be synthesized.

Icosahedral grids have approximately uniform cell area and can be used for global models avoiding the cell area differences in contrast to the rectangular grids. However, complications arise when thinking of the technical side, where we need to know how to map the field data into data structures. Such technical details are challenging with the performance demand for the models.

The values of a field in the simulation is localized with respect to the grid cell depending on the numerical formulation. In one method, values of a field are localized at the centers of the cells—this can be a single value or multiple values with higher order methods. However, other methods localize values on the vertices, while others reside on the edges separating the cells (see Fig. 1). How the cells are connected to each other, i.e., the neighbors and orientation of the cells, is defined in the connectivity. A problem domain can be organized in a regular

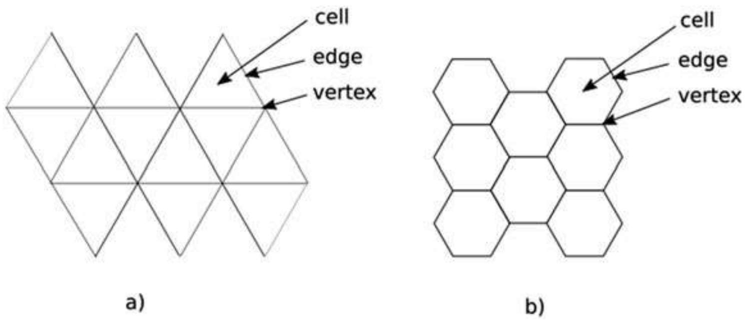


Fig. 1 Icosahedral grids and variables. (**a**) Triangular grid. (**b**) Hexagonal grid

fashion into so-called structured grids—following an easy schema to identify (left, right, . . .) neighbors. Unstructured grids can follow a complex connectivity, e.g., using smaller and larger cells, or covering complex surfaces but require to store the connectivity information explicitly. The modern models explore both schemes due to their benefit; for instance, the structured grid can utilize compiler optimizations more effectively, while unstructured grids allow local refinement around areas of interest.

General-purpose languages (GPL), e.g., Fortran, are widely used to encode the discretized computer model. The simulation of the earth with a high resolution like 1 km^2 that is necessary to cover many smaller-scale physical processes, requires a huge computation effort and likewise storage capacity to preserve the results for later analysis. Due to uncertainty, scientists run a single experiment many times, multiplying the demand for compute and storage resources. Thus, the optimization of the codes for different architectures and efficiency is of prime importance to enable the simulations.

With existing solutions, scientists rewrite some code sections repeatedly with different optimization techniques that utilize the capabilities of the different machines. Hence, scientists must learn different optimization techniques for different architectures. The code duplication brings new issues and complexities concerning the development and the maintainability of the code.

Thus, the effort from the maintainers and developers of the models, who are normally scientists and not computer scientists, is substantial. Scientists' productivity is an important point to consider as they do activities that should not be their focus. Maintaining model codes throughout the lifecycle of the model is a demanding effort under all the mentioned challenges.

The structure of the icosahedral grids brings complications not only to the computation, but also to the storage of the field data. In contrast to regular grids, where multi-dimensional array notation fits to hold the data, icosahedral grids do not necessarily map directly to simple data structures. Besides the challenge of file format support, modern models generate large amounts of data that impose pressure on storage systems. Recent models are developed with higher-resolution grids, and include more fields and processes. Simulations writing terabytes of data to the storage system push towards optimizing the use of the storage by applying data-reduction techniques.

The development of simulation models unfolds many challenges for the scientific community. Relevant challenges for this project are:

- **Long life**: The lifecycle of earth system models is long in comparison to the turnover of the computer technology mainly in terms of processor architectures.
- **Performance and efficiency**: The need for performance and the optimal use of the hardware resources is an important issue.
- **Performance-portability**: Models are run on different machines and on different architectures. They must use the available capabilities effectively.

- **Collaboration**: Another point is the collaborative efforts to develop models—involving PhD students to contribute pieces of science code to a large software project—that complicates the maintenance and software engineering of the models.
- **Data volume**: the large amounts of data must be handled efficiently.

1.1 The AIMES Project

To address challenges facing earth system modeling, especially for icosahedral models, the project *Advanced Computation and I/O Methods for Earth-System Simulations* (AIMES) investigated approaches to mitigate the aforementioned programming and storage challenges.

The AIMES project is part of the SPPEXA program and consisted of the consortium:

- Universität Hamburg, Germany
- Institut Pierre Simon Laplace (IPSL), Université Versailles Saint-Quentin-en-Yvelines, France
- RIKEN Advanced Institute for Computational Science, Japan
- Global Scientific Information and Computing Center, Tokyo Institute of Technology, Japan

The project started in March 2016 with plans for 3 years.

The main objectives of the project were: (1) enhance programmability; (2) increase storage efficiency; (3) provide a common benchmark for icosahedral models. The project was organized in three work packages covering these aspects and a supplementary project management work package to achieve the three project objectives. The strategy of the work packages is layed out in the following.

Under the first work package, higher-level scientific concepts are used to develop a dialect for each of three icosahedral models: DYNAMICO [17], ICON [54], and NICAM [47]. A domain-specific language (DSL) is the result of finding commonalities in the three dialects. Also, a light weight source-to-source translation tool is developed. Targeting different architectures to run the high-level code is an important aspect of the translation process. Optimizations include applying parallelization to the different architectures. Also, providing a memory layout that fits the different architectures is considered.

Under the second work package, data formats for icosahedral models are investigated to deal with the I/O limitations. Lossy compression methods are developed to increase storage efficiency for icosahedral models. The compression is guided with user-provided configuration to allow to use suitable compression according to the required data properties.

Under the third work package, relevant kernels are selected from the three icosahedral models. A mini-IGCM is developed based on each of the three models to offer a benchmark for icosahedral models. Developed code is used to evaluate the DSL and the compression of icosahedral global modeling.

The **key outcomes** of the project described in this article are: (1) the design of an effective light-weight DSL that is able to abstract the scientific formulation from architecture-specific capabilities; (2) the development of a compression library that separates the specification of various qualities that define the tolerable error of data from the implementation. We provide some further large-scale results of our compression library for large scale runs extending our papers about the library [36, 37]; (3) the development of benchmarks for the icosahedral models that are mini-applications of the models.

Various additional contributions were made that are summarized briefly with citations to the respective papers:

- We researched the impact of lossless data compression on energy consumption in [2]. In general, the energy consumption increases with the computational intensity of the compression algorithm. However, there are algorithms that are efficient and less computational intense that can improve the energy-efficiency.
- We researched compilation time of code that is generated from the DSL using alternative optimization options on different compilers [20]. Different optimization options for different files allow different levels of performance, however, compilation time is also an important point to consider. Results show that some files need less optimization focus while others need further care. Small performance drops are measured with considerable reduction in compile times when the suitable compilation options are chosen.
- We researched annotating code for instrumentation automatically by our translation tool, to identify resource consuming kernels [21]. Instrumentation allows to better find where to focus the optimization efforts. We used the DSL translation tool to annotate kernels and make generated code ready for instrumentation. As a result performance measurements were recorded with reduced effort as manual preparations are not needed anymore.
- We researched applying vector folding to icosahedral grid codes in a bachelor thesis [49]. Vector folding allows to improve use of caches by structuring data in a way accounting for caches and data dimensionality. Results show that vector folding was difficult to apply manually to icosahedral grids. Performance was raised but not significantly as a result of the needed effort that should be invested to rewrite kernels with this kind of optimization.
- We involved ASUCA and the use of Hybrid Fortran [43] to port original CPU code to GPUs, to look at a different model with different requirements.

This article is structured as follows: first, the scope of the state-of-the-art and related work is sketched in Sect. 2. In Sect. 3, an alternative development approach for code-modernization is introduced. Various experiments to evaluate the benefit of the approach are shown in Sect. 4. The compression strategy is described in Sect. 5 and evaluated in Sect. 6. The benchmarks for the icosahedral models are discussed in Sect. 7. Finally, the article is concluded in Sect. 8.

2 Related Work

The related work covers research closely related to domain-specific languages in climate and weather and the scientific data compression.

2.1 Domain-Specific Languages

DSLs represent an important approach to provide performance portability and support model development. A DSL is always developed having a particular domain in mind. Some approaches support multiple layers of abstraction. A high-level abstraction for the finite element method is provided with Firedrake [44]. The ExaStencils pipeline generally addresses stencil codes and their operations [18, 33] and many research works introduce sophisticated schemes for the optimization of stencils [7, 9].

One of the first DSLs which were developed to support atmospheric modeling is Atmol [52]. Atmol provided a DSL to allow scientists to describe their models using partial differential equations operators. Later, Liszt [14] provided a DSL for constructing mesh-based PDE solvers targeting heterogeneous systems.

Multi-target support was also provided by Physis [42]. Physis is a C-based DSL which allows developing models using structured grids.

Another form of DSL is Icon DSL [50]. Icon DSL was developed to apply index interchanges based on described swapping on Fortran-based models.

Further work based on C++ constructs and generic programming to improve performance portability is Stella [23] and later GridTools [13]. Computations are specified with a C++-based DSL and the tools generate code for CPUs or GPUs. GridTools are used to port some kernels from the NICAM model in our project AIMES to evaluate existing DSLs.

Although C++ provides strong features through generic programming allowing to avoid performance portability issues, scientists are reluctant to utilize alternative programming languages as the existing codes are huge. Normally scientists prefer to keep using preferred languages, e.g. Fortran, rather than moving to learning C++ features.

Other forms of DSLs used directives to drive code porting or optimization. Hybrid Fortran [43], HMPP [16], Mint [51], CLAW [12] are examples of directive-based approach. Such solutions allow adding directives to code to guide some optimization. Scientists write code in some form and add directives that allow tools to provide specific features, e.g., CLAW allows writing code for one column and allows using directives to apply simulations over the set of columns in parallel.

In the MetOffice's LFRic model [1], a DSL is embedded into the Fortran code that provides an abstraction level suitable for the model. The model ships with the PSyclone code-generator that is able to transform the code for different target platforms. In contrast to our lightweight solution, these DSLs are statically defined and require a big translation layer.

2.2 Compression

Data-reduction techniques related to our work can be structured into: (1) algorithms for the lossless data compression; (2) lossy algorithms designed for scientific (floating-point) data and the HPC environment; (3) methods to identify necessary data precision and for large-scale evaluation.

Lossless Algorithms There exist various lossless algorithms and tools, for example, the LZ77 [55] algorithm which utilizes dictionaries. By using sliding windows, it scans uncompressed data for two largest windows containing the same data and replaces the second occurrence with a pointer to the compressed data of the first. The different lossless algorithms vary in their performance characteristics and ability to compress data depending on its characteristics. A key limitation is that users have to pick an algorithm depending on the use case. In [25], we presented compression results for the analysis of typical climate data. Within that work, the lossless compression scheme MAFISC with preconditioners was introduced. With MAFISC, we also explored the automatic selection of algorithms by compressing each block with two algorithms, the best compression chain so far and one randomly chosen. It compresses data 10% more than the second best algorithm (e.g., standard compression tools).

Lossy Algorithms for Floating-Point Data SZ [15] and ZFP [41] are the de-facto standard for compressing floating-point data in lossy mode. Both provide a way of bounding the error (either bit precision or absolute error quantities) but only one quantity can be selected at a time. ZFP [41] can be applied up to three dimensions. SZ is a newer and effective HPC data compression method; it uses a predictor and the lossless compression algorithm GZIP. Its compression ratio is typically significantly better than the second-best solution of ZFP. In [26], two lossy compression algorithms (GRIB2, APAX) were evaluated regarding the loss of data precision, compression ratio, and processing time on synthetic and climate dataset. These two algorithms have equivalent compression ratios and depending on the dataset APAX signal quality may exceed GRIB2 and vice versa.

Methods The application of lossy techniques to scientific (floating-point) datasets is discussed in [11, 22, 27, 38–40]. A statistical method to predict characteristics (such as proportions of file types and compression ratio) of stored data based on representative samples was introduced in [34] and the corresponding tool in [35]. It can be used to determine compression ratio by scanning a fraction of the data, thus reducing costs.

Efforts for determination of appropriate levels of precision for lossy compression methods for climate and weather data were presented in [3] and in [4]. The basic idea is to compare the statistics derived from the data before and after applying lossy compression schemes; if the scientific conclusions drawn from the data are similar and indistinguishable without the compression, the loss of precision is acceptable.

3 Towards Higher-Level Code Design

Computations in earth system modeling run hundreds or thousands of stencil computations over wide grids with huge numbers of points. Such computations are time consuming and are sensitive to optimal use of computer resources. Compilers usually apply a set of optimizations while compiling code. Semantical rules of the general purpose language (GPL) can be applied to the source code and used within compilers to translate the code to semantically equivalent code that runs more efficiently. However, often the semantical information extracted from the source code are not enough to apply all relevant optimizations as some high-level optimizations would alter the semantics—and the rules of the GPL forbid such changes. As mostly code from GPLs is at a lower semantical level than the abstraction level of the developers, opportunity of optimization is lost. Such lost opportunities are a main obstacle to develop software in a performance-portable way in earth system modeling.

To address the lost opportunities of optimization, different techniques are applied by the scientists directly to the source code. Thus, it is the responsibility of the scientists who develop the models to write the code with those decisions and guidelines on optimizations in mind. Drawbacks of this strategy include pushing scientists to focus on machine details and optimal code design to use hardware resources. Scientists need to learn the relevant optimization techniques from computer science such as, e.g., cache blocking.

3.1 Our Approach

To solve the issues with the manual code optimization, we suggest using an additional set of language constructs which exhibit higher-level semantics. This way, tools can be used to apply optimizations based on those semantics. Optimization responsibilities are moved again from scientists to tools.

As usually, the source code is developed by scientists, however, in our approach, instead of coding on low-level and caring for optimization strategies, a DSL is used as abstraction. Machine-specific or computer scientific concepts are not needed to write the source code of a model. This is enabled by increasing the abstraction level of a GPL by providing a language extension with semantics based on the scientific concepts from the domain science itself.

The DSL implements a template mechanism to simplify and abstract the code. For the purpose of this project, it was designed to abstract climate and weather scientific concepts but other domains and models could be supported. Therefore, stencils and grids are used as the basis in the DSL. The language extensions hide the memory access and array notations. They also hide the details of applying the stencils and the traversal of the grids. The grid structure itself is hidden in the source code. A model's code uses the grid without specifying the locations of the grid points in memory or how neighbors are accessed. All such details are specified in the configuration files. Lower details are neglected at the DSL level.

Translation tools handle the processing of the higher-level source code and converting it to compatible GPL code. The semantics of the added language extensions are extracted from the source code and are used to apply further high-level transformations to the code. Applied transformations are guided by configuration files that allow users to control the optimization process and may convert the code to various back-end representations that, in-turn, can be converted to code that is executed on a machine. A key benefit of this strategy is that it increases the performance-portability: A configuration can apply optimization techniques exploiting features of a specific architecture. Therefore, a single scientific code base can be used to target multiple different machines with different architectures.

Model-specific configuration files are provided separately to guide the code translation and optimization process. Those files are developed by scientific programmers rather than domain scientists. In contrast to domain scientists, the scientific programmers must have an intermediate understanding of the scientific domain but also understand the hardware architecture of a target machine. They use their experience to generate code that uses the machine's resources, and write the configurations that serve the purpose of optimal use of that specific machine to run the selected model.

Our approach offers separation of concerns between the parties. Scientific work is done by scientists and optimization is done by scientific programmers. The concept of the approach is illustrated in Fig. 2. For the icosahedral models, a single intermediate domain language is derived that can be adjusted for the needs of each model individually (dialects). From this single source various code representations (back-ends) could be generated according to configuration, e.g., MPI+OpenMP or GASPI.

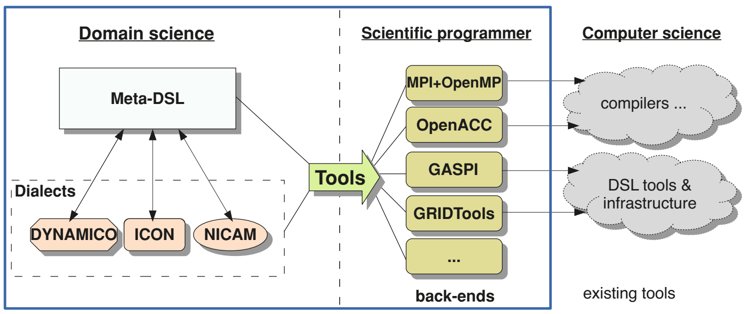


Fig. 2 Separation of concerns

3.2 Extending Modeling Language

An important point we consider in our approach is keeping the general-purpose language, e.g. Fortran, that the scientists prefer to use to develop their model. We add additional language constructs to the GPL. This simplifies the mission to port existing models which could include hundreds of thousands of lines of code. The majority of the code is kept, while porting some parts incrementally; by providing templates in the configuration files, code can be simplified while it still produces equivalent GPL constructs: Changes are replacements of loop details and field access into alternative form using the added extensions. A drawback of the incremental approach is that the full beauty of optimizations like memory adjustments requires to have ported the complete model.

Our plan to develop such language extensions was to start with the three existing modern icosahedral models of the project partners. In the first phase, special dialects were proposed to support each model. Then, we identified common concepts and defined a set of language extensions that support the domain with domain-specific language constructs. We collected requirements, and worked in collaboration with scientists from the three models to reach at the language extensions, in detail:

1. The domain scientists suggested compute-intensive and time-consuming code parts.
2. We analyzed the chosen code parts to find out possibilities to use scientific terms instead of existing code. We always kept in mind that finding a common representation across the three models leads to domain-specific language extensions.
3. We replaced codes with suggested extensions.
4. We discussed the suggestions with the scientists. Discussions and improvements were done iteratively. The result of those discussions lead to the GGDML language extensions.

3.2.1 Extensions and Domain-Specific Concepts

The *General Grid Definition and Manipulation Language (GGDML)* language extensions provide a set of constructs for:

- Grid definition
- Declaration of fields over grids
- Field access and update
- Grid traversal to apply stencils
- Stencil reduction expressions

GGDML code provides an abstraction including the order of the computation of elementary operators. Therefore, optimizations can result in minor changes of the computed result of floating-point operations; this is intentional as bit-reproducibility constraints the optimization potential.

In code that is written with GGDML, scientists can specify the name of the grid that should be traversed when applying a stencil. The definitions of the grids are provided globally through the configuration files. GGDML allows to specify a modified set of grid points rather than the whole set of grid points as provided through the grid definition grid, e.g., traversing a specific vertical level. Such possibilities are offered through naming a grid and using operators that allow adding, dropping, or modifying dimensions of that grid. Operators could change the dimensionality of the grid or override the existing dimensions.

Fields are declared over different grids through declaration specifiers. GGDML provides a flexible solution to support application requirements. A basic set of declaration specifiers allows to control the dimensionality of the grid, and the localization of the field with respect to the grid. Such declaration specifiers allow applications to deal with surfaces and spaces, and also supports using staggered as well as collocated grids.

Access specifiers provide tools the necessary information that will be used to allocate/deallocate and access the fields. Field access is an important part of stencil operations. GGDML provides an iterator statement to apply the stencil operations over a set of grid points. The GGDML iterator statement replaces loops and the necessary optimization to apply stencils. It provides the user an index that refers to the current grid point. Using this index, scientists can write their stencils without the need to deal with the actual data structures that hold the field data. The iterator applies the body to each grid point that is specified in the grid expression, which is one part of the iterator statement. This expression is composed from the name of a grid, and possibly a set of modifications using operators as mentioned above.

The iterator's index alone is not sufficient to write stencil operations, as stencils include access to neighboring points. For this purpose, GGDML uses access operators, which represent the spatial relationships between the grid points. This allows to access the fields that need to be read or written within a stencil operation using spatial terms instead of arrays and memory addresses. To support different kinds of grids, GGDML allows users to define those access operators according to the application needs.

Repetitions of the same mathematical expressions over different neighbors is common in stencil operations. To simplify writing stencils, GGDML provides a reduction expression. Reduction expressions apply a given sub-expression over multiple neighbors along with a mathematical operator applied to the set of the subexpressions.

3.3 Code Example

To demonstrate the code written with extensions, a sample code from the NICAM model written with Fortran is given in Listing 1. As we can see from the original NICAM code, a pattern is repeated in the code: the same field is accessed multiple times over multiple indices. Optimization is limited as firstly, the memory layout is hardcoded in the fields cgrad and scl, secondly the iteration order is fixed. Integrating blocking for cache optimization in this schema would increase the complexity further.

Listing 1 NICAM Fortran code

```
do d = 1, ADM_nxyz
  do l = 1, ADM_lall
!OCL PARALLEL
    ! support indices to address neighbors
    do k = 1, ADM_kall
       do n = OPRT_nstart, OPRT_nend
          ij     = n
          ip1j   = n + 1
          ijp1   = n     + ADM_gall_1d
          ip1jp1 = n + 1 + ADM_gall_1d
          im1j   = n - 1
          ijm1   = n     - ADM_gall_1d
          im1jm1 = n - 1 - ADM_gall_1d

          grad(n,k,l,d) = cgrad(n,l,0,d) * scl(ij    ,k,l) &
                        + cgrad(n,l,1,d) * scl(ip1j  ,k,l) &
                        + cgrad(n,l,2,d) * scl(ip1jp1,k,l) &
                        + cgrad(n,l,3,d) * scl(ijp1  ,k,l) &
                        + cgrad(n,l,4,d) * scl(im1j  ,k,l) &
                        + cgrad(n,l,5,d) * scl(im1jm1,k,l) &
                        + cgrad(n,l,6,d) * scl(ijm1  ,k,l)
       enddo
       grad(          1:OPRT_nstart-1,k,l,d) = 0.0_RP
       grad(OPRT_nend+1:ADM_gall     ,k,l,d) = 0.0_RP
    enddo
  enddo
enddo
```

The same semantics rewritten with the DSL is shown in Listing 2. Instead of iterating across the grid explicitly, a FOREACH loop specifies to run on each element of the grid, the coordinates are encoded in the new cell variable. We reduced the repeated occurrences of the fields with the indices with a ‘REDUCE’ expression. The Fortran indices are replaced with DSL indices that made it possible to simplify the field access expressions.

Listing 2 NICAM DSL code

```
FOREACH cell in grid | g{OPRT_nstart..OPRT_nend}
  do d = 1, ADM_nxyz
        grad(cell,d) = REDUCE(+,N={0..6},
            cgrad(cell%g,cell%l,N,d) * scl(cell%neighbor(N))
                ↪ )
  enddo
END FOREACH
```

```
FOREACH cell in GRID%cells | g{1..OPRT_nstart-1 , OPRT_nend+1
    ↪  .. gall}
  do d = 1, ADM_nxyz
           grad(cell,d) = 0.0_PRECISION
  enddo
END FOREACH
```

3.4 Workflow and Tool Design

Model code that is based on the DSL along with code of the model in general-purpose language goes through a source-to-source translation process. This step is essential to make the higher-level code ready for processing by the compilers. Our tool is designed in a modular architecture. By applying a configuration, it translates code in a file or a source tree and generates a version of the transformed code. The generated code is the optimized version for a specific machine/architecture according to the configuration. Inter-file optimizations in code trees can also be detected and applied.

The tool is implemented with Python3. Users call the main module with parameters that allow them to control the translation process, e.g. to specify a language module. The code tree is provided to the main module also as an argument.

The main module loads the other necessary modules. The DSL handler module constructs the necessary data structures according to the user-provided configuration file. The source code tree is then parsed into abstract syntax trees (AST). The generated ASTs can be analyzed for optimization among the source files. After all the optimizations/transformations are applied, the resulting code tree is serialized to the output file. Figure 3 shows the design of the translation process.

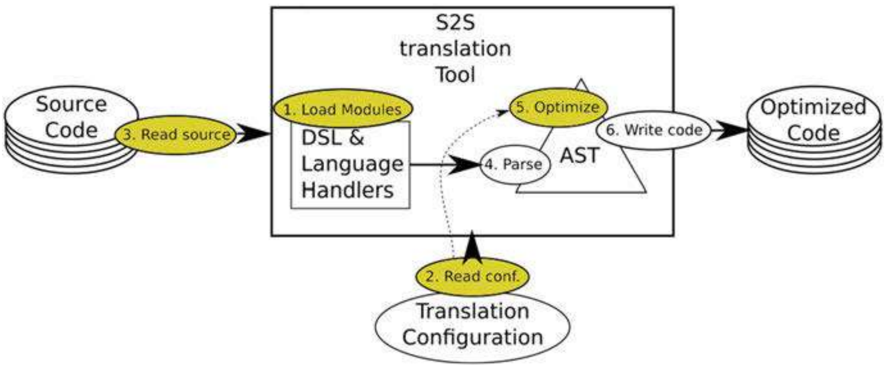


Fig. 3 Translation process. Yellow components are influenced by the user options

3.5 Configuration

Configuration files include multiple sections, among which some are essential and others can be added only if needed. Optimization procedures are driven by those configuration sections. The translation tool uses defaults in case optional sections are missing.

Blocking Among the important optimizations that help improve the performance of stencil computations is the optimal use of the caches and memory access. Reusing the data in the caches eliminates the need to read the same data elements repeatedly from memory. Often, data locality can be exploited in stencil computations, allowing for performance improvements.

Cache blocking is a technique to improve the data reuse in caches. Our translation process can apply cache blocking based on the scientific programmer's recommendations. One configuration section is used to allow to specify cache blocking information. The default when this section is missing in a configuration file is to not apply cache blocking.

An example kernel using GGDML in the C programming language is shown in Listing 3.

Listing 3 Example kernel using C with GGDML

```
foreach c in grid
{
  float df=(f_F[c.east_edge()]-f_F[c.west_edge()])/dx;
  float dg=(f_G[c.north_edge()]-f_G[c.south_edge()])/dy;
  f_HT[c]=df+dg;
}
```

Applying cache blocking using a configuration file defining 10,000 elements per block generates the loop code shown in Listing 4. The first loop handles completely occupied blocks with 10 k elements and the second loop the remainder. In both cases, the loop body (not shown) contains the generated stencil code.

Listing 4 Example loop structure for blocking

```
for (size_t blk_start = (0); blk_start < (GRIDX); blk_start
    ↪ +=10000){
    size_t blk_end = GRIDX;
    if ((blk_end - blk_start) > 10000)
        blk_end = blk_start + 10000;
    // Generated loop body
}
#pragma omp simd
for (size_t XD_index = blk_start; XD_index < blk_end;
    ↪ XD_index++) {
  // Generated loop body
}
```

Memory Layout Another important point to optimize memory bandwidth is to optimize the data layout in memory. The temporal and spacial locality of data should lead to access of data in complete cache lines such that it can be prefetched and cached effectively. Thus, data that is accessed in short time frames should be stored closer in memory. To exploit such possibilities, our translation tool provides a flexible layout transformation procedure. The DSL itself abstracts from data placement, however, the translation process generates the actual data accesses. This layout specification is described in configuration files.

Besides to data layout control, the loop nests that access the field data are also subject to user control. The order of the loops that form a nested loop is critical for the optimal data access. Loop order of loops that apply the stencils is also controlled by configuration files.

Listing 5 illustrates the resulting code after using a data layout transformation. In this case, a 2D grid is stored in a single-dimensional array.

Listing 5 Example code generated with index transformation

```
[...]
#pragma omp for
for (size_t YD_index = (0); YD_index < (local_Y_Cregion);
    ↪ YD_index++){
#pragma omp simd
   for (size_t XD_index = blk_start; XD_index < blk_end;
       ↪ XD_index++){
    float df = (f_F[(YD_index + 1) * (GRIDX + 3) + (XD_index
        ↪ + 1)+1]
        - f_F[(YD_index + 1) * (GRIDX + 3) + (XD_index) + 1])
            ↪  * invdx;
    float dg = (f_G[((YD_index + 1)+1) * (GRIDX + 3) + (
        ↪ XD_index)+1]
        - f_G[(YD_index + 1) * (GRIDX + 3) + (XD_index) + 1])
            ↪  * invdy;
    f_HT[(YD_index + 1) * (GRIDX + 3) + (XD_index) + 1] = df
        ↪ + dg;
   }
}
[...]
```

Inter-Kernel Optimization Cache blocking and memory layout allow improving the use of the caches and memory bandwidth at the level of the kernel. However, the optimal code at the kernel level does not yet guarantee optimal use of caches and memory bandwidth at the application level. Consider the example where two kernels share most of their input data but compute different outputs independently from each other. These kernels could be fused together and benefit from reusing cached data. Note that the benefit is system specific, as the size of the cache and application kernel determine the optimal size for blocking and fusion.

The inter-kernel optimization allows exploiting such data reuse across kernels. To exploit such potential, our translation tool can run an optimizer procedure to detect such opportunities and to apply them according to user decision of whether to apply any of those optimizations or not.

Therefore, the tool analyzes the calls among the code files within the code tree. This analysis leads to a list of call inlining possibilities. The inlining could lead to further optimization through loop fusions. The tool runs automatic analysis including data dependency and code consistency. This analysis detects possible loop fusions that still keep computations consistent. Such loop fusion may lead to optimized use of memory bandwidth and caches. We tested the technique experimentally (refer to Sect. 4.5) to merge kernels in the application. We could improve the use of caches and hence the performance of the application with 30–48% on different architectures.

The tool provides a list of possible inlining and loop fusion cases. Users choose from the list which case to apply—we anticipate that scientific programmers will make an informed choice for a target platform based on performance analysis tools. According to the choice that the user makes, the tool applies the corresponding transformations automatically.

Listing 6 shows two kernels to compute the two components of the flux.

Listing 6 Example code with two kernels to compute flux components

```
[...]
#pragma omp parallel for
for(size_t YD_index = (0); YD_index < (local_Y_Eregion);
    ↪ YD_index++){
#pragma omp simd
   for (size_t XD_index = blk_start; XD_index < blk_end;
      XD_index++) {
      f_F[YD_index][XD_index] = f_U[YD_index][XD_index] *
      (f_H[YD_index][XD_index] +f_H[YD_index][XD_index -1])
          ↪ /2.0;
   }
}
[...]
#pragma omp parallel for
for(size_t YD_index = (0); YD_index < (local_Y_Eregion);
    ↪ YD_index++){
#pragma omp simd
   for (size_t XD_index = blk_start; XD_index < blk_end;
       ↪ XD_index++){
      f_G[YD_index][XD_index] = f_V[YD_index][XD_index] *
      (f_H[YD_index][(XD_index] + f_H[YD_index -1][XD_index])
          ↪ /2.0;
   }
}
[...]
```

Listing 7 shows the resulting code when the two kernels are merged.

Listing 7 Merged version of the flux computation kernels

```
[...]
#pragma omp parallel for
for (size_t YD_index = 0; YD_index < (local_Y_Eregion);
    ↪ YD_index++){
#pragma omp simd
  for (size_t XD_index = blk_start; XD_index < blk_end;
      ↪ XD_index++) {
    f_F[YD_index][XD_index] = f_U[YD_index][XD_index] *
    (f_H[YD_index][XD_index] + f_H[YD_index][(XD_index) +
        ↪ (-1)])/2.0;

    f_G[YD_index][XD_index] = f_V[YD_index][XD_index] *
    (f_H[YD_index][(XD_index] + f_H[(YD_index) + (-1)][
        ↪ XD_index])/2.0;
  }
}
[...]
```

Utilizing Distributed Memory Beyond parallelization on the node resources, our techniques allow scaling the same source code that uses GGDML over multiple nodes utilizing distributed memory. This is essential to run modern models on modern supercomputer machines.

The GGDML code is unaware of underlying hardware, and does not need to be modified to run on multiple nodes. Rather, configuration files are prepared to translate the GGDML code into code that is ready to be run on multiple nodes. Configuration files allow domain decomposition to distribute the data and the computation over the nodes. Necessary communication of halo regions is also enabled through configuration files. Scientific programmers can generate simple parallelization schemes, e.g., MPI using blocking communication or sophisticated alternatives like non-blocking communication. When using non-blocking communication, a further optimization is to decouple the computation of the inner region that can be calculated without needing updated halo data and outer regions that require data from another process to be computed.

The translation tool extracts neighborhood information from the GGDML extensions. Such extracted information is analyzed by the tool to decide which halo regions should be exchanged between which nodes. Decisions and information from configuration files allow to generate the necessary code that handles the communication and the synchronization. This guarantees that the necessary data are consistent on the node where the computation takes place.

The parallelization is a flexible technique. No single library is used to handle the parallelization, rather, the communication library is provided through configuration files. Thus, different libraries or library versions can be used for this purpose. We have examined the use of MPI and GASPI as libraries for parallelization.

Listing 8 shows the resulting communication code of halo regions between multiple processes—in this case without decoupling of inner and outer area, code with decoupled areas is longer. In this listing, dirty flags are generated to communicate only necessary data. Flags are set global to all processes, and can be checked by each process such that processes that need to do communication can make use of them. This way we guarantee to handle communication properly.

Listing 8 Example generated code to handle communication of halo data

```
[...]
//part of the halo exchange code
if (f_G_dirty_flag[11] == 1) {
  if (mpi_world_size > 1) {
    comm_tag++;
    int pp = mpi_rank != 0 ? mpi_rank - 1 : mpi_world_size -
        ↪ 1;
    int np = mpi_rank != mpi_world_size - 1 ? mpi_rank + 1 :
        ↪ 0;
    MPI_Isend(f_G[0], GRIDX + 1, MPI_FLOAT, pp,
              comm_tag, MPI_COMM_WORLD, &mpi_requests[0]);
    MPI_Irecv(f_G[local_Y_Eregion], GRIDX + 1, MPI_FLOAT, np,
              comm_tag, MPI_COMM_WORLD, &mpi_requests[1]);
    MPI_Waitall(2, mpi_requests, MPI_STATUSES_IGNORE);
[...]

#pragma omp parallel for
for(size_t YD_index = (0); YD_index < (local_Y_Cregion);
    ↪ YD_index++){
#pragma omp simd
  for (size_t XD_index = blk_start; XD_index < blk_end;
      ↪ XD_index++){
   float df = (f_F[YD_index][(XD_index) + (1)]
      - f_F[YD_index][XD_index]) * invdx;
   float dg = (f_G[(YD_index) + (1)][XD_index]
      - f_G[YD_index][XD_index]) * invdy;
   f_HT[YD_index][XD_index] = df + dg;
  }
}
[...]
```

3.6 *Estimating DSL Impact on Code Quality and Development Costs*

To estimate the impact of using the DSL on the quality of the code and the costs of model development, we took two relevant kernels from each of the three icosahedral models, and analyzed the achieved code reduction in terms of lines

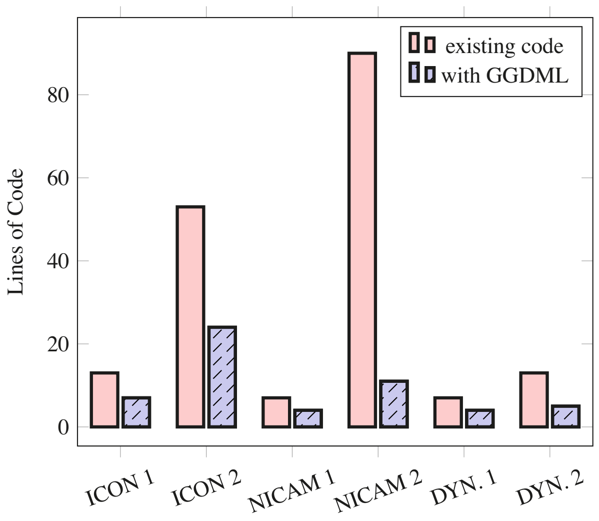


Fig. 4 GGDML impact on the LOC on several scientific kernels [32]

of code (LOC) [32]. We rewrote the kernels (originally written in Fortran) using GGDML + Fortran. Results are shown in Fig. 4.

The average reduction in terms of LOC is 70%, i.e. LOC in GGDML+Fortran in comparison to original Fortran code is 30%. More reduction is noticed in some stencils (NICAM example No.2, reduced to 12%).

Influence on readability and maintainability: Using COCOMO [6] as a model to estimate complexity of development effort and costs, we estimated in Table 1 the benefits as a result of the code reductions when applying GGDML to develop a model comparable to the ICON model. The table shows the effort in person month, development time and average number of people (rounded) for three development modes: the embedded model is typically for large project teams a big and complex code base, the organic model for small code and the semi-detached mode for in-between. We assume the semi-detached model is appropriate but as COCOMO was developed for industry projects, we don't want to restrict the development model. The estimations are based on a code with 400KLOC, where 300KLOC of the code are the scientific portion that allows for code reduction while 100KLOC are infrastructure.

From the predicted developed effort, it is apparent that the code reductions would be leading to a significant effort and cost reduction that would justify the development and investment in DSL concepts and tools.

4 Evaluating Performance of our DSL

To illustrate the performance benefits of using the DSL for modeling, we present some performance measurements that were measured for example codes written with the DSL and translated considering different optimization aspects (configura-

Table 1 COCOMO cost estimates [32]

Development Style	Codebase	Effort applied (person month)	Dev. Time (months)	People required	Dev. costs (M€)
Embedded	Fortran	4773	37.6	127	23.9
	DSL	1133	28.8	72	10.4
Semi-detached	Fortran	2462	38.5	64	12.3
	DSL	1133	29.3	39	5.7
Organic	Fortran	1295	38.1	34	6.5
	DSL	625	28.9	22	3.1

tions). Two different testcodes were used to evaluate the DSL's support for different types of grids: One application uses an unstructured triangular grid, and the other uses a structured rectangular grid that could be applied for code in cubic sphere. Both were written using GGDML (our DSL) in addition to C as the host modeling language.

4.1 Test Applications

Laplacian Solver The first application code uses an unstructured triangular grid covering the surface of the globe. The application was used in the experiments to apply the Laplacian operator of a field at the cell centers based on field values at neighboring cells. Generally, this code includes fields that are localized at the cell centers, and on the edges of the cells. The horizontal grid of the globe surface is mapped to a one dimensional array using Hilbert space-filling-curve. We used 1,048,576 grid points (and more points over multiple-node runs) to discretize the surface of the globe. The code is written with 64 vertical levels. The surface is divided into blocks. The kernels are organized into components, each of which resembles a scientific process.

Shallow Water Model The other code is a shallow water equation solver. It is developed with a structured grid. Structured grids are also important to study for icosahedral modeling, as some icosahedral grids can be structured. Fields are located at centers of cells and on edges between cells. This solver uses the finite difference method. The test code is available online.[1] As part of the testing, we investigate performance portability of code developed using the DSL.

4.2 Test Systems

The experiments were executed in different times during the course of the project and used different machines based on availability and architectural features.

- Mistral
  The German Climate Computing Center provides nodes with Intel(R) Xeon(R) E5-2695 v4 (Broadwell) @ 2.1 GHz processors.
- Piz Daint
  The Swiss supercomputer provides nodes equipped with two Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60 GHz processors and NVIDIA(R) Tesla(R) P100 GPUs.

[1] https://github.com/aimes-project/ShallowWaterEquations.

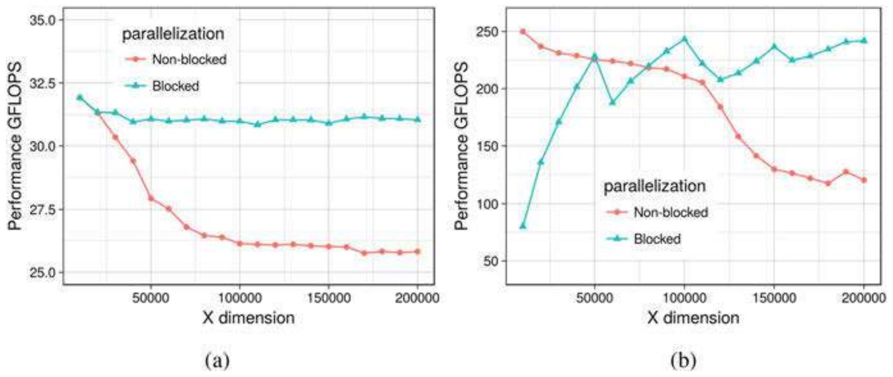


Fig. 5 Impact of blocking: Performance measurements with variable grid width. (**a**) Broadwell CPU. (**b**) P100 GPU

- NEC test system
  Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30 GHz Broadwell processors with Aurora vector engines.

4.3 Evaluating Blocking

To explore the benefit of organizing memory accesses efficiently across architectures, experiments using the shallow water equation solver code were conducted on Piz Daint. First, we generated code versions with and without blocking for the Broadwell processor and the P100 GPU. An excerpt of results presented for different size of grids is shown in [29] and in Fig. 5a. The experiment was done with grid width of 200K. In the paper, we investigated the influence of the blocking factor on the application kernel further revealing that modest block sizes are leading to best performance (256 to 10 k for CPU and 2–10 k for GPU). On both architectures, wider grids run less efficiently as a result of an inefficient use of caches. The GPU implies additional overhead for the blocked version, requiring to run with a sufficiently large grid to benefit from it. This also shows the need to dynamically turn on/off blocking factors depending on the system capabilities.

4.4 Evaluating Vectorization and Memory Layout Optimization

As part of [28], we evaluated techniques to apply memory layout transformations. The experiments were done on two architectures, the Broadwell multi-core processors and the Aurora vector engine on the NEC test platform using the shallow water equation solver code.

Table 2 Performance of memory layout variants on CPU and the NEC vector architecture

Architecture	Scattered	Short distance	Contiguous
Broadwell	3 GFlops	13 GFlops	25 GFlops
NEC Aurora	80 GFlops	161 GFlops	322 GFlops

We used alternative layouts with different distances between data elements to investigate the impact of the data layout on the performance. The explored data alternatives were data accesses to

- contiguous unit stride arrays,
- interleaved data with constant short distance separating data elements, 4 bytes separating consecutive elements. This allowed to emulate array of structures (AoS) layouts,
- scattered data elements separated with long distances.

The results are listed in brief in Table 2. Using memory profilers, we found that the contiguous unit-stride code allowed to use the memory throughput efficiently on both architectures. In the emulated AoS codes, the efficiency dropped on both architectures. The worst measurements were taken for the scattered data elements.

Besides the impact of the optimization on the use of the memory bandwidth, vectorization is also affected by those alternatives. AVX2 was used for all kernels on Broadwell for the unit-stride code. Similarly, the vector units of the vector engine showed best results with this layout. Again the use of the vectorization was degraded with the emulated AoS, and was even worse with the scattered data elements.

4.5 Evaluating Inter-Kernel Optimization

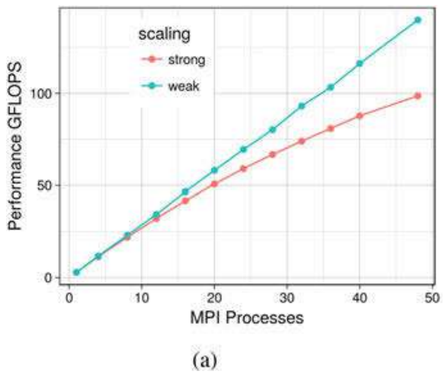
To explore the benefit of kernel merging, experiments were done using the shallow water equation solver code on the NEC test system (for vector engines) and Piz Daint (for GPUs and CPUs) [29]. The performance of regular and merged kernels are shown in Table 3. The code achieves near optimal use of the memory bandwidth already before the merge and actually decreases slightly after the merge. Exploiting the inter-kernel cache reuse allowed to reduce the data access in memory and increased the total performance of the application by 30–50%.

4.6 Scaling with Multiple-Node Runs

To demonstrate the ability of the DSL to support global icosahedral models, we carried out experiments using the two applications. Scalability experiments of

Table 3 Performance and efficiency of the kernel fusioning on all three architectures

Architecture	Theoretical Memory bandwidth (GB/s)	Before merge		With Inter-Kernel Merging	
		Measured memory throughput (GB/s) and peak	GFlops	Measured memory throughput (GB/s) and peak	GFflops
Broadwell	77	62 (80%)	24	60 (78%)	31 (+30%)
P100 GPU	500	380 (76%)	149	389 (78%)	221 (+48%)
NEC Aurora	1,200	961 (80%)	322	911 (76%)	453 (+40%)



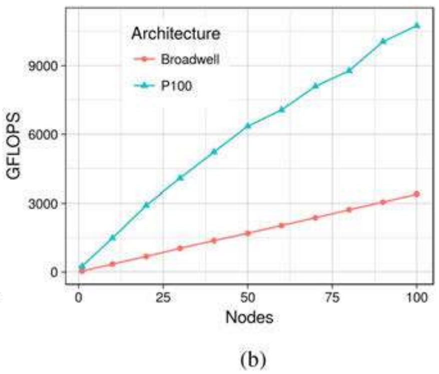


Fig. 6 Scalability of the two different models (measured on Mistral; P100 on Piz Daint). (**a**) Icosahedral code [30]. (**b**) Shallow water solver [31]

unstructured grid code were run on Mistral. Shallow water equation solver code experiments were run on Mistral for CPU tests, and on Piz Daint for GPU tests.

In the experiment using the unstructured grid, we use the global grid of the application and apply a three-dimensional Laplacian stencil. We varied the number of nodes that we use to run the code up to 48 nodes. The minimum number of the grid points we used is 1,048,576. We used this number of points for the strong-scale analysis. Weak scalability experiments were based on this number of points for each node. Figure 6a shows the results.

We could do further numbers of nodes, however, we found that the code was scaling with the tested cases and further experiments needed resources and time to get jobs to run on the test machine. For the measured cases, the weak scalability of the code is close to optimal. Thus, increasing the resolution of the grids and running the code on more nodes is achieved efficiently. This is an important point as higher resolution grids are essential for recent and future global simulations.

We also carried out experiments to scale the shallow water equation solver on both Broadwell multi-core processors and on the P100 GPUs at Piz Daint. We generated code for Broadwell experiments with OpenMP as well as MPI, and for the GPUs with OpenACC as well as MPI. Figure 6b shows the measured results of scaling the code. On the Broadwell processor, we used 36 OpenMP threads on each node.

While the performance of the GPU code scaled well, it loses quite some efficiency when running on two processes as the halo communication involves the host code. In general, the code that the tools generate for multiple nodes shows to be scalable, both on CPUs and GPUs. The DSL code does not include any details regarding single or multiple node configuration, so users do not need to care about multiple node parallelization. However, the parallelization can still be applied by the tool and the users can still control the parallelization process.

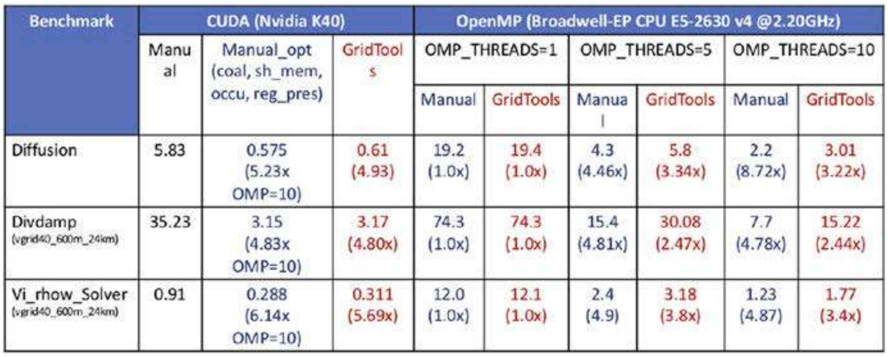
4.7 Exploring the Use of Alternative DSLs: GridTools

The GridTools framework is a set of libraries and utilities to develop performance-portable weather and climate applications. It is developed at The Swiss National Supercomputing Center [13]. To achieve the goal of performance portability, the user code is written in a generic form which is then optimized for a given architecture at compile time. The core of GridTools is the stencil composition module which implements a DSL embedded in C++ for stencils and stencil-like patterns. Further, GridTools provides modules for halo exchanges, boundary conditions, data management and bindings to C and Fortran. GridTools is successfully used to accelerate the dynamical core of the COSMO model with improved performance on CUDA-GPUs compared to the current official version, demonstrating production quality and feature-completeness of the library for models on lat-lon grids [48]. Although GridTools was developed for weather and climate applications it might be applicable for other domains with a focus on stencil-like computations.

In the context of the AIMES project, we evaluated the viability of using GridTools for the dynamical core of NICAM: namely NICAM-DC. Since NICAM-DC is written in Fortan, we first had to port the code to C++, which includes also changing the build systems. Figures 9 and 10 show simple example codes extracted from NICAM-DC and ported to GridTools, respectively. We ported the dynamical core using the following incremental approach. First, each operator was ported individually to GridTools, i.e. re-written from Fortran to C++. Next, we used a verification tool to assure that the same input to the C++ and Fortran version gives the same output. Next we move on to the following operator. Table 4 shows results from benchmarks extracted from NICAM-DC. It provides good speedup on GPU, and speed on CPU (in OpenMP) comparable to the hand-written version. Figure 7 shows results for running all operators of NICAM-DC on 10 nodes. It is worth mentioning that the most time consuming operator is more than 7x faster on GPU versus CPU.

GridTools demonstrates good GPU performance and acceptable CPU performance. The functionalities and features included in GridTools were enough to support the regular mesh code of NICAM-DC without friction (i.e. no custom features were required in GridTools to support the requirements of NICAM-DC). In addition, GridTools are transparent in the sense that no information about the platform is exposed to the end-user. On the other hand, following is a list of issues

Table 4 Execution time (seconds) of different benchmarks extracted from NICAM-DC. This includes the regular regions only, using 1 region, a single MPI rank for a 130 × 130 × 42 grid



Benchmark	CUDA (Nvidia K40)			OpenMP (Broadwell-EP CPU E5-2630 v4 @2.20GHz)					
	Manual	Manual_opt (coal, sh_mem, occu, reg_pres)	GridTools	OMP_THREADS=1		OMP_THREADS=5		OMP_THREADS=10	
				Manual	GridTools	Manual	GridTools	Manual	GridTools
Diffusion	5.83	0.575 (5.23x OMP=10)	0.61 (4.93)	19.2 (1.0x)	19.4 (1.0x)	4.3 (4.46x)	5.8 (3.34x)	2.2 (8.72x)	3.01 (3.22x)
Divdamp (vgrid40_600m_24km)	35.23	3.15 (4.83x OMP=10)	3.17 (4.80x)	74.3 (1.0x)	74.3 (1.0x)	15.4 (4.81x)	30.08 (2.47x)	7.7 (4.78x)	15.22 (2.44x)
Vi_rhow_Solver (vgrid40_600m_24km)	0.91	0.288 (6.14x OMP=10)	0.311 (5.69x)	12.0 (1.0x)	12.1 (1.0x)	2.4 (4.9)	3.18 (3.8x)	1.23 (4.87)	1.77 (3.4x)

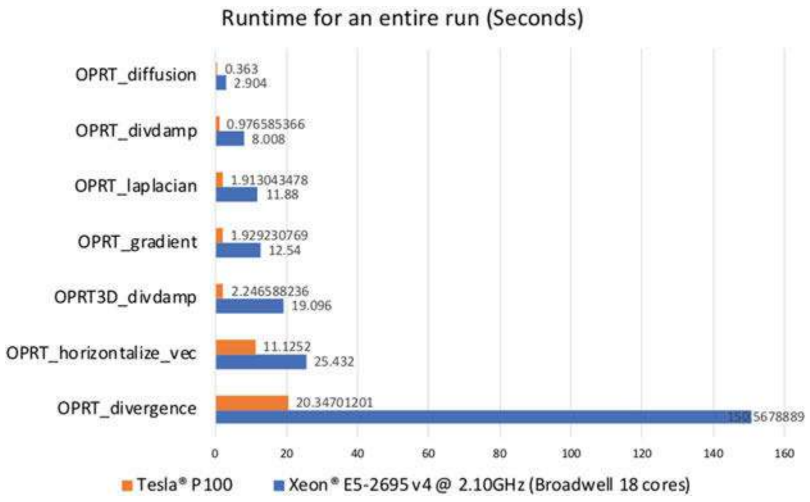


Fig. 7 NICAM-DC operators. Running on 10 nodes with one MPI rank per node. P100 is running GridTools generated kernels and Xeon uses the original Fortran code. Total grid is 32 × 32 × 10 using 40 vertical layers

that requires one's consideration when using GridTools: first, the requirement of rewriting the entire dynamical core in C++ is not a trivial task, especially since C++ templates make the code more convoluted, in comparison to Fortran. Second, while GridTools as a stencil framework does a good job for the dynamical core, separate solutions are required for the physics modules, the communicator module, and the non-regular compute modules (e.g. polar regions). Using different solutions inside the same code base typically increases the friction between code modules. Third, the interfacing between Fortran and C++ is non-trivial and can be troublesome considering that not all end-users are willing to change their build process.

Listing 9 Example of the diffusion operator extracted from NICAM-DC

```fortran
do d = XDIR, ZDIR
do j = jmin-1, jmax
do i = imin-1, imax
  vt(i,j,d) = (( + 2.0_RP * coef(i,j,d,1) &
                 - 1.0_RP * coef(i,j,d,2) &
                 - 1.0_RP * coef(i,j,d,3) ) * scl(i,j,d) &
            +  ( - 1.0_RP * coef(i,j,d,1) &
                 + 2.0_RP * coef(i,j,d,2) &
                 - 1.0_RP * coef(i,j,d,3) ) * scl(i+1,j,d)
                   ↪  &
            +  ( - 1.0_RP * coef(i,j,d,1) &
                 - 1.0_RP * coef(i,j,d,2) &
                 + 2.0_RP * coef(i,j,d,3) ) * scl(i,j+1,d)
                   ↪  &
           ) / 3.0_RP
enddo
enddo
enddo
```

Listing 10 Example of the diffusion operator ported to GridTools

```cpp
template <typename evaluation>
  GT_FUNCTION
  static void Do(evaluation const & eval, x_interval) {
    eval(vt{}) =  (( + 2.0 * eval(coef{})
                     - 1.0 * eval(coef{a+1})
                     - 1.0 * eval(coef{a+2}) ) * eval(scl
                       ↪ {})
                +  ( - 1.0 * eval(coef{})
                     + 2.0 * eval(coef{a+1})
                     - 1.0 * eval(coef{a+2}) ) * eval(scl{
                       ↪ i+1})
                +  ( - 1.0 * eval(coef{})
                     - 1.0 * eval(coef{a+1})
                     + 2.0 * eval(coef{a+2}) ) * eval(scl{
                       ↪ j+1})
                ) / 3.0;
}
```

5 Massive I/O and Compression

5.1 *The Scientific Compression Library (SCIL)*

The developed compression library SCIL [36] provides a framework to compress structured and unstructured data using the best available (lossless or lossy) compres-

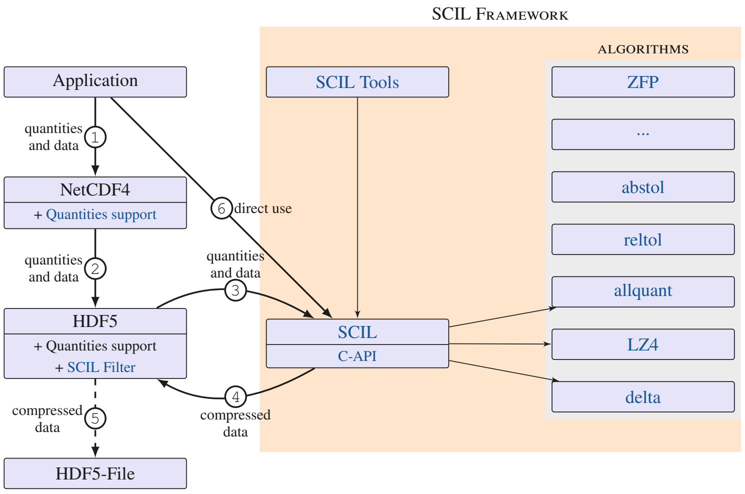


Fig. 8 SCIL compression path and components (extended)[36]

sion algorithms according to the definition of tolerable loss of accuracy and required performance. SCIL acts as a meta-compressor providing various backends such as algorithms like LZ4, ZFP, SZ but also integrates some alternative algorithms.

The data path of SCIL is illustrated in Fig. 8. An application can either use NetCDF4 [45],[2] HDF5 [19] or directly the C-interface of SCIL. Based on the defined quantities, the values and the characteristics of the data to compress, the appropriate compression algorithm is chosen. SCIL also comes with a library to generate various synthetic test patterns for compression studies, i.e., well-defined multi-dimensional data patterns of any size. Further tools are provided to plot, to add noise or to compress CSV and NetCDF3 files.

5.2 *Supported Quantities*

There are three types of quantities supported:

Accuracy Quantities define the tolerable error on lossy compression. When compressing the value v to $\hat{v}$ it bounds the residual error ($r = v - \hat{v}$):

- **absolute tolerance**: $v - \text{abstol} \leq \hat{v} \leq v + \text{abstol}$
- **relative tolerance**: $v/(1 + reltol) \leq \hat{v} \leq v \cdot (1 + reltol)$

[2]HDF5 and NetCDF4 are APIs and self-describing data formats for storing multi-dimensional data with user-relevant metadata.

- **relative error finest tolerance**: used together with relative tolerance; absolute tolerable error for small v's. If relfinest $> |v \cdot (1 \pm reltol)|$, then $v -$ relfinest $\leq \hat{v} \leq v +$ relfinest
- **significant digits**: number of significant decimal digits
- **significant bits**: number of significant digits in bits

SCIL must ensure that all the set accuracy quantities are honored regardless of the algorithm chosen, meaning that one can set, e.g., absolute and relative tolerance and the strictest of the criteria is satisfied.

Performance Quantities These quantities define the expected performance behavior for both compression and decompression (on the same system). The value can be defined according to: (1) absolute throughput in MiB or GiB; or (2) relative to network or storage speed. It is considered to be the expected performance for SCIL but it may not be as strictly handled as the qualities—there may be some cases in which performance is lower. Thus, SCIL must estimate the compression rates for the data.

Supplementary Quantities An orthogonal quantity that can be set is the so called *fill value*, a value that scientists use to mark special data points. This value must be preserved accurately and usually is a specific high or low value that may disturb a smooth compression algorithm.

5.3 Compression Chain

Internally, SCIL creates a compression chain which can involve several compression algorithms as illustrated in Fig. 9. Based on the basic datatype that is supplied, the initial stage of the chain is entered. Algorithms may be preconditioners to optimize data layout for subsequent compression algorithms, converters from one data format to another, or, on the final stage, a lossless compressor. Floating-point data can be first mapped to integer data and then to a byte stream. Intermediate steps can be skipped.

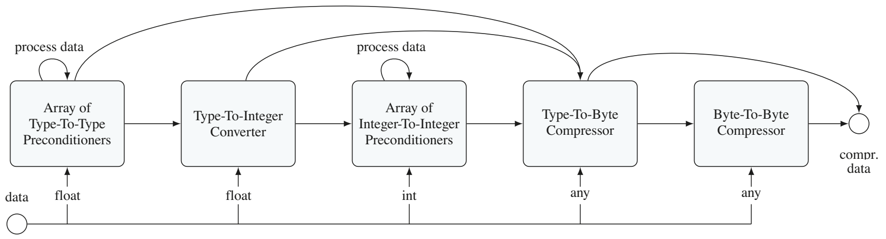

process data
Array of Type-To-Type Preconditioners
Type-To-Integer Converter
process data
Array of Integer-To-Integer Preconditioners
Type-To-Byte Compressor
Byte-To-Byte Compressor
compr. data
data
float
float
int
any
any

Fig. 9 SCIL compression chain. The data path depends on the input data type [36]

5.4 Algorithms

SCIL comes with additional algorithms that are derived to support one or multiple accuracy quantities set by the user. For example, the algorithms Abstol (for absolute tolerance), Sigbits (for significant bits/digits), and Allquant. These algorithms aim to pack the number of required bits as tightly as possible into the data buffer but operate on each value independently. While Abstol and Sigbits just consider one quantity, Allquant considers all quantities together and chooses the required operation for a data point depending on the highest precision needed. We also consider these algorithms to be useful baselines when comparing any other algorithm. ZFP and SZ, for example, work on one quantity, too.

During the project, we explored the implementation for the automatic algorithm selection but only integrated a trivial scheme for the following reasons: if only a single quantity is set, we found out that the optimal parameter depends on many factors (features); the resulting optimal choice is embedded in a multi-dimensional space—this made it infeasible to identify the optimal algorithm. Once more than a single quantity is set, only one of the newly integrated algorithms can perform the compression, which eliminates any choice. As the decoupling of SCIL enables to integrate algorithms in the future, we hope that more algorithms will be developed that can then benefit from implementing the automatic selection.

6 Evaluation of the Compression Library SCIL

We evaluated the performance and compression ratio of SCIL against several scientific data sets and the synthetic test patterns generated by SCIL itself [36].

6.1 Single Core Performance

In the following, an excerpt of the experiments conducted with SCIL on a single core is shown. These results help to understand the performance behavior of compression algorithms and their general characteristics.

Four data sets were used each with precision floating-point data (32 bit): (1) the data created with the SCIL pattern library (10 data sets each with different random seed numbers). The synthetic data has the dimensionality of ($300 \times 300 \times 100 =$ 36 MB); (2) the output of the ECHAM atmospheric model [46] which stored 123 different scientific variables for a single timestep as NetCDF; (3) the output of the hurricane Isabel model which stored 633 variables for a single timestep as binary;[3]

Table 5 Harmonic mean compression performance for different scientific data sets

	Algorithm	Ratio	Compr. MiB/s	Decomp. MiB/s
(*a*) 1% *absolute tolerance*				
NICAM	abstol	0.206	499	683
	abstol,lz4	0.015	458	643
	sz	0.008	122	313
	zfp-abstol	0.129	302	503
ECHAM	abstol	0.190	260	456
	abstol,lz4	0.062	196	400
	sz	0.078	81	169
	zfp-abstol	0.239	185	301
Isabel	abstol	0.190	352	403
	abstol,lz4	0.029	279	356
	sz	0.016	70	187
	zfp-abstol	0.039	239	428
Random	abstol	0.190	365	382
	abstol,lz4	0.194	356	382
	sz	0.242	54	125
	zfp-abstol	0.355	145	241
(*b*) 9 *bits precision*				
NICAM	sigbits	0.439	257	414
	sigbits,lz4	0.216	182	341
	zfp-precision	0.302	126	182
ECHAM	sigbits	0.448	462	615
	sigbits,lz4	0.228	227	479
	zfp-precision	0.299	155	252
Isabel	sigbits	0.467	301	506
	sigbits,lz4	0.329	197	366
	zfp-precision	0.202	133	281
Random	sigbits	0.346	358	511
	sigbits,lz4	0.348	346	459
	zfp-precision	0.252	151	251

(4) the output of the NICAM Icosahedral Global Atmospheric model which stored 83 variables as NetCDF.

The characteristics of the scientific data varies and so does data locality within the data sets. For example, in the Isabel data many variables are between 0 and 0.02, many between −80 and +80 and some are between −5000 and 3000.

We set only one quantity to allow using ZFP and SZ for comparison. Table 5 shows the harmonic mean compression ratio[4] for setting an absolute error of 1% of

[3]http://vis.computer.org/vis2004contest/data.html.

[4]The ratio is the resulting file size divided by the original file size.

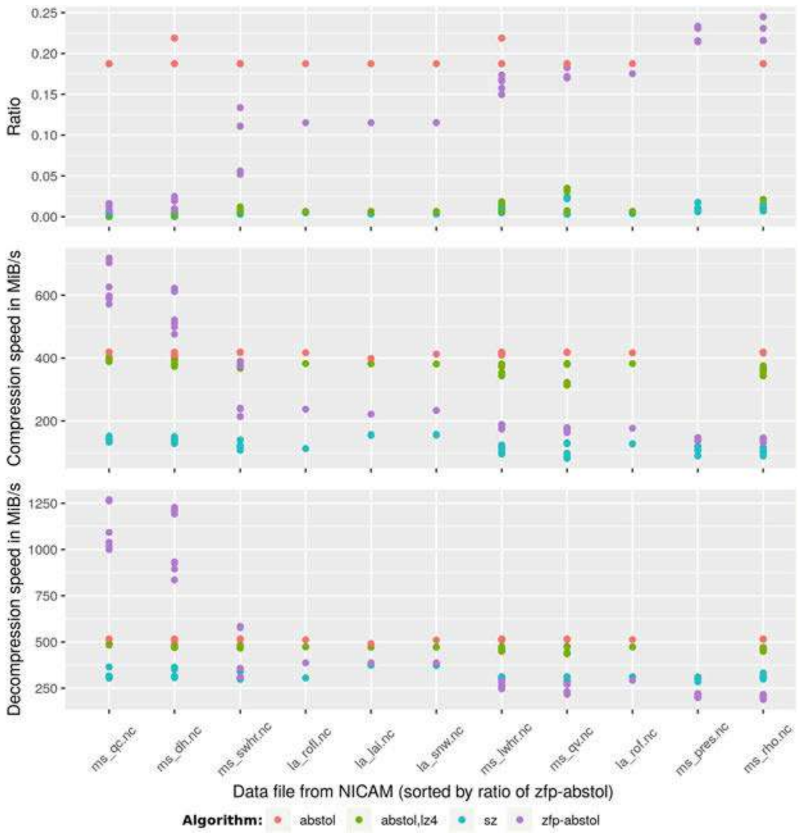


Fig. 10 Compressing various climate variables with absolute tolerance 1%

the maximum value or setting precision to 9 bit accuracy. The harmonic mean corresponds to the total reduction and performance when compressing/decompressing all the data.

The results for compressing 11 variables of the whole NICAM model via the NetCDF API are shown in Fig. 10. The x-axis represents the different data files, as each file consists of several chunks, a point in the y-axis represents one chunk. It can be observed that generally the SZ algorithm yields the best compression ratio but Abstol+LZ4 yields the second best ratio providing much better and predictable compression and decompression speeds.

Note that for some individual variables, one algorithm may supersede another in terms of ratio. As expected there are cases in which one algorithm is outperforming the other algorithms in terms of compression ratio which justifies the need for a metacompressor like SCIL that can make smart choices on behalf of the users. Some algorithms perform generally better than others in terms of performance. Since in our use cases, users define the tolerable error, we did not investigate metrics that compare the precision for a fixed compression ratio (e.g., the signal to noise ratio).

While a performance of 200 MB/s may look insufficient for a single core, with 24 cores per node a good speed per node can be achieved that is still beneficial for medium large runs on shared storage. For instance, consider a storage system that can achieve 500 GB/s. Considering that one node with typical Infiniband configuration can transfer at least 5 GB/s, 100 client nodes saturate the storage. By compressing 5:1 (or ratio of 0.2), virtually, the storage could achieve a peak performance of 2500 GB/s, and, thus, can serve up to 500 client nodes with (theoretical) maximum performance.

Trading of storage capacity vs. space is an elementary issue to optimize bigger workflows. By separating the concerns between the necessary data quality as defined by scientists and compression library, site-specific policies could be employed that depend also on the available hardware.

6.2 Compression in HDF5 and NetCDF

We tested the compression library SCIL using the icosahedral grid code. The code can use NetCDF to output the generated data periodically. In this experiment, a high-resolution grid with 268 million grid cells (single precision floating point) in the horizontal times 64 vertical levels was used and executed on the supercomputer Mistral. The code was run on 128 nodes, with one MPI process per node. It wrote one field to the output file in one timestep. The field values range between -10 to $+55$ which is important for understanding the impact of the compression.

The experiments varied the basic compression algorithms and parameters provided by SCIL. Compression is done with the algorithms

- memcopy: does not do any real compression, but allows to measure the overhead of the usage of enabling HDF5 compression and SCIL.
- lz4: the well-known compression algorithm. It is unable to compress floating-point data but slows down the execution.
- abstol,lz4: processes data elements based on the absolute value of each point, we control the tolerance by the parameter $absolute\_tolerance$, after quantization an LZ4 compression step is added.
- sigbits,lz4: processes data elements based on a percentage of the value being processed, we control the tolerance by the parameter $relative\_tolerance\_percent$, after quantization an LZ4 compression step is added.

The results of this selection of compression methods is shown in Table 6, it shows the write time in seconds and resulting data size in GB, a virtual throughput relative to the uncompressed file size, and the speedup. Without compression the performance is quite poor: achieving only 432 MB/s on Mistral on 128 nodes, while an optimal benchmark can achieve 100 GB/s. The HDF5 compression is not yet optimally parallelized and requires certain collective operations to update

Table 6 Compression results of 128 processes on Mistral

Compression method	Parameter	Write time in s	Data size in GB	Throughput* in MB/s	Speedup
No-compression		165.3	71.4	432	1.0
memcopy		570.1	71.4	125	0.3
lz4		795.3	71.9	90	0.2
abstol,lz4	absolute_tolerance=1	12.8	2.7	5578	12.9
	absolute_tolerance=.1	72.6	2.8	983	2.3
sigbits,lz4	relative_tolerance_percent=1	12.9	2.9	5535	12.8
	relative_tolerance_percent=.1	18.3	3.2	3902	9.0

the metadata. Internally, HDF5 requires additional data buffers. This leads to extra overhead in the compression slowing down the I/O (see the memcopy and and LZ4 results which serve as baselines). By activating lossy compression and accepting an accuracy of 1% or 0.1%, the performance can be improved in this example up to 13x.

Remember that these results serve as feasibility study. One of our goals was to provide a NetCDF backwards compatible compression method not to optimize the compression data path inside HDF5. The SCIL library can be used directly by existing models avoiding the overhead and leading to the results as shown above.

7 Standardized Icosahedral Benchmarks

Our research on DSL and I/O is more practical. We started with real-world applications, namely three global atmospheric models developed by the participating countries. The global atmospheric model with the icosahedral grid system is one of the new generation global climate/weather models. The grid-point calculations, which are less computational intense than the spherical harmonics transformation, are used in the model. On the other hand, patterns of the data access in differential operators are more complicated than the traditional limited-area atmospheric model with the Cartesian grid system.

There are different implementations of the dynamical core on the icosahedral grid: direct vs. indirect memory access, staggered vs. co-located data distribution, and so on. The objective of standardization of benchmarks is to provide a variety of computational patterns of the icosahedral atmospheric models. The kernels are useful for evaluating the performance of new machines and new programming models. Not only for our studies but also for the existing/future DSL studies, this benchmark set provides various samples of the source code. The icosahedral grid system is an unstructured grid coordinate, and there are a lot of challenging issues about data decomposition, data layout, loop structures, cache blocking, threading, offloading the accelerators, and so on. By applying DSLs or frameworks to the kernels, the developers can try the detailed, practical evaluation of their software.

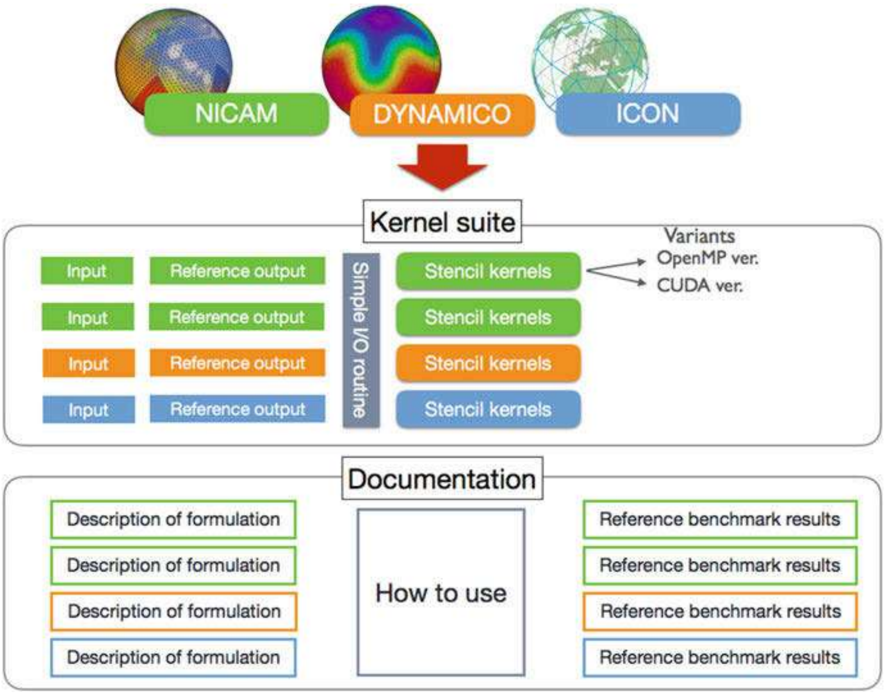


Fig. 11 Overview of IcoAtmosBenchmark v1

The benchmarks were used in the final evaluation of SCIL and they steered the DSL development by providing the relevant patterns.

7.1 IcoAtmosBenchmark v1: Kernels from Icosahedral Atmospheric Models

IcoAtmosBenchmark v1 is the package of kernels extracted from three Icosahedral Global Atmospheric models, NICAM, ICON, DYNAMICO. As shown in Fig. 11, we prepared input data and reference output data for easy validation of results. We also arranged documentation about the kernels. The package is available online.[5]

[5]https://aimes-project.github.io/IcoAtmosBenchmark_v1/.

7.1.1 Documentation

An excerpt to the NICAM kernel serves as an example. The icosahedral grid on the sphere of NICAM is the unstructured grid system. In NICAM code, the complex topology of the grid system is separated into the structured and unstructured part. The grids are decomposed into tiles, and one or more tiles are allocated to each MPI process. The horizontal grids in the tile are kept in a 2-dimensional structure. On the other hand, a topology of the network of the tiles is complex. We selected and extracted 6+1 computational kernels from the dynamical core of NICAM, as samples of the stencil calculation on the structured grid system. We also extracted a communication kernel, as a sample of halo exchange in the complex node topology. The features of each kernel are documented on the GitHub page.

All kernels are single subroutines and almost the same as the source codes in the original model, except for the ICON kernels. They are put into the wrapper for the kernel program. Values of input variables in the argument list of the kernel subroutine are stored as a data file, just before the call in the execution of the original model. They are read and given to the subroutine in the kernel program. Similarly, the values of output variables in the argument list are stored, just after the call in execution. They are read and compared to the actual output values of kernel subroutine. The differences are written to the standard output for validation. For easy handling of the input/reference data by both the Fortran program and C program, we prepared an I/O routine written in C.

We provided a user manual, which contains the brief introduction of each model, the description of each kernel, usage of kernel programs, and sample results. This information is helpful for users of this kernel suite in the future.

8 Summary and Conclusion

The numerical simulation of climate and weather is demanding for computational and I/O resources. Within the AIMES project, we addressed those challenges and researched approaches that foster the separation of concerns. This idea unites our approaches for the DSL and the compression library. While a higher level of abstraction can improve the productivity for scientists, most importantly the decoupling of requirements from the implementation allows scientific programmers to develop and improve architecture-specific optimizations.

Promising candidates for DSLs have been explored and with GGDML an alternative has been developed that covers the most relevant formulations of the three icosahedral models: DYNAMICO, ICON, and NICAM. The DSL and toolchain we developed integrates into existing code bases and suits for incremental reformulation of the code. We estimated the benefit for code reduction and demonstrated several optimizations for CPU, GPU, and vector architectures. Our DSL allows to reduce code to around 30% of the LOC in comparison to code written with GPL code.

With the semantics of GGDML, we could achieve near optimal use of memory hierarchies and memory throughput which is critical for the family of computations in hand. Our experiments show running codes with around 80% of achievable memory throughput on different archiectures. Furthermore, we could scale models to multiple nodes, which is essential for exascale computing, using the same code that is used for a single node. The separation of concerns in our approach allowed us to keep models code separate of underlying hardware changes. The single GGDML source code is used to generate code for the different architectures and on single vs. multiple nodes.

To address the I/O challenge, we developed the library SCIL, a metacompressor supporting various lossy and lossless algorithms. It allows users to specify various quantities for the tolerable error and expected performance, and allows the library to chose a suitable algorithm. SCIL is a stand-alone library but also integrates into NetCDF and HDF5 allowing users to explore the benefits of using alternative compression algorithms with their existing middleware. We evaluated the performance and compression ratio for various scientific data sets and on moderate scale. The results show that the choice of the best algorithm depends on the data and performance expectation which, in turn, motivates the need for the decoupling of quantities from the selection of the algorithm. A blocker for applying parallel large-scale compression in existing NetCDF workflows is the performance limitation of the current HDF5 stack.

Finally, benchmarks and mini-applications were created that represent the key features of the icosahedral applications.

Beside the achieved research in the AIMES project, the work done opens the door for further research in the software engineering of climate and weather prediction models. The performance portability, where we used the same code to run on different architectures and machines, including single and multiple nodes, shows that techniques are viable to continue the research in this direction. The code reduction offered by DSLs promises to save millions in development cost that can be used to contribute to the DSL development. During the runtime of the project, it became apparent that the concurrently developed solutions GridTools and PSYclone that also provide such features are preferred by most scientists as they are developed by a bigger community and supported by national centers. We still believe that the developed light-weight DSL solution provides more flexibility particularly for smaller-sized models and can be maintained as part of the development of the models itself. It also can be used in other contexts providing domain-specific templates to arbitrary codes.

In the future, we aim to extend the DSL semantics to also address I/O relevant specifications. This would allow to unify the effort towards optimal storage and computation.

References

1. Adams, S.V., Ford, R.W., Hambley, M., Hobson, J., Kavčič, I., Maynard, C., Melvin, T., Müller, E.H., Mullerworth, S., Porter, A., et al.: LFRic: Meeting the challenges of scalability and performance portability in Weather and Climate models. J. Parallel Distrib. Comput. **132**, 383–396 (2019)
2. Alforov, Y., Novikova, A., Kuhn, M., Kunkel, J., Ludwig, T.: Towards green scientific data compression through high-level I/O interfaces. In: 30th International Symposium on Computer Architecture and High Performance Computing, pp. 209–216. Springer, Berlin (2019). https://doi.org/10.1109/CAHPC.2018.8645921
3. Baker, A.H., Xu, H., Dennis, J.M., Levy, M.N., Nychka, D., Mickelson, S.A., Edwards, J., Vertenstein, M., Wegener, A.: A methodology for evaluating the impact of data compression on climate simulation data. In: Proceedings of the 23rd international symposium on High-performance parallel and distributed computing, pp. 203–214. ACM, New York (2014)
4. Baker, A.H., Hammerling, D.M., Mickelson, S.A., Xu, H., Stolpe, M.B., Naveau, P., Sanderson, B., Ebert-Uphoff, I., Samarasinghe, S., De Simone, F., Gencarelli, C.N., Dennis, J.M., Kay, J.E., Lindstrom, P.: Evaluating lossy data compression on climate simulation data within a large ensemble. Geosci. Model Dev. **9**, 4381–4403 (2016). https://doi.org/10.5194/gmd-9-4381-2016
5. Baron, T.J., Khlopkov, K., Pretorius, T., Balzani, D., Brands, D., Schröder, J.: Modeling of microstructure evolution with dynamic recrystallization in finite element simulations of martensitic steel. Steel Res. Int. **87**(1), 37–45 (2016)
6. Barry, B., et al.: Software Engineering Economics, vol. 197. Prentice-Hall, New York (1981)
7. Bastian, P., Engwer, C., Fahlke, J., Geveler, M., Göddeke, D., Iliev, O., Ippisch, O., Milk, R., Mohring, J., Müthing, S., et al.: Advances concerning multiscale methods and uncertainty quantification in EXA-DUNE. In: Software for Exascale Computing-SPPEXA 2013-2015, pp. 25–43. Springer, Berlin (2016)
8. Bauer, P., Thorpe, A., Brunet, G.: The quiet revolution of numerical weather prediction. Nature **525**(7567), 47 (2015)
9. Bauer, S., Drzisga, D., Mohr, M., Rüde, U., Waluga, C., Wohlmuth, B.: A stencil scaling approach for accelerating matrix-free finite element implementations. SIAM J. Sci. Comput. **40**(6), C748–C778 (2018)
10. Bauer, S., Huber, M., Ghelichkhan, S., Mohr, M., Rüde, U., Wohlmuth, B.: Large-scale simulation of mantle convection based on a new matrix-free approach. J. Comput. Sci. **31**, 60–76 (2019)
11. Bicer, T., Agrawal, G.: A compression framework for multidimensional scientific datasets. In: 2013 IEEE 27th International Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW), pp. 2250–2253 (2013). https://doi.org/10.1109/IPDPSW.2013.186
12. CSCS Claw. http://www.xcalablemp.org/download/workshop/4th/Valentin.pdf
13. CSCS GridTools. https://github.com/GridTools/gridtools
14. DeVito, Z., Joubert, N., Palacios, F., Oakley, S., Medina, M., Barrientos, M., Elsen, E., Ham, F., Aiken, A., Duraisamy, K., et al.: Liszt: a domain specific language for building portable mesh-based pde solvers. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, p. 9. ACM, New York (2011)
15. Di, S., Cappello, F.: Fast Error-bounded Lossy HPC data compression with SZ. In: 2016 IEEE International Parallel and Distributed Processing Symposium, pp. 730–739. IEEE, Piscataway (2016). https://doi.org/10.1109/IPDPS.2016.11
16. Dolbeau, R., Bihan, S., Bodin, F.: Hmpp: a hybrid multi-core parallel programming environment. In: Workshop on General Purpose Processing on Graphics Processing Units (GPGPU 2007), vol. 28 (2007)
17. Dubos, T., Dubey, S., Tort, M., Mittal, R., Meurdesoif, Y., Hourdin, F.: Dynamico, an icosahedral hydrostatic dynamical core designed for consistency and versatility. Geosci. Model Dev. Discuss. **8**(2) (2015)

18. Faghih-Naini, S., Kuckuk, S., Aizinger, V., Zint, D., Grosso, R., Köstler, H.: Towards whole program generation of quadrature-free discontinuous Galerkin methods for the shallow water equations. arXiv preprint:1904.08684 (2019)
19. Folk, M., Cheng, A., Yates, K.: HDF5: a file format and I/O library for high performance computing applications. In: Proceedings of Supercomputing, vol. 99, pp. 5–33 (1999)
20. Gerbes, A., Kunkel, J., Jumah, N.: Intelligent Selection of Compiler Options to Optimize Compile Time and Performance (2017). http://llvm.org/devmtg/2017-03//2017/02/20/accepted-sessions.html#42
21. Gerbes, A., Jumah, N., Kunkel, J.: Automatic Profiling for Climate Modeling (2018). http://llvm.org/devmtg/2017-03//2017/02/20/accepted-sessions.html#42
22. Gomez, L.A.B., Cappello, F.: Improving floating point compression through binary masks. In: 2013 IEEE International Conference on Big Data (2013). https://doi.org/10.1109/BigData.2013.6691591
23. Gysi, T., Fuhrer, O., Osuna, C., Cumming, B., Schulthess, T.: Stella: A domain-specific embedded language for stencil codes on structured grids. In: EGU General Assembly Conference Abstracts, vol. 16 (2014)
24. Heene, M., Hinojosa, A.P., Obersteiner, M., Bungartz, H.J., Pflüger, D.: EXAHD: an Exa-Scalable two-level sparse grid approach for higher-dimensional problems in plasma physics and beyond. In: High Performance Computing in Science and Engineering'17, pp. 513–529. Springer, Berlin (2018)
25. Hübbe, N., Kunkel, J.: Reducing the HPC-Datastorage footprint with MAFISC—Multidimensional Adaptive Filtering Improved Scientific data Compression. In: Computer Science—Research and Development, pp. 231–239 (2013). https://doi.org/10.1007/s00450-012-0222-4
26. Hübbe, N., Wegener, A., Kunkel, J., Ling, Y., Ludwig, T.: Evaluating lossy compression on climate data. In: Kunkel, J.M., Ludwig, T., Meuer, H.W. (eds.) Supercomputing, no. 7905 in Lecture Notes in Computer Science, pp. 343–356. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-38750-0_26
27. Iverson, J., Kamath, C., Karypis, G.: Fast and effective lossy compression algorithms for scientific datasets. In: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 7484, pp. 843–856. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-32820-6_83
28. Jum'ah, N., Kunkel, J.: Automatic vectorization of stencil codes with the GGDML language extensions. In: Workshop on Programming Models for SIMD/Vector Processing (WPMVP'19), February 16, 2019, Washington, DC, USA, WPMVP, pp. 1–7. PPoPP 2019. ACM, New York (2019).https://doi.org/10.1145/3303117.3306160. https://ppopp19.sigplan.org/home/WPMVP-2019
29. Jumah, N., Kunkel, J.: Optimizing memory bandwidth efficiency with user-preferred kernel merge. In: Lecture Notes in Computer Science. Springer, Berlin (2019)
30. Jum'ah, N., Kunkel, J.: Performance portability of earth system models with user-controlled GGDML code translation. In: Yokota, R., Weiland, M., Shalf, J., Alam, S. (eds.) High Performance Computing: ISC High Performance 2018 International Workshops, Frankfurt/Main, Germany, June 28, 2018, Revised Selected Papers, no. 11203. Lecture Notes in Computer Science, pp. 693–710. ISC Team, Springer, Berlin (2019). https://doi.org/10.1007/978-3-030-02465-9_50
31. Jumah, N., Kunkel, J.: Scalable Parallelization of Stencils using MODA. In: High Performance Computing: ISC High Performance 2019 International Workshops, Frankfurt/Main, Germany, June 20, 2019, Revised Selected Papers. Lecture Notes in Computer Science. Springer, Berlin (2019)
32. Jumah, N., Kunkel, J., Zängl, G., Yashiro, H., Dubos, T., Meurdesoif, Y.: GGDML: Icosahedral models language extensions. J. Comput. Sci. Technol. Updates **4**(1), 1–10 (2017). https://doi.org/10.15379/2410-2938.2017.04.01.01.http://www.cosmosscholars.com/images/JCSTU_V4N1/JCSTU-V4N1A1-Jumah.pdf

33. Kronawitter, S., Kuckuk, S., Köstler, H., Lengauer, C.: Automatic data layout transformations in the ExaStencils code generator. Mod. Phys. Lett. A **28**(03), 1850009 (2018)
34. Kunkel, J.: Analyzing data properties using statistical sampling techniques—illustrated on scientific file formats and compression features. In: Taufer, M., Mohr, B., Kunkel, J. (eds.) High Performance Computing: ISC High Performance 2016 International Workshops, ExaComm, E-MuCoCoS, HPC-IODC, IXPUG, IWOPH, P3MA, VHPC, WOPSSS, no. 9945. Lecture Notes in Computer Science, pp. 130–141. Springer, Berlin (2016). https://doi.org/10.1007/978-3-319-46079-6_10
35. Kunkel, J.: SFS: A Tool for Large Scale Analysis of Compression Characteristics. Technical Report 4, Deutsches Klimarechenzentrum GmbH, Bundesstraße 45a, D-20146 Hamburg (2017)
36. Kunkel, J., Novikova, A., Betke, E.: Towards Decoupling the Selection of Compression Algorithms from Quality Constraints—an Investigation of Lossy Compression Efficiency. Supercomputing Front. Innov. **4**(4), 17–33 (2017). https://doi.org/10.14529/jsfi170402. http://superfri.org/superfri/article/view/149
37. Kunkel, J., Novikova, A., Betke, E., Schaare, A.: Toward decoupling the selection of compression algorithms from quality constraints. In: Kunkel, J., Yokota, R., Taufer, M., Shalf, J. (eds.) High Performance Computing: ISC High Performance 2017 International Workshops, DRBSD, ExaComm, HCPM, HPC-IODC, IWOPH, IXPUG, P3MA, VHPC, Visualization at Scale, WOPSSS. No. 10524 in Lecture Notes in Computer Science, pp. 1–12. Springer, Berlin (2017). https://doi.org/10.1007/978-3-319-67630-2_1
38. Lakshminarasimhan, S., Shah, N., Ethier, S., Klasky, S., Latham, R., Ross, R., Samatova, N.: Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data. In: European Conference on Parallel and Distributed Computing (Euro-Par), Bordeaux, France (2011). https://doi.org/10.1007/978-3-642-23400-2_34
39. Laney, D., Langer, S., Weber, C., Lindstrom, P., Wegener, A.: Assessing the effects of data compression in simulations using physically motivated metrics. Super Comput. (2013). https://doi.org/10.3233/SPR-140386
40. Lindstrom, P.: Fixed-rate compressed floating-point arrays. In: IEEE Transactions on Visualization and Computer Graphics 2012 (2014). https://doi.org/10.1109/BigData.2013.6691591
41. Lindstrom, P., Isenburg, M.: Fast and efficient compression of floating-point data. IEEE Trans. Vis. Comput. Graph. **12**(5), 1245–1250 (2006). https://doi.org/10.1109/TVCG.2006.143
42. Maruyama, N., Sato, K., Nomura, T., Matsuoka, S.: Physis: an implicitly parallel programming model for stencil computations on large-scale GPU-accelerated supercomputers. In: 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1–12. IEEE, Piscataway (2011)
43. Müller, M., Aoki, T.: Hybrid fortran: high productivity GPU porting framework applied to Japanese weather prediction model. arXiv preprint: 1710.08616 (2017)
44. Rathgeber, F., Ham, D.A., Mitchell, L., Lange, M., Luporini, F., McRae, A.T., Bercea, G.T., Markall, G.R., Kelly, P.H.: Firedrake: automating the finite element method by composing abstractions. ACM Trans. Math. Softw. (TOMS) **43**(3), 24 (2017)
45. Rew, R., Davis, G.: NetCDF: an interface for scientific data access. IEEE Comput. Graph. Appl. **10**(4), 76–82 (1990)
46. Roeckner, E., Bäuml, G., Bonaventura, L., Brokopf, R., Esch, M., Giorgetta, M., Hagemann, S., Kirchner, I., Kornblueh, L., Manzini, E., et al.: The Atmospheric General Circulation Model ECHAM 5. Part I: Model Description. Report/Max-Planck-Institut für Meteorologie, p. 349, (2003). http://hdl.handle.net/11858/00-001M-0000-0012-0144-5
47. Satoh, M., Tomita, H., Yashiro, H., Miura, H., Kodama, C., Seiki, T., Noda, A.T., Yamada, Y., Goto, D., Sawada, M., et al.: The non-hydrostatic icosahedral atmospheric model: description and development. Prog. Earth. Planet. Sci. **1**(1), 18 (2014)
48. Thaler, F., Moosbrugger, S., Osuna, C., Bianco, M., Vogt, H., Afanasyev, A., Mosimann, L., Fuhrer, O., Schulthess, T.C., Hoefler, T.: Porting the cosmo weather model to manycore CPUS. In: Proceedings of the Platform for Advanced Scientific Computing Conference, PASC '19, pp. 13:1–13:11 (2019)

49. Tietz, J.: Vector Folding for Icosahedral Earth System Models (2018). https://wr.informatik.uni-hamburg.de/_media/research:theses:jonas_tietz_vector_folding_for_icosahedral_earth_system_models.pdf
50. Torres, R., Linardakis, L., Kunkel, T.J., Ludwig, T.: Icon dsl: a domain-specific language for climate modeling. In: International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, Colo (2013). http://sc13.supercomputing.org/sites/default/files/WorkshopsArchive/track139.html
51. Unat, D., Cai, X., Baden, S.B.: Mint: realizing cuda performance in 3D stencil methods with annotated c. In: Proceedings of the International Conference on Supercomputing, pp. 214–224. ACM, New York (2011)
52. van Engelen, R.A.: Atmol: a domain-specific language for atmospheric modeling. CIT. J. Comput. Inf. Technol. **9**(4), 289–303 (2001)
53. Velten, K.: Mathematical Modeling and Simulation: Introduction for Scientists and Engineers. Wiley, New York (2009)
54. Zängl, G., Reinert, D., Rípodas, P., Baldauf, M.: The icon (icosahedral non-hydrostatic) modelling framework of dwd and mpi-m: description of the non-hydrostatic dynamical core. Q. J. R. Meteorol. Soc. **141**(687), 563–579 (2015)
55. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Inf. Theory **23**(3), 337–343 (1977). https://doi.org/10.1109/TIT.1977.1055714

DASH: Distributed Data Structures and Parallel Algorithms in a Global Address Space

Karl Fürlinger, José Gracia, Andreas Knüpfer, Tobias Fuchs, Denis Hünich, Pascal Jungblut, Roger Kowalewski, and Joseph Schuchart

Abstract DASH is a new programming approach offering distributed data structures and parallel algorithms in the form of a C++ template library. This article describes recent developments in the context of DASH concerning the ability to execute tasks with remote dependencies, the exploitation of dynamic hardware locality, smart data structures, and advanced algorithms. We also present a performance and productivity study where we compare DASH with a set of established parallel programming models.

1 Introduction

DASH is a parallel programming approach that realizes the PGAS (partitioned global address space) model and is implemented as a C++ template library. DASH tries to reconcile the productivity advantages of shared memory parallel programming with the physical realities of distributed memory hardware. To achieve this goal, DASH provides the abstraction of globally accessible memory that spans multiple interconnected nodes. For performance reasons, this global memory is partitioned and data locality is not hidden but explicitly exploitable by the application developer.

DASH is realized as a C++ template library, obviating the need for a custom language and compiler. By relying on modern C++ abstraction and implementation techniques, a productive programming environment can be built solely based on

K. Fürlinger (✉) · T. Fuchs · P. Jungblut · R. Kowalewski
Ludwig-Maximilians-Universität München, Munich, Germany
e-mail: Karl.Fuerlinger@ifi.lmu.de

J. Gracia · J. Schuchart
University of Stuttgart, Stuttgart, Germany

A. Knüpfer · D. Hünich
TU Dresden, Dresden, Germany

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_6

standard components. For many application developers in HPC and in general, a big part of the appeal of the C++ programming language stems from the availability of high performance generic data structures and algorithms in the C++ standard template library (STL).

DASH can be seen as a generalization of concepts found in the STL to the distributed memory case and efforts have been made to keep DASH compatible with components of the STL. In many cases it is thus possible to mix and match algorithms and data structures freely between DASH and the STL.

DASH is developed in the context of the SPPEXA priority programme for Exascale computing since 2013. In this paper we give an overview of DASH and report on activities within the project focusing on the second half of the funding period. We first give an overview of the DASH Runtime System (DART) in Sect. 2, focusing on features related to task execution with global dependencies and dynamic hardware topology discovery. In Sect. 3 we describe two components of the DASH C++ template library, a smart data structure that offers support for productive development of stencil codes and an efficient implementation of parallel sorting. In Sect. 4 we provide an evaluation of DASH testing the feasibility of our approach. We provide an outlook on future developments in Sect. 5.

2 The DASH Runtime System

The DASH Runtime System (DART) is implemented in C and provides an abstraction layer on top of distributed computing hardware and one-sided communication substrates. The main functionality provided by DART is memory allocation and addressing as well as communication in a global address space. In DASH parlance the individual participants in an application are called *units* mapped to MPI processes in the MPI-3 remote memory access based implementation of DART.

Early versions of DASH/DART focused on data distribution and access and offered no explicit compute model. This has changed with the support for tasks in DASH and DART. We start with a discussion of these new features, followed by a description of efforts to tackle increasing hardware complexity in Sect. 2.2.

2.1 Tasks with Global Dependencies

The benefit of decoupled transfer and synchronization in the PGAS programming model promises to provide improved scalability and better exploit hardware capabilities. However, proper synchronization of local and global memory accesses is essential for the development of correct applications. So far, the synchronization constructs in DASH were limited to collective synchronization using barriers and reduction operations as well as an implementation of the MCS lock. Using atomic RMA operations, users could also create custom synchronization schemes using point-to-point signaling, i.e., by setting a flag at the target after completion of a transfer. While this approach might work for simple examples, it hardly scales to more complex examples where reads and writes from multiple processes need to be synchronized.

The need for a more fine-grained way of synchronization that allows to create more complex synchronization patterns was thus imminent. The data-centric programming model of DASH with the distributed data structure at its core lead motivated us to create a synchronization that centers around these global data structures, i.e., which is data-centric itself. At the same time, the essential property of PGAS needed to be preserved: the synchronization had to remain decoupled from data transfers, thus not forcing users to rely solely on the new synchronization mechanism for data transfers.

At the same time, the rise of task-based programming models inspired us to investigate the use of tasks as a synchronization vehicle, i.e., by encapsulating local and global memory accesses into tasks that are synchronized using a data-centric approach. Examples of widely known data-centric task-based synchronization models are OpenMPI with its task data dependencies, OmpSs, and PaRSEC. While PaRSEC uses data dependencies to express both synchronization and actual data flow between tasks, OpenMP and OmpSs use data dependencies solely for synchronization without affecting data movement. In contrast to PaRSEC, however, OpenMP and OmpSs only support shared memory parallelization.

A different approach has been taken by HPX, which facilitates synchronization through the use of future/promise pairs, which form a channel between two or more tasks and are a concept that has been well established in the C++ community. However, this synchronization concept with it's inherent communication channel hardly fits into the concept of a PGAS abstraction built around data structures in the global memory space. Moreover, DASH provides a locality-aware programming, in which processes know their location in the global address and can diverge their control accordingly, whereas HPX is a locality-agnostic programming model.

We thus decided to focus our research efforts on distributed data dependencies, extending the shared memory capabilities of task data dependencies into the global memory space while keeping synchronization and data transfer decoupled.

2.1.1 Distributed Data Dependencies

Tasks in DASH are created using the `async` function call and passing it an action that will be executed by a worker thread at a later point in time. Additionally, the `async` function accepts an arbitrary number of dependency definitions of the form `in(memory_location)` and `out(memory_location)` to define input and output dependencies, respectively. In the DASH tasking model, each unit discovers it's local task graph by creating tasks operating mainly in the local portion of the global memory space, i.e., tasks are never transferred to other units. This locality-awareness limits the number of tasks to be discovered to only the tasks that will eventually be executed in that unit: as depicted in Fig. 1.

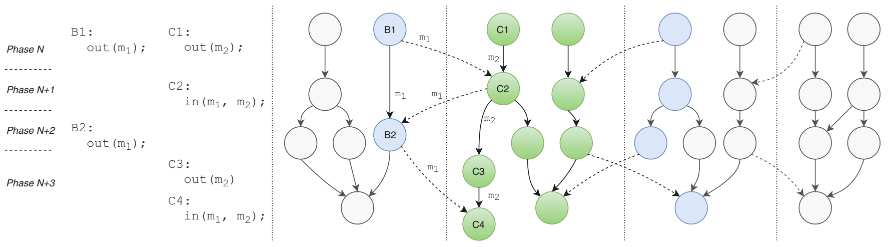


Fig. 1 Distributed task graph discovery: the unit in the middle only discovers its local tasks (green) and should only be concerned with tasks on other units that have dependencies to its local tasks (blue). All other tasks (gray) should not be considered

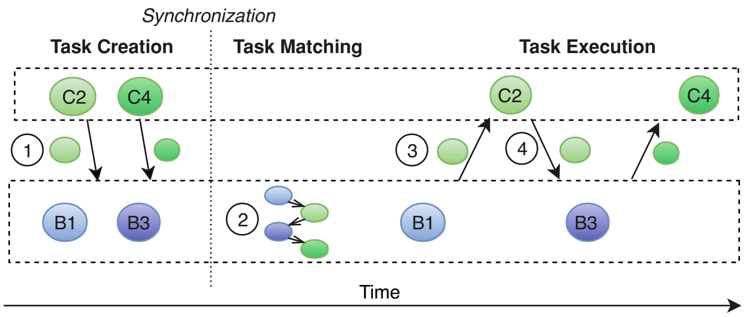


Fig. 2 Scheduler interaction required for handling remote task data dependencies

The discovery of the local task graphs thus happens in parallel and without immediate synchronization between the units, i.e., only the bold edges in Fig. 1 are immediately visible to the individual scheduler instances. In order to connect these trimmed task graphs, the schedulers need to exchange information on dependencies crossing its process boundary, i.e., dependencies referencing non-local global memory locations. The required interaction between the schedulers is depicted in Fig. 2. During the discovery of the trimmed local task graphs, schedulers communicate any encountered dependencies that reference non-local global memory to the unit owning the referenced memory ①. As soon as all dependencies have been communicated, the schedulers extend their local task graphs with the dependency information received from other units ②. A synchronization across all units is required to ensure that all relevant dependency information has been exchanged.

After the extension of the local task graphs, the units start the execution of the tasks. As soon as a task with a dependency to a task on a remote unit, e.g., through an input dependency previously communicated by the remote unit, has completed, the dependency release is communicated to the remote unit ③, where the task will eventually be executed. The completion of the execution is then communicated back to the first scheduler ④ to release any write-after-read dependency, e.g., the dependency $C_2 \rightarrow B_3$ in Fig. 1.

2.1.2 Ordering of Dependencies

As described in Sect. 2.1.1, the local task graphs are discovered by each unit separately. Local edges in the local task graph are discovered similar to the matching rules of OpenMP, i.e., an input dependency refers to the previous output dependency referencing the same memory location (read-after-write), which in turn match with previous input and output dependencies on the same memory location (write-after-read and write-after-write).

However, since the local task graphs are discovered in parallel, the schedulers cannot infer any partial ordering of tasks and dependencies across process boundaries. More specifically, the blue scheduler in Fig. 1 cannot determine the relationship between the dependencies of tasks B_1, B_2, C_2, C_4. The schedulers thus have to rely on additional information provided by the user in the form of *phases* (as depicted in Fig. 1). A task and its output dependencies are assigned to the current phase upon their discovery. Input dependencies always refer to the last matching output dependency in any previous phase while output dependencies match with any previous local input dependency in the same or earlier phase and any remote input dependency in any earlier phase, up to and including the previous output dependency on the same memory location.

As an example, the input dependency of C_2 is assigned the phase $N + 1$ whereas the input dependency of C_4 is assigned the phase $N + 3$. This information can be used to match the output dependency of B_1 in phase N to the input dependency of C_2 and the output dependency of B_2 in phase $N + 2$ to the input dependency of C_4, creating the edges $B_1 \rightarrow C_2$ and $B_2 \rightarrow C_4$. The handling of write-after-read dependencies described in Sect. 2.1.1 creates the edge $C_2 \rightarrow B_2$. The handling of local dependencies happens independent of the phase.

In our model, conflicting remote dependencies in the same phase are erroneous as the scheduler is unable to reliably match the dependencies. Two dependencies are conflicting if at least one of them is non-local and at least one is an output dependency. This restriction allows the schedulers to detect synchronization errors such as underdefined phases and report them to the user. This is in contrast to the collective synchronization through barriers traditionally used in DASH, in which synchronization errors cannot be easily detected and often go unnoticed unless the resulting non-deterministic behavior leads to deviations in the application's results.

2.1.3 Implementation

A single task is created using the `async` function in DASH, which accepts both an action to be performed when the task is executed and a set of dependencies that describe the expected inputs and outputs of the task. In the example provided in Listing 1, every call to `async` (lines 5, 11, 19, and 27) is passed a C++ lambda in addition to input and output dependencies.

Instead of pure input dependencies, the example uses `copyin` dependencies, which combine an input dependency with the transfer of the remote memory range into a local buffer. This allows for both a more precise expression of the algorithms and allows the scheduler to map the transfer onto two-sided MPI communication primitives, which may be beneficial on systems that do not efficiently support MPI RMA. The action performed by the task could still access any global memory location, keeping communication and synchronization decoupled in principle and retaining a one-sided programming model while allowing the use of two-sided communication in the background.

```
 1 dash::Matrix<2, double> matrix{N, N, dash::TILE(NB), dash::TILE(NB)};
 2
 3 for (int k = 0; k < num_blocks; ++k) {
 4   if (mat.block(k,k).is_local()) {
 5     dash::tasks::async([&](){ potrf(matrix.block(k,k)); },
 6       dash::tasks::out(mat.block(k,k)));
 7   }
 8   dash::tasks::async_fence(); // <- advance to next phase
 9   for (int i = k+1; i < num_blocks; ++i)
10     if (mat.block(k,i).is_local())
11       dash::tasks::async([&](){
12           trsm(cache[k], matrix.block(k,i)); },
13         dash::tasks::copyin(mat.block(k,k), cache[k]),
14         dash::tasks::out(mat.block(k,i)));
15   dash::tasks::async_fence(); // <- advance to next phase
16   for (int i = k+1; i < num_blocks; ++i) {
17     for (int j = k+1; j < i; ++j) {
18       if (mat.block(j,i).is_local()) {
19         dash::tasks::async([&](){
     ↪
20             gemm(cache[i], cache[j], mat.block(j,i)); },
21           dash::tasks::copyin(mat.block(k,i), cache[i]),
22           dash::tasks::copyin(mat.block(k,j), cache[j]),
23           dash::tasks::out(mat.block(j,i)));
24       }
25     }
26     if (mat.block(i,i).is_local()) {
27       dash::tasks::async([&](){
28           syrk(cache[i], mat.block(i,i)); },
29         dash::tasks::copyin(mat.block(k,i), cache[i]),
30         dash::tasks::out(mat.block(i,i)));
31     }
32   }
33   dash::tasks::async_fence();  // <- advance to next phase
34 }
35 dash::tasks::complete(); // <- wait for all tasks to execute
```

Listing 1 Implementation of Blocked Cholesky Factorization using global task data dependencies in DASH. Some optimizations omitted for clarity

The specification of phases is done through calls to the `async_fence` function (lines 8, 15, and 33 in Listing 1). Similar to a barrier, it is the user's responsibility to ensure that all units advance phases in lock-step. However, the phase transition triggered by `async_fence` does not incur any communication. Instead, the call causes an increment of the phase counter, whose new value will be assigned to all ensuing tasks.

Eventually, the application waits for the completion of the execution of the global task graph in the call to `complete()`. Due to the required internal synchronization and the matching of remote task dependencies, the execution of all but the tasks in the first phase has to be post-poned until all its dependencies in the global task-graph are known. DASH, however, provides the option to trigger intermediate matching steps triggered by a phase increment and allows the specification of a an upper bound on the number of active phases[1] to avoid the need for discovering the full local task graph before execution starts. This way the worker threads executing threads may be kept busy while the main thread continues discovering the next window in the task graph.

In addition to the single task creation construct described above, DASH also provides the `taskloop()` construct, for which an example is provided in Listing 2. The `taskloop` function divides the iteration space `[begin, end)` into chunks that are assigned to tasks, which perform the provided action on the assigned subrange (lines 7–9). The user may control the size of each chunk (or the overall number of chunks to be created) by passing an instance of `chunk_size` (or `num_chunks`) to the call (Line 5). In addition, the call accepts a second lambda that is used to specify the dependencies of each task assigned a chunk (lines 11–14), which allows a depth-first scheduler to chain the execution of chunks of multiple data-dependent loops, effectively improving cache locality without changing the structure of the loops.

```
 1 dash::Array<int> arr(N);
 2 
 3 if (dash::myid() == 0) {
 4   dash::tasks::taskloop(
 5     arr.begin(), arr.end(), dash::tasks::chunk_size(10),
 6     // task action
 7     [&] (auto begin, auto end) {
 8       // perform action on elements in [begin, end)
 9     },
10     // generate out dependencies on elements in [begin, end)
11     [&] (auto begin, auto end, auto deps) {
12       for (auto it = begin; it != end; ++it)
13         *deps = dash::tasks::out(it);
14     });
15 }
```

Listing 2 Example of using the `dash::taskloop` in combination with a dependency generator

2.1.4 Results: Blocked Cholesky Factorization

The implementation of the Blocked Cholesky Factorization discussed in Sect. 2.1.3 has been compared against two implementations in PaRSEC. The first implementation uses the parameterized task graph (PTG), in which the problem is described as an directed acyclic graph in a domain-specific language called JDF. In this version,

[1]A phase is considered active while a task discovered in that phase has not completed execution.

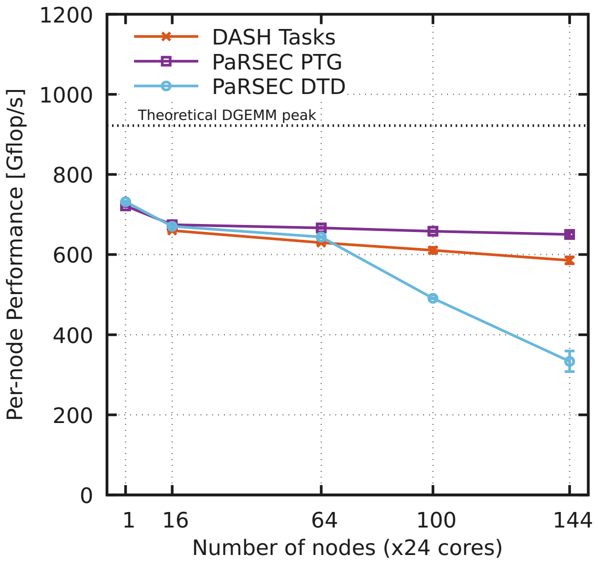


Fig. 3 Per-node weak-scaling performance of Blocked Cholesky Factorization of a matrix with leading dimension $N = 25k/node$ and block size $NB = 320$ on a Cray XC40 (higher is better)

the task graph is not dynamically discovered but is instead inherently contained within the resulting binary.

The second PaRSEC version uses the Dynamic Task Discovery (DTD) interface of PaRSEC, in which problems are expressed with a global view, i.e., all processes discover the global task graph to discover the dependencies to tasks executing on remote processes.

For all runs, a background communication thread has been employed, each time running on a dedicated core, leading to one main thread and 22 worker threads executing the application tasks on the Cray XC40.

The results presented in Fig. 3 indicate that PaRSEC PTG outperforms both DTD and DASH, due to the missing discovery of tasks and their dependencies. DTD exhibits a drop in per-node performance above 64 nodes, which may be explained with the global task graph discovery. Although the per-node performance of DASH does not exhibit perfect scaling, it still achieves about 80% of the performance of PaRSEC PTG at 144 nodes.

2.1.5 Related Work

HPX [18] is an implementation of the ParalleX [17] programming paradigm, in which tasks are spawned dynamically and moved to the data, instead of the data being moved to where the task is being executed. HPX is locality-agnostic in that distributed parallelism capabilities are implicit in the programming model, rather than explicitly exposed to the user. An Active Global Address Space (AGAS) is used to transparently manage the locality of global objects. Synchronization of tasks is expressed using futures and continuations, which are also used to exchange data between tasks.

In the Active Partitioned Global Address Space (APGAS) [27], in contrast, the locality of so-called places is explicitly exposed to the user, who is responsible for selecting the place at which a task is to be executed. Implementations of the APGAS model can be found in the X10 [9] and Chapel [8] languages as well as part of UPC and UPC++ [21].

The Charm++ programming system encapsulates computation in objects that can communicate using message objects and can be migrated between localities to achieve load balancing.

Several approaches have been proposed to facilitate dynamic synchronization by waiting for events to occur. AsyncShmem is an extension of the OpenShmem standard, which allows dynamic synchronization of tasks across process boundaries by blocking tasks waiting for a state change in the global address space [14]. The concept of phasers has been introduced into the X10 language to implement non-blocking barrier-like synchronization, with the distinction of readers and writers contributing to the phaser [29].

Tasklets have recently been introduced to the XcalableMP programming model [35]. The synchronization is modeled to resemble message-based communication, using data dependencies for tasks on the same location and notify-wait with explicitly specified target and tags.

Regent is a region- and task-based programming language that is compiled into C++ code using the Legion programming model to automatically partition the computation into logical regions [4, 30].

The PaRSEC programming system uses a domain specific language called JDF to express computational problems in the form of a parameterized task graph (PTG) [7]. The PTG is implicitly contained in the application and not discovered dynamically at runtime. In contrast to that, the dynamic task discovery (DTD) frontend of PaRSEC dynamically discovers the global task-graph, i.e., each process is aware of all nodes and edges in the graph.

A similar approach is taken by the sequential task flow (STF) frontend of StarPU, which complements the explicit MPI send/recv tasks to encapsulate communication in tasks and implicitly express dependencies across process boundaries [1].

Several task-based parallelization models have been proposed for shared memory concurrency, including OpenMP [3, 24], Intel thread building blocks (TBB) [25] and Cilk++ [26] as well as SuperGlue [34]. With ClusterSs, an approach has been made to introduce the APGAS model into OmpSs [32].

2.2 Dynamic Hardware Topology

Portable applications for heterogeneous hosts adapt communication schemes and virtual process topologies depending on system components and the algorithm scenario. This involves concepts of vertical and horizontal locality that are not based

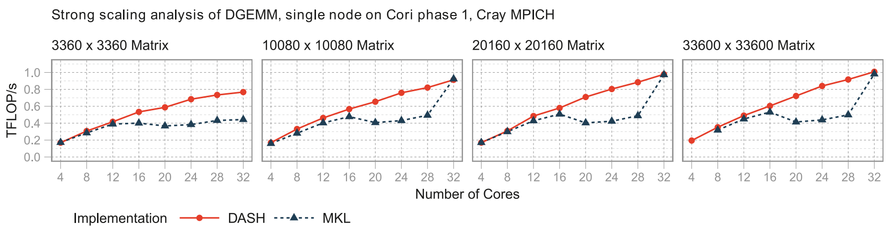


Fig. 4 Strong scaling of matrix multiplication on single node for 4 to 32 cores with increasing matrix size N × N on Cori phase 1, Cray MPICH

on latency and throughput as distance measure. For example in a typical accelerator-offloading scenario, data distribution to processes optimizes for horizontal locality to reduce communication distance between collaborating tasks. For communication in the reduction phase, distance is measured based on vertical locality.

The goal to provide a domain-agnostic replacement for the C++ Standard Template Library (STL) implies portability as a crucial criterion for every model and implementation of the DASH library. This includes additional programming abstractions provided in DASH, such as n-dimensional containers which are commonly used in HPC. These are not part of the C++ standard specifications but comply with its concepts. Achieving portable efficiency of PGAS algorithms and containers that satisfy semantics of their conventional counterparts is a multivariate, hard problem, even for the seemingly most simple use cases.

Performance evaluation of the of the DASH NArray and dense matrix-matrix multiplication abstractions on different system configurations substantiated the portable efficiency of DASH. The comparison also revealed drastic performance variance of the established solutions, for example node-local DGEMM of Intel MKL on Cori phase 1 shown in Fig. 4 which apparently expected a power of two amount of processing cores for multi-threaded scenarios.

The DASH variant of DGEMM internally uses the identical Intel MKL distribution for multiplication of partitioned matrix blocks but still achieves robust scaling. This is because DASH implements a custom, adaptive variant of the SUMMA algorithm for matrix-matrix multiplication and assigns one process per core, each using MKL in sequential mode. This finding motivated to find abstractions that allow expressions for domain decomposition and process placement depending on machine component topology. In this case: to group processes by NUMA domains with one process per physical core.

2.2.1 Locality-Aware Virtual Process Topology

In the DASH execution model, individual computation entities are called *units*. In the MPI-based implementation of the DASH runtime, a unit corresponds to an MPI rank but may occupy a locality domain containing several CPU cores or, in principle, multiple compute nodes.

Units are organized in hierarchical *teams* to match the logical structure of algorithms and machine components. Each unit is an immediate member of exactly one team at any time, initially in the predefined team `ALL`. Units in a team can be partitioned into child teams using the team's `split` operation which also supports locality-aware parameters.

On systems with asymmetric or deep memory hierarchies, it is highly desirable to split a team such that locality of units within every child team is optimized. A locality-aware split at node level could group units by affinity to the same NUMA domain, for example. For this, locality discovery has been added to the DASH runtime. Local hardware information from hwloc, PAPI, libnuma, and LIKWID of all nodes is collected into a global, uniform data structure that allows to query locality information by process ID or scope in the memory hierarchy.

This query interface proved to be useful for static load balancing on heterogeneous systems where team are split depending on the machine component capacities and capabilities. These are stored in a hierarchy of domains with two *property maps*:

Capabilities invariant hardware locality properties that do not depend on the locality graph's structure, like the number of threads per core, cache sizes, or SIMD width

Capacities derivative properties that might become invalid when the graph structure is modified, like memory in a NUMA domain available per unit

Figure 5 outlines the data structure and its concept of hardware abstraction in a simplified example of a topology-aware split. Domain capacities are accumulated from its subdomains and recalculated on restructuring. Team 1 and 2 both contain twelve cores but a different number of units. A specific unit's maximum number of threads is determined by the number of cores assigned to the unit and the number of threads per core.

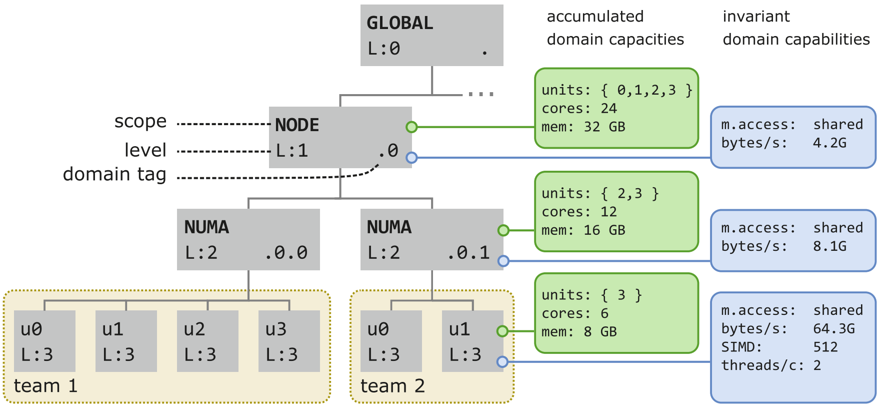


Fig. 5 Domains in a locality hierarchy with domain attributes in dynamically accumulated capacities and invariant capabilities

2.2.2 Locality Domain Graph

The machine component topology of the DASH runtime to support queries and topology-aware restructuring extends the tree-based hwloc topology model to represent properties and relations of machine components in a graph structure. It evolved to the *Locality Domain Graph* (LDG) concept which is available as the standalone library *dyloc*.[2]

In formal terms, a locality domain graph models hardware topology as directed, multi-indexed multigraph. In this, nodes represent *Locality Domains* that refer to any physical or logical component of a distributed system with memory and computation capabilities, corresponding to *places* in X10 or Chapel's *locales* [8]. Edges in the graph are directed and denote the following relationships, for example:

- **Containment** indicating that the target domain is logically or physically contained in the source domain
- **Alias** source and target domains are only logically separated and refer to the same physical domain; this is relevant when searching for a shortest path, for example
- **Leader** the source domain is restricted to communication with the target domain

2.2.3 Dynamic Hardware Locality

Dynamic locality support requires means to specify transformations on the physical topology graph as *views*. Views realize a projection but must not actually modify the original graph data. Invariant properties are therefore stored separately and assigned to domains by reference only.

Conceptually, multi-index graph algebra can express any operation on a locality domain graph, but complex to formulate. When a topology is projected to an acyclic hierarchy, transformations like partitioning, selection and grouping of domains can be expressed in conventional relational or set semantics. A partition or contraction of a topology graph can be projected to a tree data structure and converted to a hwloc topology object (Fig. 6).

A locality domain topology is specific to a team and only contains domains that are populated by the team's units. At initialization, the runtime initializes the default team `ALL` as root of the team hierarchy with all units and associates the team with the global locality graph containing all domains of the machine topology. When a team is split, its locality graph is partitioned among child teams such that a single partition is coherent and only contains domains with at least one leaf occupied by a unit in the child team.

In a map-reduce scenario, dynamic views on machine topology to express for domain decomposition and process placement depending on machine component

[2]https://github.com/dash-project/dyloc.

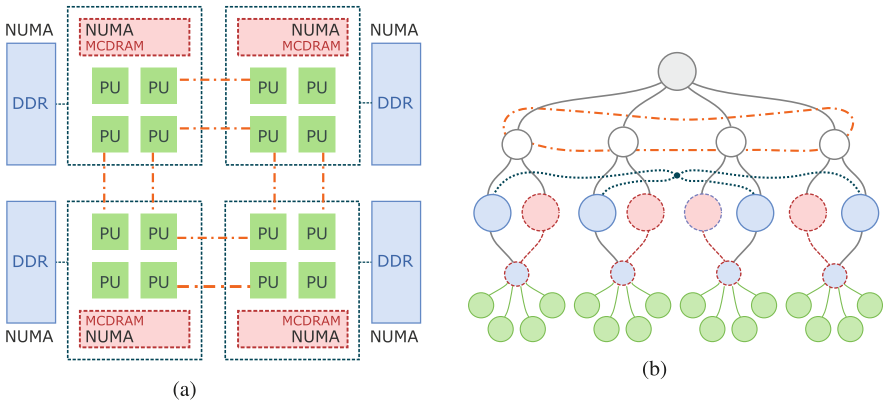


Fig. 6 Illustration of a hardware locality domain graph as a model of node-level system architectures that cannot be correctly or unambiguously represented in a single tree structure. (**a**) Cluster in Intel Knights Landing, configured in Sub-NUMA clustering, Hybrid mode. Contains a quarter of the processor's cores, MCDRAM local memory, affine to DDR NUMA domain. (**b**) Exemplary graph representation of Knights Landing topology in (**a**). Vertex categories model different aspects of component relationships, like cache-coherence and adjacency

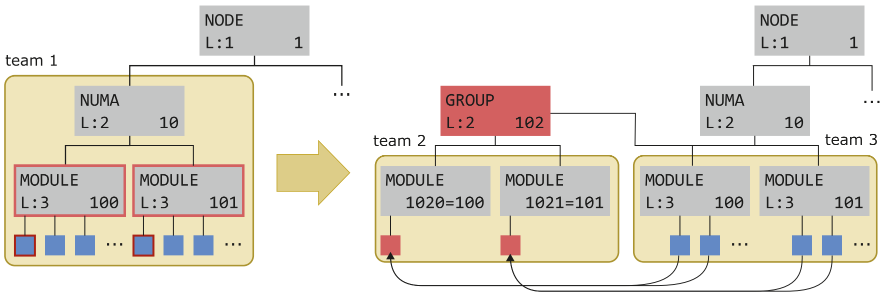


Fig. 7 Illustration of the domain grouping algorithm to define a leader group for vertical communication. One core is selected as leader in domains 100 and 110 and separated into a group. To preserve the original topology structure, the group includes their parent domains and is added as a subdomain of their lowest common ancestor

topology and improve portable efficiency. In the map phase, the algorithm is mostly concerned with *horizontal locality* in domain decomposition to distribute data according to the physical arrangement of cooperating processes. In the reduce phase, *vertical locality* of processes in the component topology determines efficient upwards communication of partial results. The locality domain graph can be used to project hardware topology to tree views for both cases. Figure 7 illustrates a locality-aware split of units in two modules such that one unit per module is selected for upwards communication. This principle is known as *leader communication scheme*. Partial results of units are then first reduced at the unit in the respective leader team. This drastically reduces communication overhead as the physical bus between the

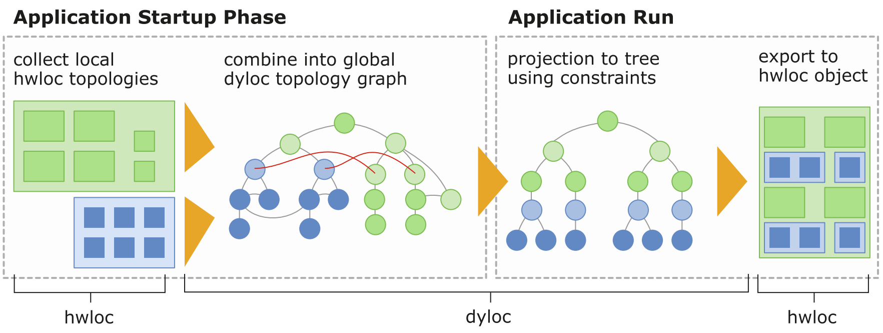

Application Startup Phase
Application Run
collect local hwloc topologies
combine into global dyloc topology graph
projection to tree using constraints
export to hwloc object
hwloc
dyloc
hwloc

Fig. 8 Using dyloc as intermediate process in locality discovery

modules and their NUMA node is only shared by two leader processes instead of all processes in the modules (Fig. 8).

2.2.4 Supporting Portable Efficiency

As an example of both increased depth of the machine hierarchy and heterogeneous node-level architecture, the SuperMIC system[3] consists of 32 compute nodes with symmetric hardware configuration of two NUMA domains, each containing an Ivy Bridge (8 cores) host processor and a Xeon Phi "Knights Corner" coprocessors (Intel MIC 5110P) as illustrated in Fig. 9.

For portable work load balancing on heterogeneous systems, domain decomposition and virtual process topology must dynamically adapt the machine components' inter-connectivity, capacities and capabilities.

Capacities: Total memory capacity on MIC modules is 8 GB for 60 cores, significantly less than 64 GB for 32 cores on host level

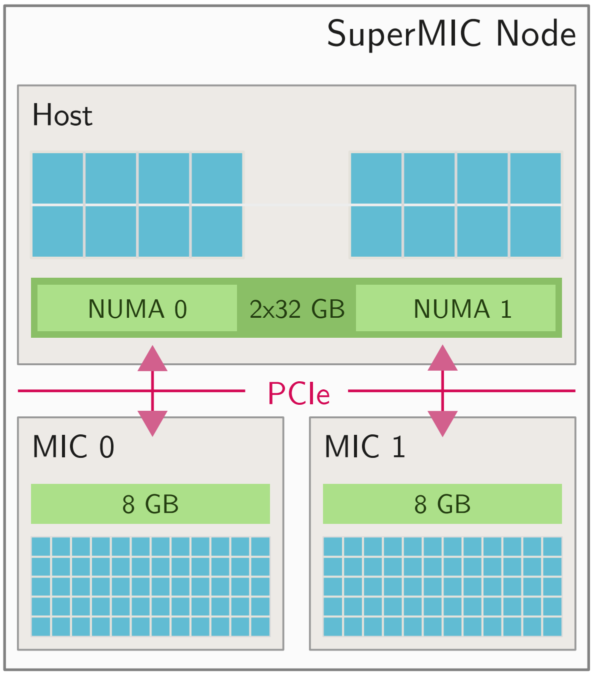
Capabilities: MIC cores have a base clock frequency of 1.1 GHz and 4 SMT threads, with 2.8 GHz and 2 SMT threads on host level

To illustrate the benefit of dynamic locality, we briefly discuss the implementation of the `min_element` algorithm in DASH. Its original variant is implemented as follows: domain decomposition divides the element range into contiguous blocks of identical size. All units then run a thread-parallel scan on their local block for a local minimum and enter a collective barrier once it has been found. Once all units finished their local work load, local results are reduced to the global minimum.

Listing 3 contains the abbreviated implementation of the `min_element` scenario utilizing runtime support based on a dynamic hardware locality graph.

[3]https://www.lrz.de/services/compute/supermuc/supermic.

Fig. 9 SuperMIC node



Dynamic topology queries are utilized in three essential ways to improve overall load-balance: In domain decomposition (lines 3–6), to determine the number of threads available to the respective unit (line 19) and for a simple leader-based communication scheme (lines 8–10, 26).

This implementation achieves portable efficiency across systems with different memory hierarchies and hardware component properties, and dynamically adapts to runtime-specific team size, range size, and available hardware components assigned to the team. Figure 10 shows timeline plots comparing time to completion and process idle time from a benchmark run executed on SuperMIC.

```
 1 // Dynamic topology-aware domain decomposition depending on
 2 // machine component properties and number of units in team:
 3 TeamLocality        tloc(dash::Team::All());
 4 LocBalancedPattern pattern(array_size, tloc);
 5 dash::Array<T>      array(pattern);
 6
 7 GlobIt min_element(GlobIt first, GlobIt last) {
 8   auto uloc      = UnitLocality(myid());
 9   auto leader    = uloc.at_scope(scope::MODULE)
10                       .unit_ids()[0];
11   auto loc_min   = first;
12
13   // Allocate shared variable for reduction result at leader:
14   dash::Shared<GlobIt> glob_min(leader);
15   // Allocate shared array for local minimum values:
  dash::Array<GlobIt>(dash::Team::All().size()) loc_mins;

  // Dynamic query of locality runtime for number of threads:
  auto nthreads = uloc.num_threads();
  #pragma omp parallel for num_threads(nthreads)
  for (...) { /* ... find local result ... */ }
  // Local write, no communication
  loc_mins[my_id] = loc_min;
  dash::barrier();

  if (myid() == leader) {
     // leader reduces local results (instead of all-to-all
     // reduction)
     glob_min = std::min_element(loc_mins.begin(),
                                 loc_mins.end());

  }
  // ...
}
```

Listing 3 Code excerpt of the modified min_element algorithm

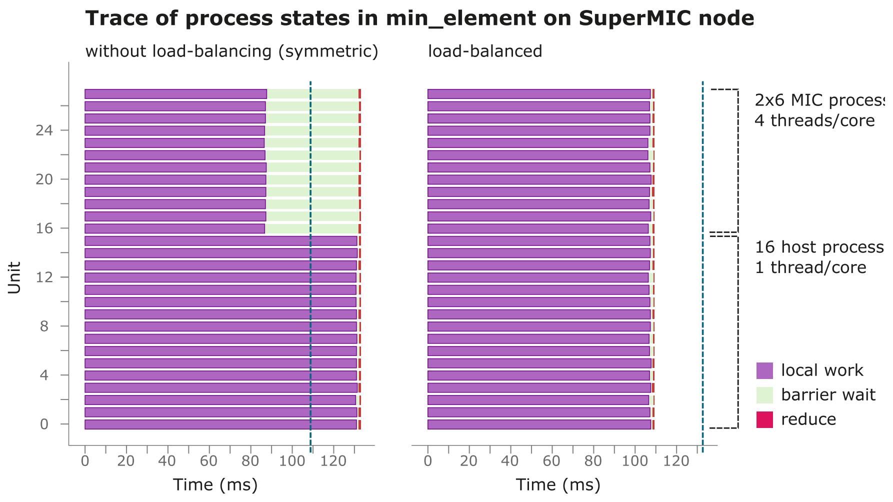


Fig. 10 Trace of process activities in the `min_element` algorithm exposing the effect of load balancing based on dynamic hardware locality

3 DASH C++ Data Structures and Algorithms

The core of DASH is formed by data structures and algorithms implemented as C++ templates. These components are conceptually modeled after their equivalents in the C++ standard template library, shortening learning curves and increasing programmer productivity. A basic DASH data structure is the distributed array with configurable data distribution (`dash::Array`) which closely follows the functionality of a STL `vector` except for the lack of runtime resizing. DASH also offers a multidimensional array and supports a rich variety of data distribution patterns [11]. A focus of the second half of the funding period was placed on *smart* data structures which are more specialized and support users in the development of certain types of applications. One such data structures for the development of stencil-based applications is described in Sect. 3.1.

Similar to data structures, DASH also offers generalized parallel algorithms. Many of the over 100 generic algorithms contained in the STL have an equivalent in DASH (e.g., `dash::fill`). One of the most useful but also challenging algorithms is sorting and Sect. 3.2 describes our implementation of scalable distributed sorting in the DASH library.

3.1 Smart Data Structures: Halo

Typical data structure used in ODE/PDE solvers or 2D/3D image analyzers are multi-dimensional arrays. The DASH NArray distributes data elements of a structured data grid and can be used similar to STL containers. But PDE solvers use stencil operations, not using the current data elements (center), but surrounding data elements (neighbors) as well. The use of the NArray it self is highly inefficient with stencil operations, because neighbors located in another sub-arrays may require remote access (via RDMA or otherwise). A more efficient approach is the use of so called "halo areas". These areas contain copies of all required neighbor elements located on other compute nodes. The halo area width depends on the shape of the stencils and is determined by the largest distance from the center (per dimension). The stencil shape defines all participating data elements—center and neighbors. Figure 11 shows two 9-point stencils with different shapes. The first stencil shape (a) accesses ± 2 data elements in both horizontal and vertical direction and the second one (b) accesses ± 1 stencil point in each direction. While the stencil shape Fig. 11a needs four halo areas with a width of two data elements. The other stencil shape requires eight halo areas with a width one data elements. Using halo areas ensures local data access for all stencil operations used on each sub-array.

The Dash NArray Halo Wrapper wraps the local part of the NArray and automatically sets up a halo environment for stencil codes and halo accesses. Figure 12 shows an overview about all main components, which are explained in the following.

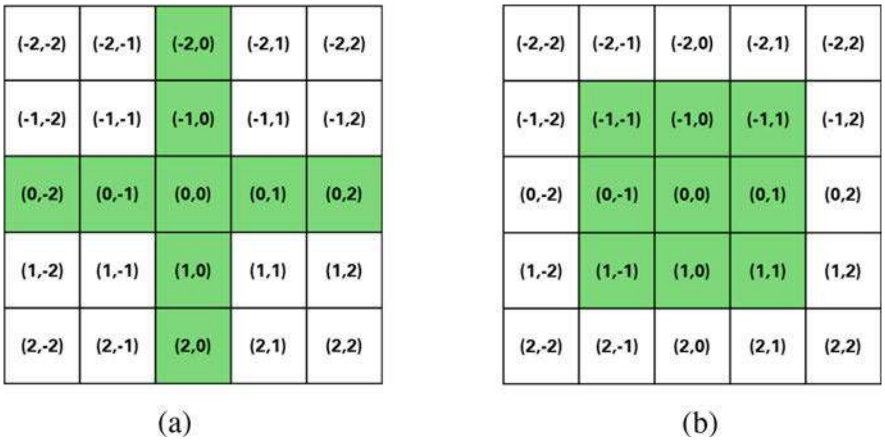


Fig. 11 Two shapes of a 9-point stencil. (**a**) ±2 center stencil in horizontal and vertical directions. (**b**) Center ±1 stencil point in each direction

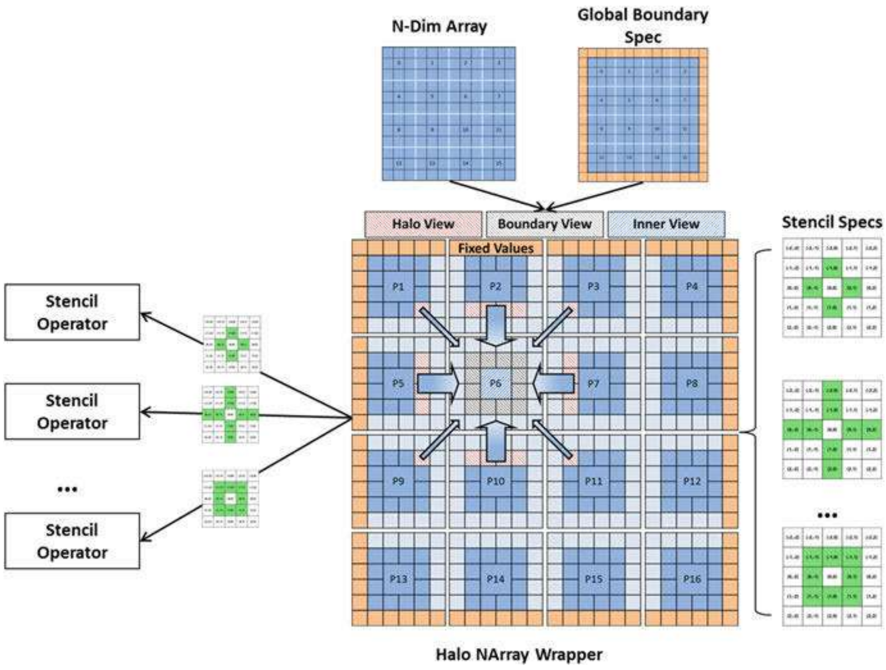


Fig. 12 Architecture of the DASH Halo NArray Wrapper

3.1.1 Stencil Specification

The discretization of the problem to be solved always determines the stencil shape to be applied on the structured grid. For a DASH based code this has to be specified as a stencil specification (StencilSpec), which is a collection of stencil points (StencilPoint). A StencilPoint consists of coordinates relative to the center and an optional weight (coefficient). The StencilSpec specified in Listing 4 describes the stencil shown in Fig. 11b. The center doesn't have to be declared directly.

```
using PointT = dash::halo::StencilPoint<2>;
dash::halo::StencilSpec<PointT,6> stencil_spec(
                PointT(-1,-1), PointT(-1, 0), PointT(-1,1),
                PointT( 0,-1),                , PointT( 0,1),
                PointT( 1,-1), PointT( 1, 0), PointT( 1,1));
```

Listing 4 Stencil specification for an 9-point stencil

3.1.2 Region and Halo Specifications

The region specification (RegionSpec) defines the location of all neighboring partitions. Every unit keeps 3^n regions representing neighbor partitions for "left", "middle", and "right" in each of the n dimensions. All regions are identified by a region index and a corresponding region coordinate. Indexing are done with the Row Major linearization (last index grows fastest). Figure 13 shows all possible regions with its indexes and coordinates for a two dimensional scenario. Region 4 with the coordinates (1,1) is mapped to the center region and represents the local partition. Region 6 (2,0) points to a remote partition located in the south west.

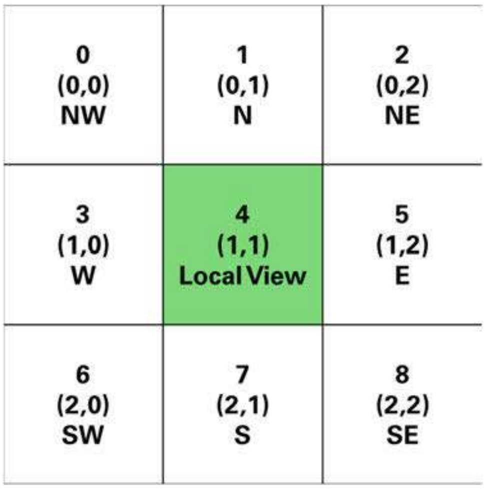


Fig. 13 Mapping of region coordinates and indexes

The halo specification (HaloSpec) uses the RegionSpec to map neighbor partitions which are the origins for halo copies to the local halo areas. From one or multiple StencilSpecs it infers which neighbor partitions are necessary. In case no StencilPoint has a negative offset from the center in horizontal direction, no halo regions for the 'NW', 'W', and 'SW' (Fig. 13) need to be created. If no StencilPoint has diagonal offsets (i.e. only one non-zero coordinate in the offsets) the diagonal regions 'NW','NE', 'SW', and 'SE' can be omitted.

3.1.3 Global Boundary Specification

Additionally, the global boundary specification (GlobalBoundarySpec) allows to control the behavior at the outside of the global grid. For convenience, three different scenarios are supported. The default setting is `NONE` meaning that there are no halo areas in this direction. Therefore, the stencil operations are not applied in the respective boundary region where the stencil would require the halo to be present. As an alternative, the setting `CYCLIC` can be set. This will wrap around the simulation grid, so that logically the minimum coordinate becomes a neighbor to the maximum coordinate. Furthermore, the setting `CUSTOM` creates a halo area but never performs automatic update of its elements from any neighbors. Instead, this special halo area can be written by the simulation (initially only or updated regularly). This offers a convenient way to provide boundary conditions to a PDE solver. The GlobalBoundarySpec can be defined separately per dimension.

3.1.4 Halo Wrapper

Finally, using the aforementioned specifications as inputs, the halo wrapper (HaloWrapper) creates HaloBlocks for all local sub-arrays. The halo area extension is derived from all declared StencilSpecs by determining the maximum offset of any StencilPoint in the given direction.

The mapping between the local HaloBlocks and the halo regions pointing to the remote neighbor data elements is subjected to the HaloWrapper, as well as the orchestration of efficient data transfers for halo data element updates. The data transfer has to be done block-wise instead of element-wise to gain decent performance. While the HaloBlock can access contiguous memory, the part of the neighbor partition marked as halo area, usually can't be accessed contiguously—compare Fig. 14. Therefore, the HaloWrapper relies on DART's support for efficient strided data transfers.

The halo data exchange can be done per region or for all regions at once. It can be called asynchronously and operates independent between all processes and doesn't use process synchronization. The required subsequent wait operation waits for local completion only.

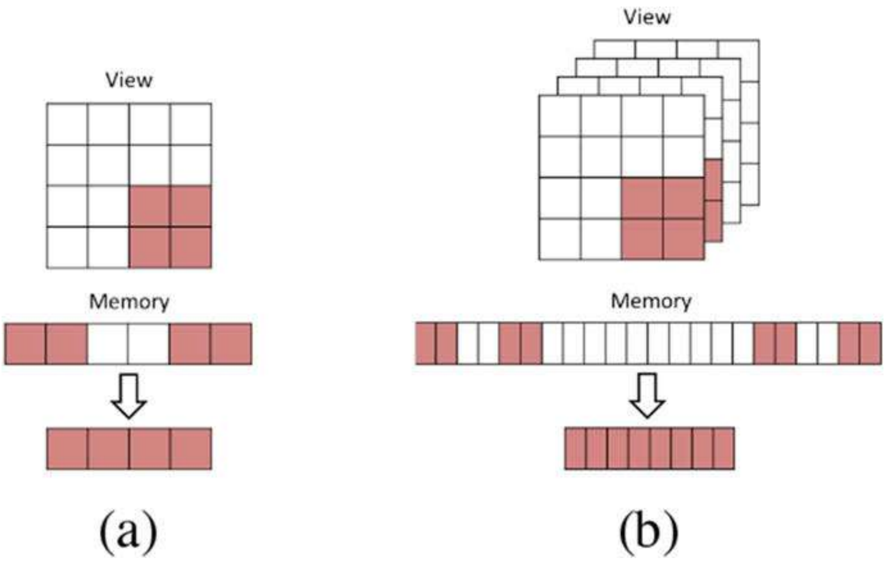


Fig. 14 Halo data exchange of remote strided data to contiguous memory: (**a**) in 2D a corner halo region has a fixed stride, whereas (**b**) in 3D the corner halo region has two different strides

3.1.5 Stencil Operator and Stencil Iterator

So far, the HaloWrapper was used to create HaloBlocks to fit all given StencilSpecs. Besides that, the HaloWrapper also provides specific views and operations for each StencilSpec.

First, for every StencilSpec the HaloWrapper provides a StencilOperator with adapted *inner* and *boundary* views. The inner view contains all data elements that don't need a halo area when using the given stencil operation. All other data elements are marked via the boundary view. These two kind of views are necessary to overlap the halo data transfer with the inner computation. The individual view per StencilSpec allows to make the inner view as large as possible, regardless of other StencilSpecs.

Second, the HaloWrapper offers StencilSpec specific StencilIterators. They iterate over all elements assigned by a given view (inner) or a set of views (boundary). With these iterators center elements can be accessed—equivalent to STL iterators—via the dereference operator. Neighboring data elements can be accessed with a provided method. Stencil points pointing to elements within a halo area, are resolved automatically without conditionals in the surrounding code. StencilIterators can be used with random access, but are optimized for the increment operator.

3.1.6 Performance Comparison

A code abstraction hiding complexity is useful only, if no or minor performance impact is added. Therefore, a plain MPI implementation of a heat equation was compared to a DASH based one regarding weak and strong scaling behavior. All measurements were performed on the Bull HPC-Cluster "Taurus" at ZIH, TU Dresden. Each compute node has two Haswell E5-2680 v3 CPUs at 2.50 GHz with 12 physical cores each and 64 GB memory. Both implementations were built with gcc 7.1.0 and OpenMPI 3.0.0.

The weak scaling scenario increases the number of grid elements proportional to the number of compute nodes. The accumulated main memory is almost entirely used up by each compute grid. Figure 15 shows that both implementations almost have identical and perfect weak scaling behavior. Note that the accumulated waiting times differ significantly. This is due to two effects. One is contiguous halo areas (north and south) vs. strided halo areas (east and west). The other is intra node vs. inter node communication.

The strong scaling scenario uses $55{,}000^2$ grid elements to fit into the main memory of a single compute node. It is solved with 1 to 768 *CPU cores* (== MPI ranks), where 24 cores equals to one full compute node and 768 cores to 32 compute nodes. Figure 16 shows again an almost identical performance behavior between DASH and MPI for the total runtime. Notably, both show the same performance artifact around 16 to 24 cores. This can be ascribed to an increased number of last level cache load misses which indicates that both implementations are memory bound at this number of cores per node.

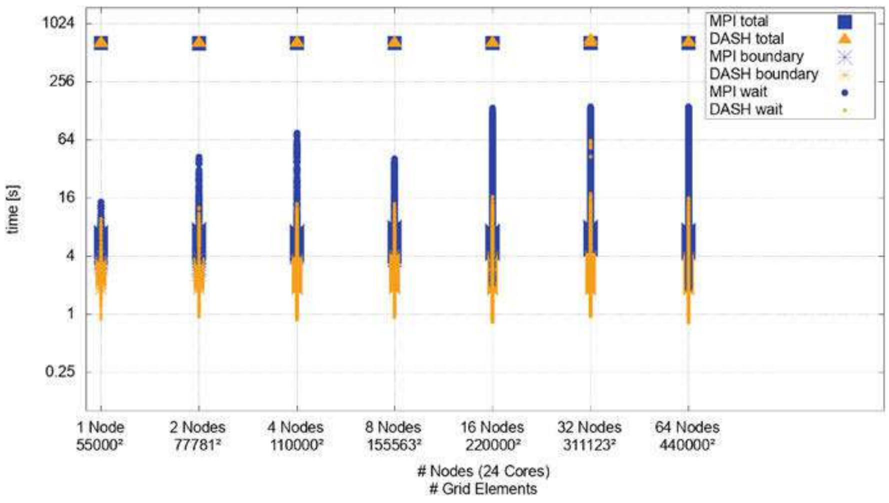


Fig. 15 Weak scaling in DASH vs. MPI

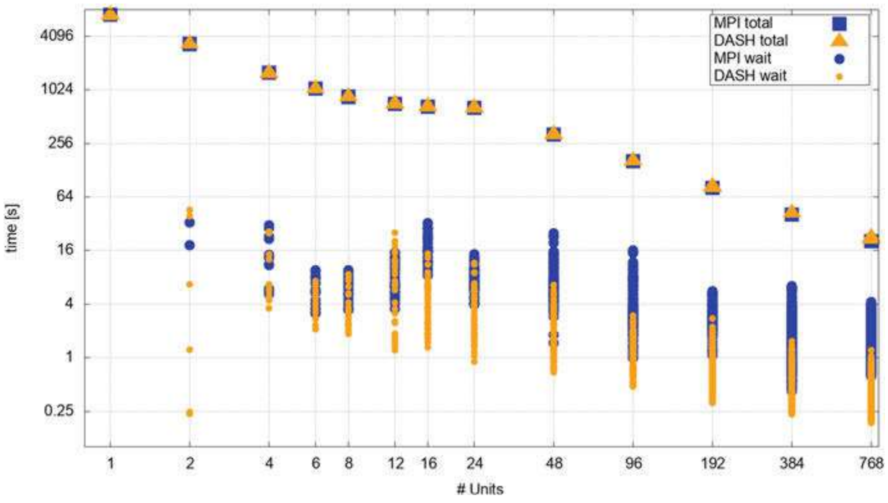


Fig. 16 Strong scaling for 55,000 × 55,000 elements in DASH vs. MPI. Effective wait time for asynchronous halo exchanges is shown in addition

3.2 Parallel Algorithms: Sort

Sorting is one of the most important and well studied non-numerical algorithms in computer science and serves as a basic building block in a wide spectrum of applications. A notable example in the scientific domain are N-Body particle simulations which are inherently communication bound due to load imbalance. Common strategies to mitigate this problem include redistributing particles according to a space filling curve (e.g., Morton Order) which can be achieved with sorting. Other interesting use cases which can be addressed using DASH are Big Data applications, e.g., Google PageRank.

Key to achieve performance is obviously to minimize communication. This applies not only to distributed memory machines but to shared memory architectures as well. Current supercomputers facilitate nodes with large memory hierarchies organized in a growing number of NUMA domains. Depending on the data distribution, sorting is subject to a high fraction of data movement and the more we communicate across NUMA boundaries the more negative the result performance impact becomes.

In the remainder of this section we briefly describe the problem of sorting in a more formal manner and summarize the basic approaches in related work. It follows a more detailed elaboration of our sorting algorithm [20]. Case studies on both distributed and shared memory demonstrate our performance efficiency. Results reveal that we can outperform state of the art implementations with our PGAS algorithm.

3.2.1 Preliminaries

Let X be a set of N keys evenly partitioned among P processors, thus, each processor contributes $n_i \sim N/P$ keys. We further assume there are no duplicate keys which can technically be achieved in a straightforward manner. Sorting permutes all keys by a predicate which is a binary relation in set X. Recursively applying this predicate to any ordered pair (x, y) drawn from X enables to determine the rank of an element $I(x) = k$ with x as the k-th order statistic in X. Assuming our predicate is *less than* (i.e., $<$) the output invariant after sorting guarantees that for any two subsequent elements $x, y \in X$

$$x < y \Leftrightarrow I(x) < I(y).$$

Scientific applications usually require a balanced load to maximize performance. Given a load balance threshold ϵ, *local balancing* means that in the sorted sequence each processor P_i owns at most $N(1+\epsilon)/P$ keys. This does not always result in a *globally balanced* load which is an even stronger guarantee.

Definition 1 For all $i \in \{1..P\}$ we have to determine splitter S_i to partition the input sequence into P subsequences such that

$$\frac{Ni}{P} - \frac{N\epsilon}{2P} \leq I(s_i) \leq \frac{Ni}{P} + \frac{N\epsilon}{2P}$$

Determining these splitters boils down to the *k-way selection* problem which is a core algorithm in this work. If $\epsilon = 0$ we need to perfectly partition the input which increases communication complexity. However, it often is the easiest solution in terms of programming productivity which is a major goal of the DASH library.

3.2.2 Related Work

Sorting large inputs can be achieved through parallel sample sort which is a generalization of *Quicksort* with multiple pivots [5]. Each processor partitions local elements into p pieces which are obtained out of a sufficiently large sample of the input. Then, all processors exchange elements among each other such that piece i is copied to processor i. In a final step, all processors sort received pieces locally, resulting in a globally sorted sequence. *Perfect partitioning* can be difficult to achieve as splitter selection is based only on a sample of the input.

In parallel p-way mergesort each processor first sorts the local data portion and subsequently partitions it, similar to sample sort, into p pieces. Using an ALL-TO-ALL exchange all pieces are copied to the destination processors which finally merge them to obtain a globally sorted sequence. Although this algorithm has worse isoefficiency due to the partitioning overhead compared to sample sort, *perfect partitioning* becomes feasible since data is locally sorted.

Scalable sorting algorithms are compromises between these two extremes and apply various strategies to mitigate negative performance impacts of splitter selection (partitioning) and ALL-TO-ALL communication [2, 15, 16, 31]. Instead of communicating data only once, partitioning is done recursively from a coarse-grained to a more fine-grained solution. Each recursion leads to independent subpartitions until the solution is found. Ideally, the level of recursion maps to the underlying hardware resources and network topology.

This work presents two contributions. First, we generalize a distributed selection algorithm to achieve scalable partitioning [28]. Second, we address the problem of communication-computation overlap in the ALL-TO-ALL exchange, which is conceptually limited in MPI as the underlying communication substrate.

3.2.3 Histogram Sort

The presented sorting algorithm consists of four supersteps as delineated in Fig. 17.

Local Sort Sorts the local portion using a fast shared memory algorithm.
Splitting Each processor partitions the local array into p pieces. We generalize distributed selection to a p-way multiselect.
Data Exchange Each processor exchanges piece i with processor i according to the splitter boundaries.
Local Merge Each processor merges the received sorted pieces.

Splitting is based on distributed selection [28]. Instead of finding one pivot we collect multiple pivots (splitters) in a single iteration, one for each *active* range. If a pivot matches a specific rank we do not consider this range anymore and discard it from the set of active ranges. Otherwise, we examine each of the two resulting

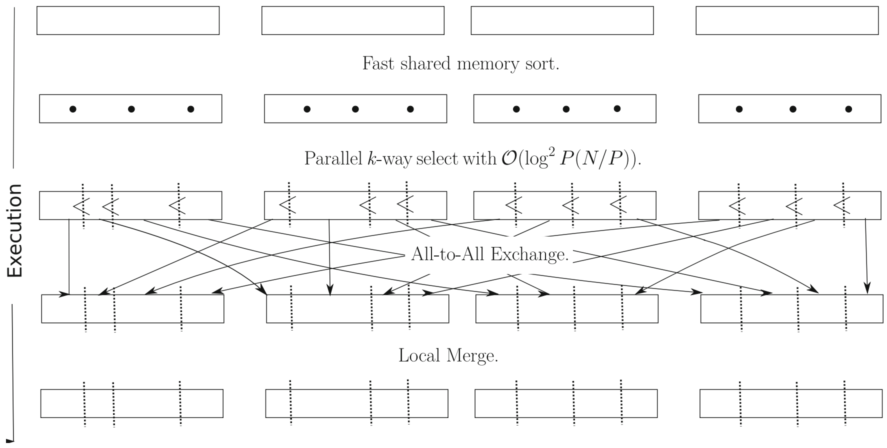


Fig. 17 Algorithmic schema of `dash::sort` with four processors ($P = 4$)

subranges whether they need to be considered in future iterations and add them to the set of active ranges. The vector of splitters follows from Definition 1 (page 126) and is the result of a prefix sum over the local capacities of all processors.

Another difference compared to the original algorithm [28] is that in our case we replace the local partition with binary search which is possible due to our initial sorting step in all processors. Thus, to determine a local histogram over p pieces requires logarithmic computation complexity instead of linear complexity. A global histogram to determine if all splitters are valid (or if we need to refine the boundaries) requires a single REDUCE over all processors with logarithmic communication complexity.

The question how many iterations we need is answered as follows. We know that for the base case with only two processors (i.e., only one splitter) distributed selection has a recursive depth of $\mathcal{O}(\log p)$. This follows from the weighted median for the pivot selection which guarantees a reduction of the working set by at least one quarter each iteration. As described earlier instead of a single pivot we collect multiple pivots in a single iteration which we achieve by a list of active ranges. Although the local computation complexity increases by a factor of $\mathcal{O}(\log p)$ the recursion depth does not change. Hence, the overall communication complexity is $\mathcal{O}(\log^2 p)$ including the REDUCE call each iteration.

After successfully determining the splitters all processors communicate the locally partitioned pieces with an ALL-TO-ALL exchange. Merging all received pieces leads to a globally sorted sequence over all processors. Due to the high communication volume communication-computation overlap is required to achieve good scaling efficiency. However, for collective operations MPI provides a very limited interface. While we can use a non-blocking ALL-TO-ALL we cannot operate on partially received pieces. For this reason we designed our own ALL-TO-ALL algorithm to pipeline communication and merging. Similar to the Butterfly algorithm processor i sends to destination $(i + r) \pmod p$ and receives from $(i - r) \pmod p$ in round r [33]. However we schedule communication requests only as long as a communication buffer of a fixed length is not completely allocated. As soon as *some* communication requests complete we schedule new requests while merging the received chunks. PGAS provides additional optimizations. For communication within a shared memory node we use a cache-efficient ALL-TO-ALL algorithm to minimize negative cache effects among the involved processors. Instead of scheduling send receive pairs processor ranks are reordered according to a Morton order. Data transfer is performed using one-sided communication mechanisms. Similar optimizations can be applied to processor pairs running on nearby nodes. We are preparing a paper to describe the involved optimizations in more detail. First experimental evaluations reveal that we can achieve up to 22% speedup compared to a single ALL-TO-ALL followed by a p-way local merge.

In the next section we demonstrate our performance scalability against a state-of-the-art implementation in Charm++.

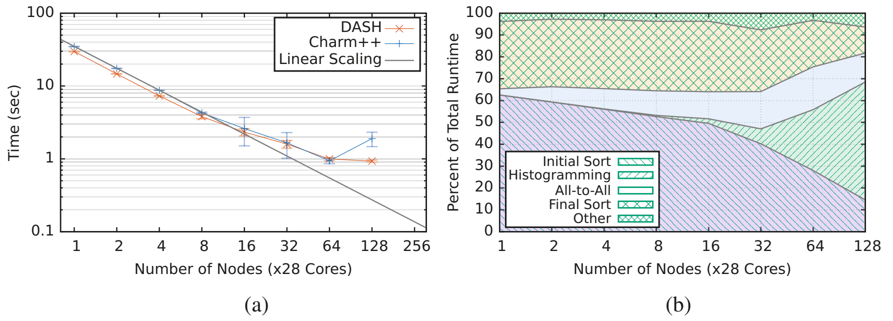


Fig. 18 Strong scaling study with Charm++ and DASH. (**a**) Median execution time. (**b**) Strong scaling behavior of `dash::sort`

3.2.4 Evaluation and Conclusion

We conducted the experiments on SuperMUC Phase 2 hosted at the Leibnitz Supercomputing Center. This system is an island-based computing cluster, each equipped with 512 nodes. Each node has two Intel Xeon E5-2697v3 14-core processors with a nominal frequency of 2.6 GHZ and 64 GB of memory, although only 56 GB are usable due to the operating system. Computation nodes are interconnected in a non-blocking fat tree with Infiniband FDR14 which achieves a peak bisection bandwidth of 5.1 TB/s. We compiled our binaries with Intel ICC 18.0.2 and linked the Intel MPI 2018.2 library for communication. The Charm++ implementation was executed using the most recent stable release.[4] On each node we scheduled only 16 MPI ranks (28 cores available) because the Charm++ implementation requires the number of ranks to be a power of two. We emphasize that our implementation in DASH does not rely on such constraints.

The strong scaling performance results are depicted in Fig. 18a. We sort 28 GBytes of uniformly distributed 64-bit signed integers. This is the maximum memory capacity on a single node because our algorithm is not in-place. We always report the median time out of 10 executions along with the 95% confidence interval, excluding an initial warmup run. For Charm++ we can see wider confidence intervals. We attribute this to a volatile histogramming phase which we can see after analyzing generated log files in the Charm++ experiments. Overall, we observe that both implementations achieve nearly linear speedup with a low number of cores. Starting from 32–64 nodes scalability gets worse. DASH still achieves a scaling efficiency of ≈0.6 on 3500 cores while Charm++ is slightly below. Figure 18b visualizes the relative fraction of the most relevant algorithm phases in a single run. It clearly identifies histogramming as the bottleneck if we scale up the number of processors. This is not surprising because with 128 nodes (2048 ranks) each rank operates on only 8 MB of memory.

[4] v6.9.0, http://charmplusplus.org/download/.

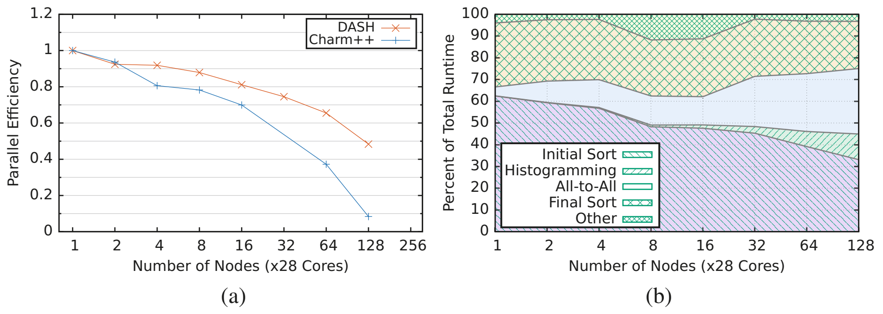


Fig. 19 Weak scaling study with Charm++ and DASH. (**a**) Weak scaling efficiency. (**b**) Weak scaling behavior of `dash::sort`

Figure 19a depicts the weak scaling efficiency. The absolute median execution time for DASH started from 2.3 s on one node and ended with 4.6 s if we scale to 128 nodes (3584 cores). As expected, the largest fraction of time is consumed in local sorting and the ALL-TO-ALL data exchange because we have to communicate 256 GB across the network. Figure 19b confirms this. The collective ALLREDUCE of $P - 1$ splitters among all processors in histogramming overhead is almost amortized from the data exchange which gives an overall good scalability for DASH. The Charm++ histogramming algorithm again shows high volatility with running times from 5–25 s, resulting in drastic performance degradation.

Our implementation shows good scalability on parallel machines with a large processor count. Compared to other algorithms we do not pose any assumptions on the number of ranks, the globally allocated memory volume or the key distribution. Performance measurements reveal that our general purpose approach does not result in performance degradation compare to other state-of-the-art algorithms. Our optimized MPI ALL-TO-ALL exchange with advanced PGAS techniques shows how we can significantly improve communication-computation overlap. Finally, the STL compliant interface enables programmers to easily integrate a scalable sorting algorithm into scientific implementations.

4 Use Cases and Applications

4.1 A Productivity Study: The Cowichan Benchmarks

In this section we present an evaluation of DASH focusing on productivity and performance by comparison with four established parallel programming approaches (Go, Chapel, Cilk, TBB) using the Cowichan set of benchmark kernels.

4.1.1 The Cowichan Problems

The Cowichan problems [36], named after a tribal area in the Canadian Northwest, is a set of small benchmark kernels that have been developed primarily for the purpose of assessing the usability of parallel programming systems. There are two versions of the Cowichan problems and here we restrict ourselves to a subset of the problems found in the second set. The comparison presented in this section is based on previous work by Nanz et al. [23] as we use their publicly available code[5] to compare with our own implementation of the Cowichan benchmarks using DASH. The code developed as part of a study by Nanz et al. has been created by expert programmers in Go, Chapel, Cilk and TBB and can thus be regarded as idiomatic for each approach and free of obvious performance defects.

The five (plus one) problems we consider in our comparison are the following:

randmat: Generate a (`nrows` × `ncols`) matrix `mat` of random integers in the range $0, \ldots, \texttt{max} - 1$ using a deterministic pseudo-random number generator (PRNG).

thresh: Given an integer matrix `mat`, and a thresholding percentage `p`, compute a boolean matrix `mask` of similar size, such that `mask` selects `p` percent of the largest values of `mat`.

winnow: Given an integer matrix `mat`, a boolean matrix `mask`, and a desired number of target elements `nelem`, perform a *weighted point selection* operation using sorting and selection.

outer: Given a vector of `nelem` (row, col) points, compute an (`nelem` × `nelem`) *outer product* matrix `omat` and a vector `vec` of floating point values based on the Euclidean distance between the points and the origin, respectively.

matvec: Given an `nelem` × `nelem` matrix `mat` and a vector `vec`, compute the matrix-vector product (row-by-row inner product) `res`.

chain: Combine the kernels in a sequence such that the output of one becomes the input for the next. I.e., chain = randmat ∘ thresh ∘ winnow ∘ outer ∘ matvec.

4.1.2 The Parallel Programming Approaches Compared

We compare our implementation of the Cowichan problems with existing solutions in the following four programming approaches.

Chapel [8] is an object-oriented partitioned global address space (PGAS) programming language developed since the early 2000s by Cray, originally as part of DARPA's High Productivity Computing Systems (HPCS) program. We have used Chapel version 1.15.0 in our experiments.

Go [10] is a general-purpose systems-level programming language developed at Google in the late 2000s that focuses on concurrency as a first-class concern. Go supports lightweight threads called *goroutines* which are invoked by prefixing

[5]https://bitbucket.org/nanzs/multicore-languages/src.

a function call with the go keyword. Channels provide the idiomatic way for communication between goroutines but since all goroutines share a single address space, pointers can also be used for data sharing. We have used Go version 1.8 in our experiments.

Cilk [6] started as an academic project at MIT in the 1990s. Since the late 2000s the technology has been extended and integrated as Cilk Plus into the commercial compiler offerings from Intel and more recently open source implementations for the GNU Compiler Collection (GCC) and LLVM became available. Cilk's initial focus was on lightweight tasks invoked using the spawn keyword and dynamic workstealing. Later a parallel loop construct (cilk_for) was added. We have used Cilk as integrated in Intel C/C++ compilers version 18.0.2.

Intel Threading Building Blocks (TBB) [25] is a C++ template library for parallel programming that provides tasks, parallel algorithms and containers using a work-stealing approach that was inspired by the early work on Cilk. We have used TBB version 2018.0 in our experiments, which is part of Intel Parallel Studio XE 2018.

4.1.3 Implementation Challenges and DASH Features Used

In this section we briefly describe the challenges encountered when implementing the Cowichan problems, a more detailed discussion can be found in a recent publication [13]. Naturally this small set of benchmarks only exercises a limited set of the features offered by either programming approach. However, we believe that the requirements embedded in the Cowichan problems are relevant to a wide set of other uses cases, including the classic HPC application areas.

Memory Allocation and Data Structure Instantiation The Cowichan problems use one- and two-dimensional arrays as the main data structures. 1D arrays are widely supported by all programming systems. True multidimensional arrays, however, are not universally available and as a consequence workarounds are commonly used. The Cilk and TBB implementation both adopt a linearized representation of the 2D matrix and use a single malloc call to allocate the whole matrix. Element-wise access is performed by explicitly computing the offset of the element in the linearized representation by $mat[i * ncols + j]$. Go uses a similar approach but bundles the dimensions together with the allocated memory in a custom type. In contrast, Chapel and DASH support a concise and elegant syntax for the allocation and direct element-wise access of their built-in multidimensional arrays. In the case of DASH, the distributed multidimensional array is realized as a C++ template class that follows the container concept of the standard template library (STL) [11].

Work Sharing In all benchmarks, work has to be distributed between multiple processes or threads, for example when computing the random values in *randmat* in parallel. *randmat* requires that the result be independent of the degree of parallelism used and all implementations solve this issue by using a separate deterministic seed value for each row of the matrix. A whole matrix row is the unit of work that is distributed among the processes or threads. The same strategy is also used for *outer* and *product*.

Cilk uses `cilk_for` to automatically partition the matrix rows and TBB uses C++ template mechanisms to achieve a similar goal. Go does not offer built-in constructs for simple work sharing and the functionality has to be laboriously created manually using goroutines, channels, and ranges.

In Chapel this type of work distribution can simply be expressed as a parallel loop (`forall`).

In DASH, the work distribution follows the data distribution. I.e., each unit is responsible for computing on the data that is locally resident, the *owner computes* model. Each unit determines its locally stored portion of the matrix (guaranteed to be a set of rows by the data distribution pattern used) and works on it independently.

Global Max Reduction In *thresh*, the largest matrix entry has to be determined to initialize other data structures to their correct size. The Go reference implementation doesn't actually perform this step and instead just uses a default size of 100, Go is thus not discussed further in this section.

In Cilk a `reducer_max` object together with a parallel loop over the rows is employed to find the maximum. Local maximal values are computed in parallel and then the global maximum is found using the reducer object. In TBB a similar construct is used (`tbb::parallel_reduce`). In these approaches finding the local maximum and computing the global maximum are separate steps that require a considerable amount of code (several 10s of lines of code).

Chapel again has the most concise syntax of all approaches, the maximum value is found simply by `nmax = max reduce matrix`. The code in the DASH solution is nearly as compact, by using the `max_element()` algorithm to find the maximum. Instead of specifying the matrix object directly, in DASH we have to couple the algorithm and the container using the iterator interface by passing `mat.begin()` and `mat.end()` to denote the range of elements to be processed.

Parallel Histogramming *thresh* requires the computation of a global cumulative histogram over an integer matrix. Thus, for each integer value $0, \ldots, \texttt{nmax} - 1$ we need to determine the number of occurrences in the given matrix in parallel. The strategy used by all implementations is to compute one or multiple histograms by each thread in parallel and to later combine them into a single global histogram.

In DASH we use a distributed array to compute the histogram. First, each unit computes the histogram for the locally stored data, by simply iterating over all local matrix elements and updating the local histogram (`histo.local`). Then `dash::transform` is used to combine the local histograms into a single global histogram located at unit 0. `dash::transform` is modeled after `std::transform`, a mutating sequence algorithm. Like the STL variant, the algorithm works with two input ranges that are combined using the specified operation into an output range.

Parallel Sorting *winnow* requires the sorting of 3-tuples using a custom comparison operator. Cilk and TBB use a parallel shared memory sort. Go and Chapel call their respective internal sort implementations (quicksort in the case of Chapel) which appears to be unparallelized. DASH can take advantage of a parallel and distributed sort implementation based on histogram sort cf. 3.2.

4.1.4 Evaluation

We first compare the productivity of DASH compared to the other parallel programming approaches before analyzing the performance differences.

Productivity We evaluate programmer productivity by analyzing source code complexity. Table 1 shows the lines of code (LOC) used in the implementation for each kernel, counting only lines that are not empty or comments. Of course, LOC is a crude approximation for source code complexity but few other metrics are universally accepted or available for different programming languages. LOC at least gives a rough idea for source code size, and, as a proxy, development time, likelihood for programming errors and productivity. The overall winners in the productivity category are Chapel and Cilk, which achieve the smallest source code size for three benchmark kernels. For most kernels, DASH also achieves a remarkably small source code size considering that the same source code can run on shared memory as well as on distributed memory machines.

Performance As the hardware platform for our experiments we have used one or more nodes of SuperMUC Phase 2 (SuperMUC-HW) with Intel Xeon E5-2697 (Haswell) CPUs with 2.6 GHz, 28 cores and 64 GB of main memory per node.

Single Node Comparison We first investigate the performance differences between DASH and the four established parallel programming models on a single Haswell node. We select a problem size of nrows = ncols = 30,000 because this is the largest size that successfully ran with all programming approaches.

Table 2 shows the absolute performance (runtime in seconds) and relative runtime (compared to DASH) when using all 28 cores of a single node. Evidently, DASH is the fastest implementation, followed by Cilk and TBB. Chapel and especially Go can not deliver competitive performance in this setting.

Analyzing the scaling behavior in more detail (not shown graphically due to space restrictions) by increasing the number of cores used on the system from 1 to 28 reveals a few interesting trends. For outer and randmat, DASH, Cilk and TBB

Table 1 Lines-of-code (LOC) measure for each kernel and programming approach, counting non-empty and non-comment lines only

	DASH	Go	Chapel	TBB	Cilk
Randmat	18	29	14	15	12
Thresh	31	63	30	56	52
Winnow	67	94	31	74	78
Outer	23	38	15	19	15
Product	19	27	11	14	10

Table 2 Performance comparison for each kernel and programming approach using all cores on one node of SuperMUC-HW

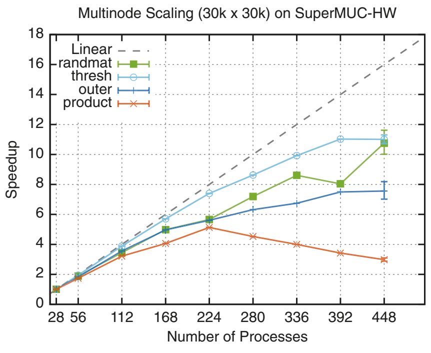
	Absolute runtime (sec.)					Relative runtime				
	DASH	Go	Chapel	TBB	Cilk	DASH	Go	Chapel	TBB	Cilk
Randmat	0.12	5.84	0.36	0.15	0.19	1.00	48.69	3.03	1.26	1.58
Thresh	0.20	0.64	0.53	0.41	0.40	1.00	3.19	2.64	2.06	2.00
Winnow	2.99	366.13	256.40	9.45	4.28	1.00	122.45	85.75	3.16	1.43
Outer	0.25	1.18	0.39	0.27	0.31	1.00	4.70	1.56	1.06	1.24
Product	0.06	0.46	0.19	0.12	0.13	1.00	7.66	3.15	2.01	2.16

behave nearly identical in terms of absolute performance and scaling behavior. The small performance advantage of DASH can be attributed to better NUMA locality of DASH, where all work is done on process-local data. For thresh and winnow DASH can take advantage of optimized parallel algorithms (for global max reduction and sorting, respectively) whereas these operations are performed sequentially in some of the other approaches. For product the DASH implementation takes advantage of a local copy optimization to improve data locality. The scaling study reveals that this optimization at first costs performance but pays off at larger core counts.

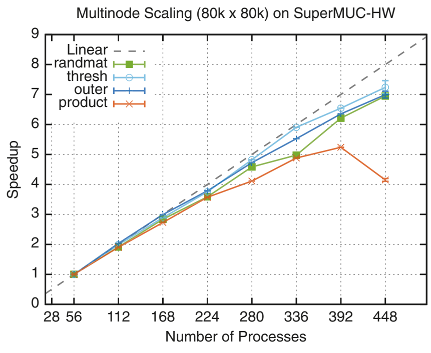
Multinode Scaling We next investigate the scaling of the DASH implementation on up to 16 nodes (448 total cores) of SuperMUC-HW. None of the other approaches can be compared with DASH in this scenario. Cilk and TBB are naturally restricted to shared memory systems by their threading-based nature. Go realizes the CSP (communicating sequential processes) model that would, in principle, allow for a distributed memory implementation but since data sharing via pointers is allowed, Go is also restricted to a single shared memory node. Finally, Chapel targets both shared and distributed memory systems, but the implementation of the Cowichan problems available in this study is not prepared to be used with multiple locales and cannot make use of multiple nodes (it lacks the `dmapped` specification for data distribution).

The scaling results are shown in Fig. 20 for two data set sizes. In Fig. 20 (left) we show the speedup relative to one node for a small problem size (nrows = ncols = 30,000) and in Fig. 20 (right) we show the speedup of a larger problem size (nrows = ncols = 80,000) relative to two nodes, since this problem is too big to fit into the memory of a single node.

Evidently for the smaller problem size, the benchmark implementations reach their scaling limit at about 10 nodes, whereas the larger problem sizes manage to scale well even to 16 nodes, with the exception of the product benchmark which shows the worst scaling behavior. This behavior can be explained by the relatively large communication requirement of the product benchmark kernel.



(a)



(b)

Fig. 20 Scaling behavior of the Cowichan benchmarks with up to 16 nodes on SuperMUC-HW. (**a**) Multinode Scaling, $30 \times 30\,\text{k}$ Matrix. (**b**) Multinode Scaling: $80k \times 80k$ Matrix

4.1.5 Summary

In this section we have evaluated DASH, a new realization of the PGAS approach in the form of a C++ template library by comparing our implementation of the Cowichan problems with those developed by expert programmers in Cilk, TBB, Chapel, and Go. We were able to show that DASH achieves both remarkable performance and productivity that is comparable with established shared memory programming approaches. DASH is also the only approach in our study where the same source code can be used both on shared memory systems and on scalable distributed memory systems. This step, from shared memory to distributed memory systems is often the most difficult for parallel programmers because it frequently goes hand in hand with a re-structuring of the entire data distribution layout of the application. With DASH the same application can seamlessly scale from a single shared memory node to multiple interconnecting nodes.

4.2 Task-Based Application Study: LULESH

The Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) is part of the Department of Energy’s Coral proxy application benchmark suite [19]. The domain typically scales with the number of processes and is divided into a grid of nodes, elements, and regions of elements. The distribution and data exchange follows a 27-point stencil with three communication steps per timestep.

The reference implementation of LULESH uses MPI send/recv communication to facilitate the boundary exchange between neighboring processes and OpenMP worksharing constructs are used for shared memory parallelization. Instead, we

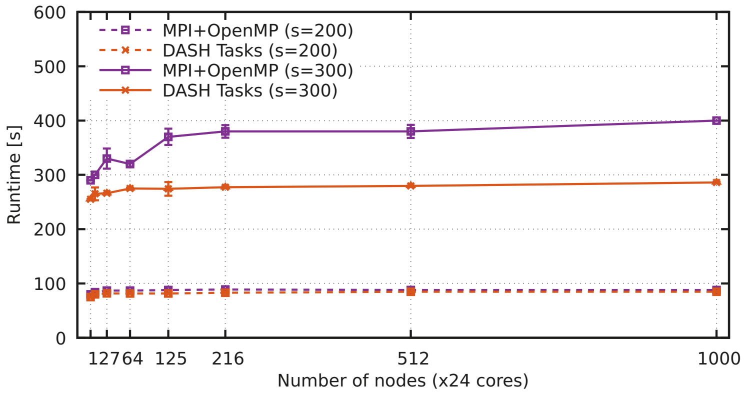


Fig. 21 Lulesh reference implementation and Lulesh using DASH tasks running on a Cray XC40

iteratively ported LULESH to use DASH distributed data structures for neighbor communication and DASH tasks for shared memory parallelization, with the ability to partially overlap communication and computation.

The first step was to adapt the communication to use DASH distributed data structures instead of MPI [12]. In order to gradually introduce tasks, we started by porting the OpenMP worksharing loop constructs with to use the DASH task-loop construct discussed in Sect. 2.1.3. In a second step, the iteration chunks of the taskloops were connected through dependencies, allowing a breadth-first scheduler to execute tasks from different task-loop statements concurrently. In a last step, a set of high-level tasks has been introduced that encapsulate the task-loop statements and coordinate the computation and communication tasks.

The resulting performance at scale on a Cray XC40 is shown in Fig. 21. For larger problem sizes ($s = 300^3$ elements per node), the speedup at scale of the DASH port over the reference implementation is about 25%. For smaller problem sizes ($s = 200^3$), the speedup is significantly smaller at about 5%. We believe that further optimizations in the tasking scheduler may yield improvements even for smaller problem sizes.

5 Outlook and Conclusion

We have presented an overview of our parallel programming approach DASH, focusing on recent activities in the areas of support for task-based execution, dynamic locality, parallel algorithms, and smart data structures. Our results show that DASH offers a productive programming environment that also allows programmers to write highly efficient and scalable programs with performance on-par or exceeding solutions relying on established programming systems.

Work on DASH is not finished. The hardware landscape in high performance computing is getting still more complex, while the application areas are getting more diverse. Heterogeneous compute resources are common, nonvolatile memories are making their appearance and domain specific architectures are on the horizon. These and other challenges must be addressed by DASH to be a viable parallel programming approach for many users.

The challenge of utilizing heterogeneous computing resources, primarily in the form of graphics processing units (GPUs) used in high performance computing systems, is addressed in a project building on DASH funded by the German Federal Ministry of Education and Research (BMBF) called MEPHISTO. We close our report on DASH with a short discussion of MEPHISTO.

5.1 MEPHISTO

The PGAS model simplifies the implementation of programs for distributed memory plattforms, but data locality plays an important role for performance critical applications. The *owner-computes* paradigm aims to maximize the performance by minimizing the intra-node data movement. This focus on locality also encourages the usage of shared memory parallelism. During the DASH project the partners already experimented with shared memory parallelism and how it can be integrated into the DASH ecosystem. OpenMP was integrated into suitable algorithms, experiments with Intel’s Thread Building Blocks as well as conventional POSIX-thread-based parallelism were conducted. These, however, had to be fixed for a given algorithm: a user had no control over the acceleration and the library authors had to find sensible configurations (e.g. level of parallelism, striding, scheduling). Giving users of the DASH library more possibilities and flexibility is a focus of the MEPHISTO project.

The MEPHISTO project partly builds on the work that has been done in DASH. One of its goal is to integrate abstractions for better data locality and heterogeneous programming with DASH. Within the scope of MEPHISTO two further projects are being integrated with DASH:

- Abstraction Library for Parallel Kernel Acceleration (ALPAKA) [22]
- Low Level Abstraction of Memory Access[6] (LLAMA)

ALPAKA is a library that adds abstractions for node-level parallelism for C++ programs. Once a kernel is written with ALPAKA, it can be easily ported to different accelerators and setups. For example a kernel written with ALPAKA can be executed in a multi-threaded CPU environment as well as on a GPU, for example using the CUDA backend. ALPAKA provides optimized code for each accelerator that can be further customized by developers. To switch from one back end to

[6]https://github.com/mephisto-hpc/llama.

another requires just the change of one type definition in the code so that the code is portable.

Integrating ALPAKA and DASH brings flexibility for node-level parallelism within a PGAS environment: switching an algorithm from a multi threaded CPU implementation to an accelerator now only requires passing one more parameter to the function call. It also gives the developer an interface to control *how* and *where* the computation should happen. The kernels of algorithms like `dash::transform_reduce` are currently extended to work with external *executors* like ALPAKA. Listing 5 shows how executors are used to offload computation using ALPAKA. Note that the interface allows offloading to other implementations as well. In the future DASH will support any accelerator that implements a simple standard interface for node-level parallelism.

```
policy.executor().bulk_twoway_execute(
  [=](size_t      block_index,
      size_t      element_index,
      T*          res,
      value_type* block_first) {
    res[block_index] = binary_op(
                         res[block_index],
                         unary_op(block_first[element_index])
    ↪ );
  },
  in_first, // a "shape"
  [&]() -> std::vector<std::future<T>>& {
    return results;
  },
  std::ignore); // shared state (unused)
```

Listing 5 Offloading using an executor inside `dash::transform_reduce`. The executor can be provided by MEPHISTO

LLAMA on the other hand focuses solely on the format of data in memory. DASH already provides a flexible *Pattern Concept* to define the data placement for a distributed container. However, LLAMA gives developers finer grained control over the data layout. DASH's patterns map elements of a container to locations in memory, but the layout of the elements itself is fixed. With LLAMA, developers can specify the data layout with a C++ Domain Specific Language (DSL) to fit the application's needs. A typical example is a conversion from Structure of Arrays (SoA) to Array of Structures (AoS) and vice versa. But also more complex transformations like projections are being evaluated.

Additionally to the integration of LLAMA, a more flexible, hierarchical and context-sensitive pattern concept is being evaluated. Since the current patterns map elements to memory locations in terms of units (i.e., MPI processes), using other sources for parallelism can be a complex task. Mapping elements to (possibly multiple) accelerators was not easily possible. Local patterns extend the existing patterns. By matching elements with *entities* (e.g. a GPU), the node-local data may be assigned to other compute units beside processes.

Both projects are currently evaluated in the context of DASH to explore how the PGAS programming model can be used more flexibly and efficiently.

Acknowledgments The authors would like to thank the many dedicated and talented individuals who contributed to the success of the DASH project. DASH started out as "just an idea" that was turned into reality with the help of Benedikt Bleimhofer, Daniel Rubio Bonilla, Daniel Diefenthaler, Stefan Effenberger, Colin Glass, Kamran Idrees, Benedikt Lehmann, Michael Maier, Matthias Maiterth, Yousri Mhedheb, Felix Mössbauer, Sebastian Oeste, Bernhard Saumweber, Jie Tao, Bert Wesarg, Huan Zhou, Lei Zhou.

We would also like to thank the German research foundation (DFG) for the funding received through the SPPEXA priority programme and initiators and managers of SPPEXA for their foresight and level-headed approach.

References

1. Agullo, E., Aumage, O., Faverge, M., Furmento, N., Pruvost, F., Sergent, M., Thibault, S.P.: Achieving high performance on supercomputers with a sequential task-based programming model. IEEE Trans. Parallel Distrib. Syst. (2018). https://doi.org/10.1109/TPDS.2017.2766064
2. Axtmann, M., Bingmann, T., Sanders, P., Schulz, C.: Practical massively parallel sorting. In: Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures, pp. 13–23. ACM, New York (2015)
3. Ayguade, E., Copty, N., Duran, A., Hoeflinger, J., Lin, Y., Massaioli, F., Teruel, X., Unnikrishnan, P., Zhang, G.: The design of OpenMP tasks. IEEE Trans. Parallel Distrib. Syst. **20**, 404–418 (2009). https://doi.org/10.1109/TPDS.2008.105
4. Bauer, M., Treichler, S., Slaughter, E., Aiken, A.: Legion: expressing locality and independence with logical regions. In: 2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1–11 (2012). https://doi.org/10.1109/SC.2012.71
5. Blelloch, G.E., Leiserson, C.E., Maggs, B.M., Plaxton, C.G., Smith, S.J., Zagha, M.: A comparison of sorting algorithms for the connection machine cm-2. In: Proceedings of the Third Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 3–16. ACM, New York (1991)
6. Blumofe, R.D., Joerg, C.F., Kuszmaul, B.C., Leiserson, C.E., Randall, K.H., Zhou, Y.: Cilk: An Efficient Multithreaded Runtime System, vol. 30. ACM, New York (1995)
7. Bosilca, G., Bouteiller, A., Danalis, A., Herault, T., Lemariner, P., Dongarra, J.: Dague: a generic distributed dag engine for high performance computing. In: Proceedings of the Workshops of the 25th IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2011 Workshops), pp. 1151–1158 (2011)
8. Chamberlain, B.L., Callahan, D., Zima, H.P.: Parallel programmability and the Chapel language. Int. J. High Perform. Comput. Appl. **21**, 291–312 (2007)
9. Charles, P., Grothoff, C., Saraswat, V., Donawa, C., Kielstra, A., Ebcioglu, K., Von Praun, C., Sarkar, V.: X10: an object-oriented approach to non-uniform cluster computing. ACM SIGPLAN Not. **40**, 519–538 (2005)
10. Donovan, A.A., Kernighan, B.W.: The Go Programming Language, 1st edn. Addison-Wesley Professional, Boston (2015)
11. Fuchs, T., Fürlinger, K.: A multi-dimensional distributed array abstraction for PGAS. In: Proceedings of the 18th IEEE International Conference on High Performance Computing and Communications (HPCC 2016), Sydney, pp. 1061–1068 (2016). https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0150

12. Fürlinger, K., Fuchs, T., Kowalewski, R.: DASH: a C++ PGAS library for distributed data structures and parallel algorithms. In: Proceedings of the 18th IEEE International Conference on High Performance Computing and Communications (HPCC 2016), Sydney, pp. 983–990 (2016). https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0140
13. Fürlinger, K., Kowalewski, R., Fuchs, T., Lehmann, B.: Investigating the performance and productivity of DASH using the cowichan problems. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region Workshops, Tokyo (2018). https://doi.org/10.1145/3176364.3176366
14. Grossman, M., Kumar, V., Budimlić, Z., Sarkar, V.: Integrating asynchronous task parallelism with OpenSHMEM. In: Workshop on OpenSHMEM and Related Technologies, pp. 3–17. Springer, Berlin (2016)
15. Harsh, V., Kalé, L.V., Solomonik, E.: Histogram sort with sampling. CoRR abs/1803.01237 (2018). http://arxiv.org/abs/1803.01237
16. Helman, D.R., JáJá, J., Bader, D.A.: A new deterministic parallel sorting algorithm with an experimental evaluation. ACM J. Exp. Algorithmics **3**, 4 (1998)
17. Kaiser, H., Brodowicz, M., Sterling, T.: Parallex an advanced parallel execution model for scaling-impaired applications. In: 2009 International Conference on Parallel Processing Workshops (2009). https://doi.org/10.1109/ICPPW.2009.14
18. Kaiser, H., Heller, T., Adelstein-Lelbach, B., Serio, A., Fey, D.: HPX: a task based programming model in a global address space. In: Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models, PGAS '14. ACM, New York (2014). http://doi.acm.org/10.1145/2676870.2676883
19. Karlin, I., Keasler, J., Neely, R.: Lulesh 2.0 updates and changes. Technical Report. LLNL-TR-641973 (2013)
20. Kowalewski, R., Jungblut, P., Fürlinger, K.: Engineering a distributed histogram sort. In: 2019 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, Piscataway (2019)
21. Kumar, V., Zheng, Y., Cavé, V., Budimlić, Z., Sarkar, V.: HabaneroUPC++: a compiler-free PGAS library. In: Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models, PGAS '14. ACM, New York (2014). https://doi.org/10.1145/2676870.2676879
22. Matthes, A., Widera, R., Zenker, E., Worpitz, B., Huebl, A., Bussmann, M.: Tuning and optimization for a variety of many-core architectures without changing a single line of implementation code using the alpaka library (2017). https://arxiv.org/abs/1706.10086
23. Nanz, S., West, S., Da Silveira, K.S., Meyer, B.: Benchmarking usability and performance of multicore languages. In: 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 183–192. IEEE, Piscataway (2013)
24. OpenMP Architecture Review Board: OpenMP Application Programming Interface, Version 5.0 (2018). https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf
25. Reinders, J.: Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism. O'Reilly & Associates, Sebastopol (2007)
26. Robison, A.D.: Composable parallel patterns with intel cilk plus. Comput. Sci. Eng. **15**, 66–71 (2013). https://doi.org/10.1109/MCSE.2013.21
27. Saraswat, V., Almasi, G., Bikshandi, G., Cascaval, C., Cunningham, D., Grove, D., Kodali, S., Peshansky, I., Tardieu, O.: The asynchronous partitioned global address space model. In: The First Workshop on Advances in Message Passing, pp. 1–8 (2010)
28. Saukas, E., Song, S.: A note on parallel selection on coarse-grained multicomputers. Algorithmica **24**(3-4), 371–380 (1999)
29. Shirako, J., Peixotto, D.M., Sarkar, V., Scherer, W.N.: Phasers: a unified deadlock-free construct for collective and point-to-point synchronization. In: Proceedings of the 22nd Annual International Conference on Supercomputing, ICS '08. ACM, New York (2008). https://doi.org/10.1145/1375527.1375568
30. Slaughter, E., Lee, W., Treichler, S., Bauer, M., Aiken, A.: Regent: a high-productivity programming language for HPC with logical regions. In: SC15: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–12 (2015). https://doi.org/10.1145/2807591.2807629

31. Sundar, H., Malhotra, D., Biros, G.: Hyksort: a new variant of hypercube quicksort on distributed memory architectures. In: Proceedings of the 27th international ACM conference on International Conference on Supercomputing, pp. 293–302. ACM, New York (2013)
32. Tejedor, E., Farreras, M., Grove, D., Badia, R.M., Almasi, G., Labarta, J.: A high-productivity task-based programming model for clusters. Concurr. Comp. Pract. Exp. **24**, 2421–2448 (2012). https://doi.org/10.1002/cpe.2831
33. Thakur, R., Rabenseifner, R., Gropp, W.: Optimization of collective communication operations in MPICH. Int. J. High Perform. Comput. Appl. **19**(1), 49–66 (2005). https://doi.org/10.1177/1094342005051521
34. Tillenius, M.: SuperGlue: a shared memory framework using data versioning for dependency-aware task-based parallelization. SIAM J. Sci. Comput. **37**, C617–C642 (2015). http://epubs.siam.org/doi/10.1137/140989716
35. Tsugane, K., Lee, J., Murai, H., Sato, M.: Multi-tasking execution in pgas language xcalablemp and communication optimization on many-core clusters. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region. ACM, New York (2018). https://doi.org/10.1145/3149457.3154482
36. Wilson, G.V., Irvin, R.B.: Assessing and comparing the usability of parallel programming systems. University of Toronto. Computer Systems Research Institute (1995)

ESSEX: Equipping Sparse Solvers For Exascale

Christie L. Alappat, Andreas Alvermann, Achim Basermann, Holger Fehske, Yasunori Futamura, Martin Galgon, Georg Hager, Sarah Huber, Akira Imakura, Masatoshi Kawai, Moritz Kreutzer, Bruno Lang, Kengo Nakajima, Melven Röhrig-Zöllner, Tetsuya Sakurai, Faisal Shahzad, Jonas Thies, and Gerhard Wellein

Abstract The ESSEX project has investigated programming concepts, data structures, and numerical algorithms for scalable, efficient, and robust sparse eigenvalue solvers on future heterogeneous exascale systems. Starting without the burden of legacy code, a holistic performance engineering process could be deployed across the traditional software layers to identify efficient implementations and guide sustainable software development. At the basic building blocks level, a flexible MPI+X programming approach was implemented together with a new sparse data structure (SELL-C-σ) to support heterogeneous architectures by design. Furthermore, ESSEX focused on hardware-efficient kernels for all relevant architectures and efficient data structures for block vector formulations of the eigensolvers. The algorithm layer addressed standard, generalized, and nonlinear eigenvalue problems and provided some widely usable solver implementations including a block Jacobi–Davidson algorithm, contour-based integration schemes, and filter polynomial approaches. Adding to the highly efficient kernel implementations, algorithmic advances such as adaptive precision, optimized filtering coefficients, and preconditioning have further improved time to solution. These developments

C. L. Alappat · G. Hager · M. Kreutzer · F. Shahzad · G. Wellein (✉)
Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany

M. Röhrig-Zöllner · J. Thies · A. Basermann
German Aerospace Center, Cologne, Germany

M. Galgon · S. Huber · B. Lang
Bergische Universität Wuppertal, Wuppertal, Germany

A. Alvermann · H. Fehske
Universität Greifswald, Greifswald, Germany

M. Kawai · K. Nakajima
University of Tokyo, Tokyo, Japan

Y. Futamura · A. Imakura · T. Sakurai
University of Tsukuba, Tsukuba, Japan

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_7

were guided by quantum physics applications, especially from the field of topological insulator- or graphene-based systems. For these, ScaMaC, a scalable matrix generation framework for a broad set of quantum physics problems, was developed. As the central software core of ESSEX, the PHIST library for sparse systems of linear equations and eigenvalue problems has been established. It abstracts algorithmic developments from low-level optimization. Finally, central ESSEX software components and solvers have demonstrated scalability and hardware efficiency on up to 256 K cores using million-way process/thread-level parallelism.

1 Introduction

The efficient solution of linear systems or eigenvalue problems involving large sparse matrices has been an active research field in parallel and high performance computing for many decades. Software packages like Trilinos [33] or PETSc [9] have been developed to great maturity, and algorithmic improvements were accompanied by advances in programming abstractions addressing, e.g., node-level heterogeneity (cf. Kokkos [19]). Completely new developments such as Ginkgo[1] are rare and do not focus on large-scale applications or node-level efficiency.

Despite projections from the late 2000s, hardware architectures have not developed away from traditional clustered multicore systems. However, a clear trend of increased node-level parallelism and heterogeneity has been observed. Although several new architectures entered the field (and some vanished again), the basic concepts of core-level code execution and data parallelism have not changed. This is why the MPI+X concept is still a viable response to the challenge of hardware diversity.

Performance analysis of highly parallel code typically concentrated on scalability, but provably optimal node-level performance was rarely an issue. Moreover, strong abstraction boundaries between linear algebra building blocks, solvers, and applications made it hard to get a holistic view on a minimization of time to solution, encompassing optimizations in the algorithmic and implementation dimensions.

In this setting, the ESSEX project took the opportunity to start from a clean slate, deliberately breaking said abstraction boundaries to investigate performance bottlenecks together with algorithmic improvements from the core to the highly parallel level. Driven by the targeted application fields, bespoke solutions were developed for selected algorithms and applications. The experience gained in the development process will lead the way towards more generic approaches rather than compete with established libraries in terms of generality. The overarching motif was a consistent performance engineering process that coordinated all performance-relevant activities across the different software layers [1, 3, 4, 20, 52–54, 56, 62, 64].

[1]https://github.com/ginkgo-project/ginkgo.

Consequently, the ESSEX parallel building blocks layer implemented in the GHOST library [55] supports MPI+X, with X being a combination of node-level programming models able to fully exploit hardware heterogeneity, functional parallelism, and data parallelism. Despite fluctuations in hardware architectures and new programming models hitting the market every year, OpenMP or CUDA is still the most promising and probably most sustainable choice for X, and ESSEX-II adhered to it. In addition, engineering highly specialized kernels including sparse-matrix multiple-vector operations and appropriate data structures for all relevant compute architectures provided the foundation for hardware- and energy-efficient large-scale computations.

Building on these high-performance building blocks, one focus of the algorithm layer was put on the block formulation of Jacobi–Davidson [64] and filter diagonalization [56] methods, the hardware efficiency of preconditioners [46–48], and the development of hardware-aware coloring schemes [1]. In terms of scalability, the project has investigated new contour-based integration eigensolvers [23, 24] that can exploit additional parallelism layers beyond the usual data parallelism. The solvers developed in ESSEX can tackle standard, generalized, and nonlinear eigenvalue problems and may also be used to extract large bulks of extremal and inner eigenvalues.

The applications layer applies the algorithms and building blocks to deliver scalable solutions for topical quantum materials like graphene, topological insulators, or superconductors, and nonlinear dynamical systems like reaction-diffusion systems. A key issue for large-scale simulations is the scalable (in terms of size and parallelism) generation of the sparse matrix representing the model Hamiltonian. Our matrix generation framework ScaMaC can be integrated into application code to allow the on-the-fly, in-place construction of the sparse matrix. Beyond the ESSEX application fields, matrices from many other relevant areas can be produced by ScaMaC.

The PHIST library [78] is the sustainable outcome of the performance-centric efforts in ESSEX. It is built on a rigorous software and performance engineering process, comprises diverse solver components, and supports multiple backends (e.g., Trilinos, PETSc, ESSEX kernels). It also interfaces to multiple languages such as C, C++, Fortran 2003, and Python. The CRAFT library [73] provides user-friendly access to fault tolerance via checkpoint/restart and automatic recovery for iterative codes using standard C++.

Scalabilty, performance, and portability have been tested on three top-10 supercomputers covering the full range of architecures available during the ESSEX project time frame: Piz Daint[2] (heterogeneous CPU-GPU), OakForest-PACS[3] (many-core), and SuperMUC-NG[4] (standard multi-core).

[2]https://www.cscs.ch/computers/piz-daint/.

[3]https://www.cc.u-tokyo.ac.jp/en/supercomputer/ofp/service/.

[4]https://doku.lrz.de/display/PUBLIC/SuperMUC-NG.

This review focuses on important developments in ESSEX-II. After presenting a brief overview of the most relevant achievements in the first project phase ESSEX-I in Sects. 2 and 3 details algorithmic developments in ESSEX-II, notably with respect to preconditioners and projection-based methods for obtaining inner eigenvalues. Moreover, we present the RACE (Recursive Algebraic Coloring Engine) method, which delivers hardware-efficient graph colorings for parallelization of algorithms and kernels with data dependencies. In Sect. 4 we showcase performance and parallel efficiency numbers for library components developed in ESSEX-II that are of paramount importance for the application work packages: GPGPU-based tall and skinny matrix-matrix multiplication and the computation of inner eigenvalues using polynomial filter techniques. Section 5 describes the software packages that were developed to a usable and sustainable state, together with their areas of applicability. In Sect. 6 we show application results from the important areas of quantum physics and nonlinear dynamical systems. Finally, in Sect. 7 we highlight the collaborations sparked and supported by SPPEXA through the ESSEX-II project.

2 Summary of the ESSEX-I Software Structure

The Exascale-enabled Sparse Solver Repository (ESSR) was developed along the requirements of the algorithms and applications under investigation in ESSEX. It was not intended as a full-fledged replacement of existing libraries like Trilinos[5] [33], but rather as a toolbox that can supply developers with blueprints as a starting point for their own developments. In ESSEX-I, the foundations for a sustainable software framework were laid. See Sect. 5 for developments in ESSEX-II.

The initial version of the ESSR [77] comprised four components:

- GHOST (General, Hybrid and Optimized Sparse Toolkit) [55], a library of basic sparse and dense linear algebra building blocks that are not available in this form in other software packages. The development of GHOST was strictly guided by performance engineering techniques; implementations of standard kernels such as sparse matrix-vector multiplication (spMVM) and sparse matrix-multiple-vector multiplication (spMMVM) as well as tailor-made fused kernels, for instance those employed in the Kernel Polynomial Method (KPM) [81], were modeled using the roofline model. GHOST supports, by design, strongly heterogeneous environments using the MPI+X approach. See [51] for a comprehensive overview of GHOST and its building blocks.
- ESSEX-Physics, a collection of prototype implementations of polynomial eigensolvers such as the KPM and Chebyshev Filter Diagonalization (ChebFD). These were implemented on top of GHOST using tailored kernels and were shown to perform well on heterogeneous CPU-GPU systems [53].

[5]https://trilinos.org/.

- PHIST (Pipelined Hybrid-parallel Iterative Solver Toolkit), which comprises Jacobi–Davidson type eigensolvers and Krylov methods for linear systems. One important component is a test framework that allows for continuous integration (CI) throughout the development cycle. PHIST can not only use plain GHOST as its basic linear algebra layer; it is also equipped with fallback kernel implementations and adapters for the Trilinos and Anasazi libraries. A major achievement in the development of PHIST was an efficient block Jacobi–Davidson eigenvalue solver, which could be shown to have significant performance advantages over nonblocked versions when using optimized building blocks from GHOST [64].
- BEAST (Beyond fEAST), which implements innovative projection-based eigensolvers motivated by the contour integration-based FEAST method [23]. The ESSEX-I project has contributed to improving FEAST in two ways: by proposing techniques for solving or avoiding the linear systems that arise, and by improving robustness and performance of the algorithmic scheme.

A pivotal choice for any sparse algorithm implementation is the sparse matrix storage format. In order to avoid data conversion and the need to support multiple hardware-specific formats in a single code, we developed the SELL-C-σ format [52]. It shows competitive performance on the dominating processor architectures in HPC: standard multicore server CPUs with short-vector single instruction multiple data (SIMD) capabilities, general-purpose graphics processing units (GPGPUs), and many-core designs with rather weak cores such as the Intel Xeon Phi. SELL-C-σ circumvents the performance penalties of matrices with few nonzero entries per row on architectures on which SIMD vectorization is a key element for performance even with memory-bound workloads.

In order to convert a sparse matrix to SELL-C-σ, its rows are first sorted according to their respective numbers of nonzeros. This sorting is performed across row blocks of length σ. After that, the matrix is cut into row blocks of length C. Within each block, rows are padded with zeros to equal length and then stored in column-major order. See Fig. 1 for visualizations of SELL-C-σ with $C = 6$ and $\sigma \in \{1, 12, 24\}$. Incidentally, known and popular formats can be recovered as corner cases: SELL-1-1 is the well-known CSR storage format and SELL-N-1 is ELLPACK. The particular choice of C and σ influences the performance of the spMVM operation; optimal values are typically matrix- and hardware-dependent. However, in practice one can usually find parameters that yield good performance across architectures for a particular matrix. A roofline performance model was constructed in [52] that sets an upper limit for the spMVM performance for any combination of matrix and architecture. This way, "bad" performance is easily identified. SELL-C-σ was quickly adopted by the community and is in use, in pure or adapted form, in many performance-oriented projects [5, 6, 28, 60, 80].

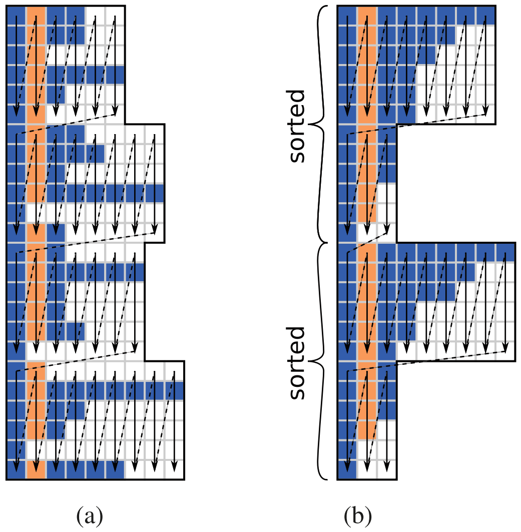


Fig. 1 Variants of the SELL-C-σ storage format. Arrows indicate the storage order of matrix values and column indices. Image from [52]. (**a**) SELL-6-1, $\beta = 0.51$. (**b**) SELL-6-12, $\beta = 0.66$. (**c**) SELL-6-24, $\beta = 0.84$

3 Algorithmic Developments

In this section we describe selected developments within ESSEX-II on the algorithmic level, in particular preconditioners for the solution of linear systems that occur in the eigensolvers, a versatile framework for computing inner eigenvalues, and a nonlinear eigensolver. We also cover a systematic comparison of contour-based methods. We close the section with the introduction of RACE, which is an algorithmic development for graph coloring guided by the constraints of hardware efficiency.

3.1 Preconditioners (ppOpen-SOL)

Two kinds of solvers have been developed: a preconditioner targeting the ill-conditioned large scale problems arising in the BEAST-C method (cf. Sect. 3.2) and a multigrid solver targeting problems arising from finite difference discretizations of partial differential equations (PDEs).

3.1.1 Regularization

The BEAST-C method leads to a large number of ill-conditioned linear systems with complex diagonal shifts [24]. Furthermore, in many of our quantum physics applications, the system matrices have small (and sometimes random) diagonal elements. In order to apply a classic incomplete Cholesky (IC) factorization preconditioner, we used two types of regularization to achieve robustness: a blocking technique (BIC) and an additional diagonal shift [47]. Using this approach, we solved a set of 120 prototypical linear systems from this context (e.g., BEAST-C applied to quantum physics applications). Due to the complex shift, the system matrix is symmetric but not Hermitian. Hence we use an adaptation of the Conjugate Gradient (CG) method for complex symmetric matrices called COCG (conjugate orthogonal conjugate gradient [79]).

The blocking technique is a well-known approach for improving the convergence rate. In this study, we apply the technique not only for better convergence but also for more robustness. The diagonal entries in the target equations are small. By applying the blocking technique, the diagonal blocks to be inverted include larger off-diagonal entries.

The diagonal shifting is a direct measure for transforming the ill-conditioned matrices to be more diagonally dominant before performing the incomplete factorization. On the other hand, this may deteriorate the convergence of the overall method. We therefore investigate the best value for the diagonal shifting for our applications.

Figure 2 shows the effect of the regularized IC preconditioner with the COCG method. By using the diagonal shifted block IC-COCG (BIC-COCG), we solve all target linear systems.

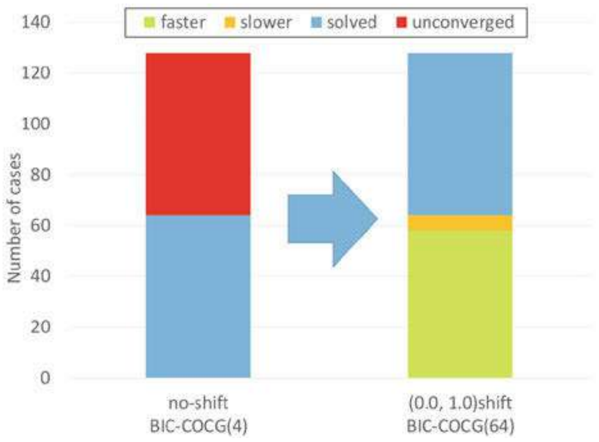


Fig. 2 Effect of the regularized IC preconditioner with the COCG method. By using the diagonal shifted block IC-COCG (BIC-COCG), we can solve all test problems from our benchmark set

3.1.2 Hierarchical Parallel Reordering

In this section, we present scalability results for the BIC preconditioner parallelized by a hierarchical parallel graph coloring algorithm. This approach yields an almost constant convergence rate with respect to the number of compute nodes, and good parallel performance.

Node-wise multi-coloring (with domain decomposition between nodes) is widely used for parallelizing IC preconditioners on clusters of shared memory CPUs. Such "localized" multi-coloring leads to a loss of robustness of the regularized IC-COCG method, and the convergence rate decreases at high levels of parallelism. To solve this problem, we parallelize the block IC preconditioner for the hybrid-parallel cluster system. In addition, we proposed the hierarchical parallelization for the multi-coloring algorithms [46]. This versatile scheme allows us to parallelize almost any multi-coloring algorithm.

Figure 3 shows the number of iterations and computational time of the BIC-COCG method on the Oakleaf-FX cluster, using up to 4,800 nodes. The benchmark matrix is the Hamiltonian of a graphene sheet simulation with more than 500 million linear equations, for which interior eigenvalues are of interest [24]. Hierarchical parallelization yields almost constant convergence with respect to the number of nodes. The computational time with 4,600 nodes is 30 times smaller than with 128 nodes, amounting to a parallel efficiency of 83.5% if the 128-node case is taken as the baseline.

3.1.3 Multiplicative Schwarz-Type Block Red-Black Gauß–Seidel Smoother

Multigrid methods are among the most useful preconditioners for elliptic PDEs. In [45] we proposed a multiplicative Schwarz block red/black Gauß–Seidel (MS-

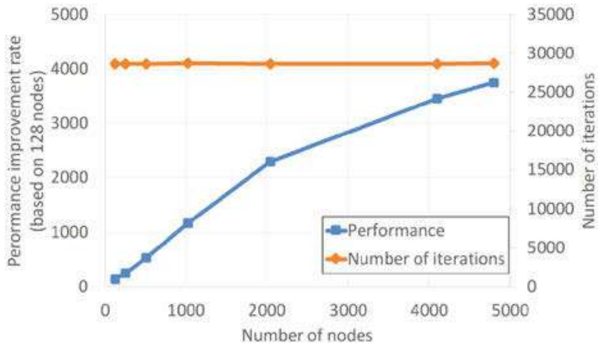
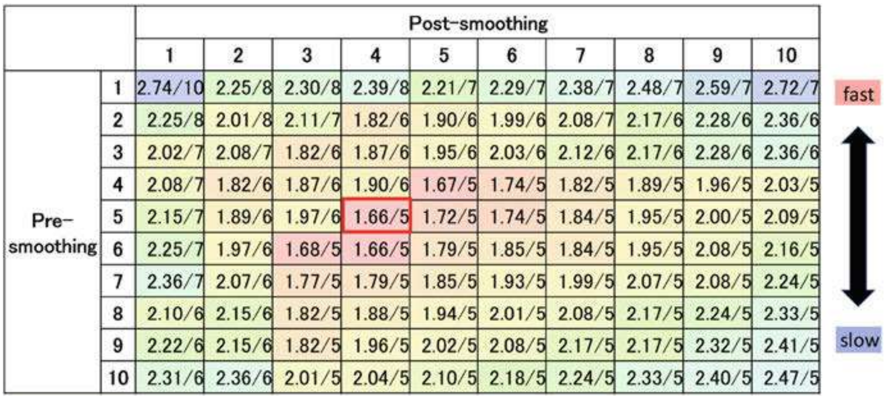


Fig. 3 Computational time and convergence of BIC-COCG for a graphene benchmark problem (strong scaling)



		Post-smoothing									
		1	2	3	4	5	6	7	8	9	10
Pre-smoothing	1	2.74/10	2.25/8	2.30/8	2.39/8	2.21/7	2.29/7	2.38/7	2.48/7	2.59/7	2.72/7
	2	2.25/8	2.01/8	2.11/7	1.82/6	1.90/6	1.99/6	2.08/7	2.17/6	2.28/6	2.36/6
	3	2.02/7	2.08/7	1.82/6	1.87/6	1.95/6	2.03/6	2.12/6	2.17/6	2.28/6	2.36/6
	4	2.08/7	1.82/6	1.87/6	1.90/6	1.67/5	1.74/5	1.82/5	1.89/5	1.96/5	2.03/5
	5	2.15/7	1.89/6	1.97/6	$\boxed{1.66/5}$	1.72/5	1.74/5	1.84/5	1.95/5	2.00/5	2.09/5
	6	2.25/7	1.97/6	1.68/5	1.66/5	1.79/5	1.85/5	1.84/5	1.95/5	2.08/5	2.16/5
	7	2.36/7	2.07/6	1.77/5	1.79/5	1.85/5	1.93/5	1.99/5	2.07/5	2.08/5	2.24/5
	8	2.10/6	2.15/6	1.82/5	1.88/5	1.94/5	2.01/5	2.08/5	2.17/5	2.24/5	2.33/5
	9	2.22/6	2.15/6	1.82/5	1.96/5	2.02/5	2.08/5	2.17/5	2.17/5	2.32/5	2.41/5
	10	2.31/6	2.36/6	2.01/5	2.04/5	2.10/5	2.18/5	2.24/5	2.33/5	2.40/5	2.47/5

Fig. 4 Computational time and number of iterations of a geometric multigrid solver with the MS-BRB-GS(α) smoother

BRB-GS) smoother for geometric multigrid methods. It is a modified version of the block red-black Gauß–Seidel (BRB-GS) smoother that improves convergence rate and data locality by applying multiple consecutive Gauß–Seidel sweeps on each block.

The unknowns are divided into blocks so that the amount of data for processing each block fits into the cache, and α Gauß–Seidel iterations are applied to the block per smoother step. The computational cost for the additional iterations is much lower than for the first iteration because of data locality.

Figure 4 shows the effect of the MS-BRB-GS(α) smoother on a single node of the ITO system (Intel Xeon Gold 6154 (Skylake-SP) Cluster at Kyushu University). By increasing the number of both pre- and post-smoothing steps, the number of iterations is decreased. In the best case, MS-BRB-GS is 1.64$\times$ faster than BRB-GS.

3.2 The BEAST Framework for Interior Definite Generalized Eigenproblems

The BEAST framework targets the solution of interior definite eigenproblems

$$AX = BX\Lambda \ ,$$

i.e., for finding all eigenvectors and eigenvalues of a definite matrix pair (A, B), with A and B Hermitian and B additionally positive definite, within a given interval $[\underline{\lambda}, \overline{\lambda}]$. The framework is based on the Rayleigh–Ritz subspace iteration procedure, in particular the spectral filtering approach: Arbitrary continuous portions of the spectrum may be selected for computation with appropriate filtering functions that are applied via an implicit approximate projector to compute a suitable subspace basis. Starting with an initial subspace Y, the following three main steps are repeated until a suitable convergence criterion is met:

```
Compute a subspace $U$ by approximately projecting $Y$
Rayleigh–Ritz extraction: solve the reduced eigenproblem $A_U V = B_U V \Lambda$,
    where $A_U = U^H AU$, $B_U = U^H BU$, and let $X = UV$
Obtain new $Y$ from $X$ or $U$
```

In the following we highlight some of BEAST's algorithmic features, skipping other topics such as locking converged eigenpairs, adjusting the dimension of the subspace, and others.

3.2.1 Projector Types

BEAST provides three variants of approximate projectors. First, polynomial approximation (BEAST-P) using Chebyshev polynomials, which only requires matrix vector multiplications but is restricted to standard eigenproblems. Second, Cauchy integral-based contour integration (BEAST-C), as in the FEAST method [63]. As a third method, an iterative implementation of the Sakurai–Sugiura method [65] is available (BEAST-M), which shares algorithmic similarities with FEAST. In the following we briefly elaborate on the algorithmic ideas.

- In BEAST-P, we have $U = p(A) \cdot Y$ with a polynomial $p(z) = \sum_{k=0}^{d} c_k T_k(z)$ of suitable degree d. Here, T_k denotes the kth Chebyshev polynomial,

$$T_0(z) \equiv 1, \quad T_1(z) = z, \quad T_k(z) = 2z \cdot T_{k-1}(z) - T_{k-2}(z),\ k \geq 2.$$

  Due to the use of the T_k, this method is also known as Chebyshev filter diagonalization.
  In addition to well-known methods for computing the coefficients c_k [18, 62], BEAST also provides the option of using new, improved coefficients [25]. Their computation depends on two parameters, μ and σ, and for suitable combinations of these, the filtering quality of the polynomial can be improved significantly; see Fig. 5, which shows the "gain," i.e., the reduction of the width of those λ values *outside* the search interval, for which a damping of corresponding eigenvectors by at least a factor 100 cannot be guaranteed. For some combinations (σ, μ), marked red in the picture, this "no guarantee" area can be reduced by a factor of more than 2, which in turn allows using lower-degree polynomials to achieve comparable overall convergence. A parallelized method for finding suitable parameter combinations and computing the c_k is included with BEAST.

- In BEAST-C, the exact projection

$$\frac{1}{2\pi \mathrm{i}} \int_{\Gamma} \mathrm{d}z \, (zB - A)^{-1} BY$$

(integration is over a contour Γ in the complex plane that encloses the eigenvalues $\lambda \in [\underline{\lambda}, \overline{\lambda}]$, but no others) is approximated using an N-point quadrature rule,

$$U = \sum_{j=1}^{N} \omega_j (z_j B - A)^{-1} BY,$$

leading to N linear systems, where the number of right-hand sides (RHS) corresponds to the dimension of the current subspace U (and Y).
- BEAST-M is also based on contour integration, but moments are used to reduce the number of RHS in the linear systems. Taking M moments, we have

$$U = [U_0, \ldots, U_{M-1}] \quad \text{with} \quad U_k = \sum_{j=1}^{N} \omega_j z_j^k (z_j B - A)^{-1} BY,$$

and thus an M times smaller number of RHS (dimension of Y) is sufficient to achieve the same dimension of U.

The linear systems in the contour-based schemes may be ill-conditioned if the integration points z_j are close to the spectrum (this happens, e.g., for narrow search intervals $[\underline{\lambda}, \overline{\lambda}]$); cf. also Sects. 3.1 and 3.4 for approaches to address this issue.

3.2.2 Flexibility, Adaptivity and Auto-Tuning

The BEAST framework provides flexibility at the algorithmic, parameter, and working precision levels, which we describe in detail in the following.

Algorithmic Level

The projector can be chosen from the three types described above, and the type may even be changed between iterations. In particular, an innovative subspace-iterative version of Sakurai–Sugiura methods (SSM) has been investigated for possible cost savings in the solution of linear systems via a limited subspace size and the overall reduction of number of right hand sides over iterations by using moments. Given, however, the potentially reduced convergence threshold with a constrained subspace size, we support switching from the multi-moment method, BEAST-M, to a single-moment method, BEAST-C. The efficiency, robustness, and

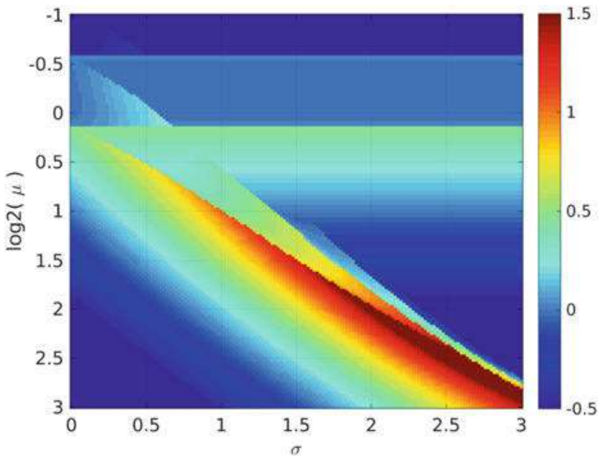


Fig. 5 Base-2 log of the "gain" from using modified coefficients with parameters (σ, μ) for the interval $[\underline{\lambda}, \overline{\lambda}] = [-0.584, -0.560]$ (matrix scaled such that $\mathrm{spec}(A) = [-1, +1]$) and degree $d = 1600$

accuracy of this approach in comparison with traditional SSM and FEAST has been explored [35].

We further studied this scheme along with another performance-based implementation of SSM, z-PARES [65, 67]. These investigations considered the scaling and computational cost of the libraries as well as heuristics for parameter choice, in particular with respect to the number of quadrature nodes. We observed that the scaling behavior improved when the number of quadrature nodes increased, as seen in Fig. 6. As the linear systems solved at each quadrature node are independent and the quality of numerical integration improves with increased quadrature degree, exploiting this property makes sense, particularly within the context of exascale computations. However, it is a slightly surprising result, as previous experiments with FEAST showed diminishing returns for convergence with increased quadrature degree [23], something we do not observe here.

Parameter Level

In addition to the projector type, several algorithmic parameters determine the efficiency of the overall method, most notably the dimension of the subspace and the degree of the polynomial (BEAST-P) or the number of integration nodes (BEAST-C and BEAST-M).

With certain assumptions on the overall distribution of the eigenvalues, clear recommendations for optimum subspace size (as a multiple of the number of expected eigenvalues) and the degree can be given, in the sense that overall work

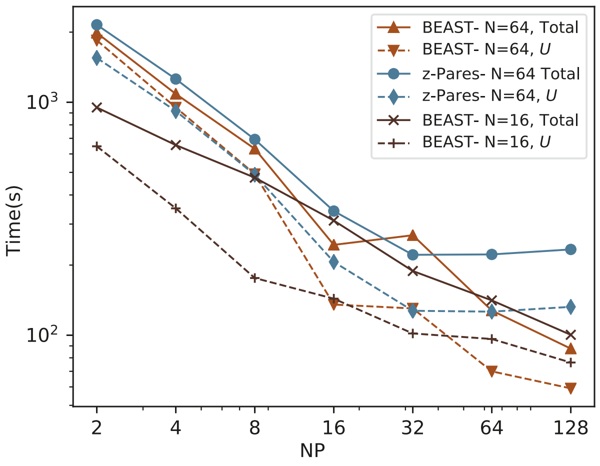


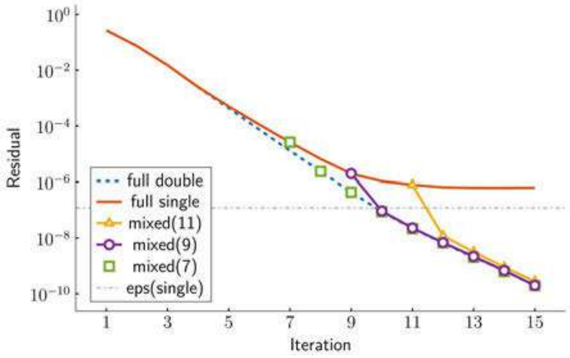
Fig. 6 Strong scaling of BEAST and z-Pares for a 1M × 1M standard eigenproblem based on a graphene sheet of dimension 2000 × 500. Both solvers found 260 eigenpairs in the interval $[-0.01, 0.01]$ to a tolerance of 1×10^{-8}. Both methods used 4 moments and began with random initial block vector Y. For BEAST, Y contained 100 columns; for z-Pares, 130. Testing performed on the Emmy HPC cluster at RRZE. MUMPS was used for the direct solution of all linear systems. N refers to the number of quadrature nodes along a circular contour, NP to the number of processes

is minimized. For more details, together with a description of a performance-tuned kernel for the evaluation of $p(A) \cdot Y$, the reader is referred to [62].

If such information is not available, or for the contour integration-type projectors, a heuristic has been developed that automatically adjusts the degree (or number of integration nodes) during successive iterations in order to achieve a damping of the unwanted components by a factor of 100 per iteration, which leads to close-to-optimum overall effort; cf. [26].

Working Precision Level

Given the iterative nature of BEAST, with one iteration being comparatively expensive, the possibility to reduce the cost of at least some of these iterations is attractive. We have observed that before a given residual tolerance is surpassed, systematic errors in the computation of the projector and other operations do not impair convergence speed per se, but impose a limit on what residual can be reached before progress stagnates. One such systematic error is the finite accuracy of floating-point computations, which typically are available in single and double precision. In the light of the aforementioned behavior, it seems natural to perform initial iterations in single precision and thereby save on computation time before a switch to double precision becomes inevitable; cf. Fig. 7.



Run / precision	Time total	Time projection
double	420 s	305 s
mixed(7)	333 s	265 s
mixed(9)	316 s	257 s
mixed(11)	301 s	245 s
single	266 s	225 s

Fig. 7 Left: average residual over the BEAST iterations for using double or single precision throughout, and for switching from single to double precision in the 7th, 9th, or 11th iteration, respectively. Right: time (in seconds) to convergence for a size 1,048,576 topological insulator (complex values) with a search space size of 256 and a polynomial degree of 135 on 8 nodes of the Emmy-cluster at RRZE. Convergence is reached after identical numbers of iterations (with the exception of pure single precision, of course). The timings can vary for different ratios of polynomial degree and search space size and depend on the single precision performance of the underlying libraries

Therefore, mixed precision has been implemented in all BEAST schemes mentioned above, allowing an adaptive strategy to automatically switch from single to double precision after a given residual tolerance is reached. A comprehensive description and results are presented in [4]. These results and our initial investigations also suggest that increased precision beyond double precision (i.e., quad precision) will have no benefit for the convergence rate until a certain double precision specific threshold is reached; convergence beyond this point would require all operations to be carried out with increased precision.

3.2.3 Levels of Parallelism

The BEAST framework exploits multiple levels of parallelism using an MPI+X paradigm. We rely on the GHOST and PHIST libraries for efficient sparse matrix/dense vector storage and computation; cf. Sect. 5. The operations implemented therein are themselves hybrid parallel and constitute the lowest level of parallelism in BEAST. Additional levels are addressed by parallelizing over blocks of vectors in Y and, for BEAST-C and BEAST-M, over integration nodes during the application of the approximate projector. A final level is added by exploiting the ability of the method to subdivide the search interval $[\underline{\lambda}, \overline{\lambda}]$ and to process the subintervals independently and in parallel. Making use of these properties, however, may lead to non-orthogonal eigenvectors, which necessitates postprocessing as explained in the following.

3.2.4 A Posteriori Cross-Interval Orthogonalization

Rayleigh–Ritz-based subspace iteration algorithms naturally produce a B-orthogonal set of eigenvectors X, i.e., $\mathrm{orth}(X)$ is small, where

$$\mathrm{orth}(X) = \max\left\{\mathrm{orth}(x_i, x_j) | i \neq j\right\} \quad \text{with} \quad \mathrm{orth}(x, y) = \frac{\langle y, x\rangle}{\|x\|\|y\|}.$$

By contrast, the orthogonality

$$\mathrm{orth}(X, Y) = \max\left\{\mathrm{orth}(x_i, y_j)\right\}$$

between two or more independently computed sets of eigenvectors may suffer if the distance between the involved eigenvalues is small [49, 50]. Simultaneous re-orthogonalization of evolving approximate eigenvectors during subspace iteration has proven ineffective unless the vectors have advanced reasonably far. A large scale re-orthogonalization of finished eigenvector blocks, on the other hand, requires a careful choice of methodology in order to not diminish the quality of the previously established residual.

Orthogonalization of multiple vector blocks implies Gram–Schmidt style propagation of orthogonality, assuming $\mathrm{orth}(X, Y)$ can be arbitrarily poor. In practice, the independently computed eigenvectors will exhibit multiple grades of orthogonality, but rarely will there be no orthogonality (in the sense above) at all. This, in turn, allows for the use of less strict orthogonalization methods. While, in theory, the orthogonalization of p blocks requires at least $p(p-1)/2 + (p-1)$ block-block or intra-block orthogonalizations and ensures global orthogonality, an iterative scheme allows for more educated choices on the ordering of orthogonalizations in order to reduce losses in residual and improve the communication pattern, eliminating the need for broadcasts of vector blocks at the cost of additional orthogonalization operations in the form of multiple sweeps. In practice, very few sweeps ($\sim$2) are sufficient in most cases.

Every block-block orthogonalization $X = X - Y(Y^H BX)$ disturbs the orthogonality $\mathrm{orth}(X)$ of the modified block, as well as its residual. Local re-orthogonalization of X disturbs the residual further. We have identified orthogonalization patterns and selected orthogonalization algorithms that reduce the loss of residual accuracy to a degree that essentially eliminates the need for additional post-iteration.

The implementation of an all-to-all interaction of many participating vector blocks can be performed in multiple ways with different requirements regarding storage, communication, runtime, and with different implications on accuracy and loss of residual. Among several such strategies and algorithms that have been implemented and tested, the most promising is a purely iterative scheme, both for global and local orthogonalization operations. It is based on a comparison of interval properties, most notably the achieved residual from the subspace iteration. We are continuing to explore the possibility to detect certain orthogonalizations as unnecessary without computing the associated inner products in order to further reduce the workload without sacrificing orthogonality.

3.2.5 Robustness and Resilience

In the advent of large scale HPC clusters, hardware faults, both detectable an undetectable, have to be expected.

Detectable hardware faults, e.g., the outage of a component that violently halts execution, can typically only be mitigated by frequent on-the-fly storage of the most vital information. In the case of subspace iteration, as is used in BEAST, almost all required information for being able to resume computation is encoded in the iterated subspace basis in form of the approximate eigenvectors, besides runtime information about the general program flow. Relying on the CRAFT library [73], a per-iteration checkpointing mechanism has been implemented in BEAST.

Additionally, for also being able to react to "silent" computation errors that merely distort the results but do not halt execution, the most expensive operation (application of the approximate projector) has been augmented to monitor the sanity of the results. This can be done in two ways: A checksum-style entrainment of additional vectors, linear combinations of the right-hand sides, can be checked during and after the application of the projector to detect errors and allow for the re-computation of the incorrect parts. The comparison of approximate filter values obtained from the computed basis and the expected values obtained from the scalar representation of the filter function, on the other hand, gives an additional a posteriori test for the overall plausibility of the basis.

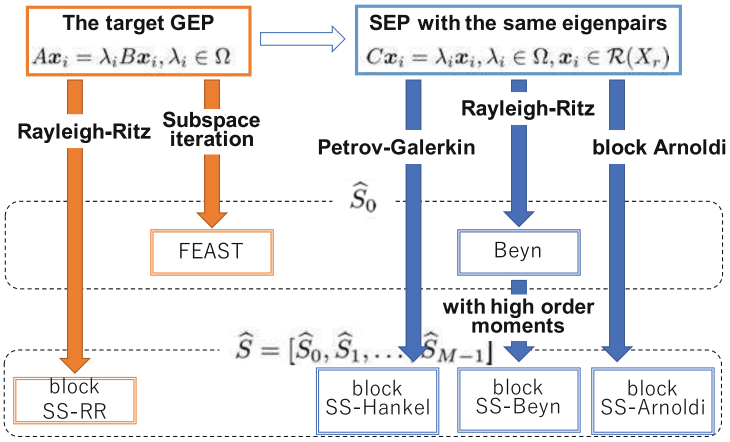
Practical tests have shown that small distortions of the subspace basis have not enough impact on the overall process in order to justify expensive measures. If the error is not recurring, just continuing the subspace iteration is often the best and most cost-efficient option. This is particularly true in early iterations, where small errors have no effect at all.

3.3 Further Progress on Contour Integral-Based Eigensolvers

3.3.1 Relationship Among Contour Integral-Based Eigensolvers

The complex moment-based eigensolvers such as the Sakurai–Sugiura method can be regarded as projection methods using a subspace constructed by the contour integral

$$\frac{1}{2\pi \mathrm{i}} \int_{\Gamma} \mathrm{d}z\, z^k (zB - A)^{-1} BY.$$



The target GEP
$Ax_i = \lambda_i B x_i, \lambda_i \in \Omega$
SEP with the same eigenpairs
$Cx_i = \lambda_i x_i, \lambda_i \in \Omega, x_i \in \mathcal{R}(X_r)$
Rayleigh-Ritz
Subspace iteration
Petrov-Galerkin
Rayleigh-Ritz
block Arnoldi
$\widehat{S}_0$
FEAST
Beyn
with high order moments
$\widehat{S} = [\widehat{S}_0, \widehat{S}_1, \ldots, \widehat{S}_{M-1}]$
block SS-RR
block SS-Hankel
block SS-Beyn
block SS-Arnoldi

Fig. 8 A map of the relationships among the contour integral-based eigensolvers

The property of the subspace is well analyzed by using a filter function

$$f(\lambda) := \sum_{j=1}^{d} \frac{\omega_j}{z_j - \lambda},$$

which approximates a band-pass filter for the target region where the wanted eigenvalues are located. Using the filter function, error analyses of the complex moment-based eigensolvers were shown in [30, 41, 42, 66, 76]. By using the results of the error analyses, an error resilience technique and an accuracy deterioration technique have also been given in [32, 43].

The relationship between typical complex moment-based eigensolvers was also analyzed in [42] focusing on the subspace. The block SS-RR method [36] and the FEAST algorithm [76] are projection methods for solving the target generalized eigenvalue problem, whereas the block SS-Hankel method [37], Beyn [12], the block SS-Arnoldi methods [40] and its improvements [38] are projection methods for solving an implicitly constructed standard eigenvalue problem; see [42] for details. Figure 8 shows a map of the relationships among the contour integral-based eigensolvers.

3.3.2 Extension to Nonlinear Eigenvalue Problems

The complex moment-based eigensolvers were extended to nonlinear eigenvalue problems (NEPs):

$$T(\lambda_i)x_i = 0, \quad x_i \in \mathbb{C}^n \setminus \{0\}, \quad \lambda_i \in \Omega \subset \mathbb{C},$$

where the matrix-valued function $T : \Omega \to \mathbb{C}^{n\times n}$ is holomorphic in an open domain Ω. The projection for a nonlinear matrix function $T(\lambda)$ is given by

$$\frac{1}{2\pi \mathrm{i}} \int_{\Gamma} \mathrm{d}z \; z^k T(z)^{-1} Y.$$

This projection is approximated by

$$U_k = \sum_{j=1}^{N} \omega_j z_j^k T(z_j)^{-1} Y, \quad k = 0, 1, \ldots, m-1.$$

The block SS-Hankel [7, 8], block SS-RR [83], and block SS-CAA methods [39] are simple extensions of the GEP solvers. A technique for improving the numerical stability of the block SS-RR method for NEP was developed in [15, 16].

Beyn proposed a method using Keldysh's theorem and the singular value decomposition [12]. Van Barel and Kravanja proposed an improvement of the Beyn method using the canonical polyadic (CP) decomposition [10].

3.4 Recursive Algebraic Coloring Engine (RACE)

The standard approach to solve the ill-conditioned linear systems arising in BEAST-C or FEAST is to use direct solvers. However, in [24] it was shown that the Kaczmarz iterative solver accelerated by a Conjugate Gradient (CG) method (the so-called CGMN solver [29]) is a robust alternative to direct solvers. Standard multicoloring (MC) was used in [29] for the parallelization of the CGMN kernels. After analyzing the shortcomings of this strategy in view of hardware efficiency, we developed in collaboration with the EXASTEEL-II project the Recursive Algebraic Coloring Engine (RACE) [1]. It is an alternative to the well-known MC and algebraic block multicoloring (ABMC) algorithms [44], which have the problem that their matrix reordering can adversely affect data access locality. RACE aims at improving data locality, reducing synchronization, and generating sufficient parallelism while still retaining simple matrix storage formats such as compressed row storage (CRS). We further identified distance-2 coloring of the underlying graph as an opportunity for parallelization of the symmetric spMVM (SymmSpMV) kernel.

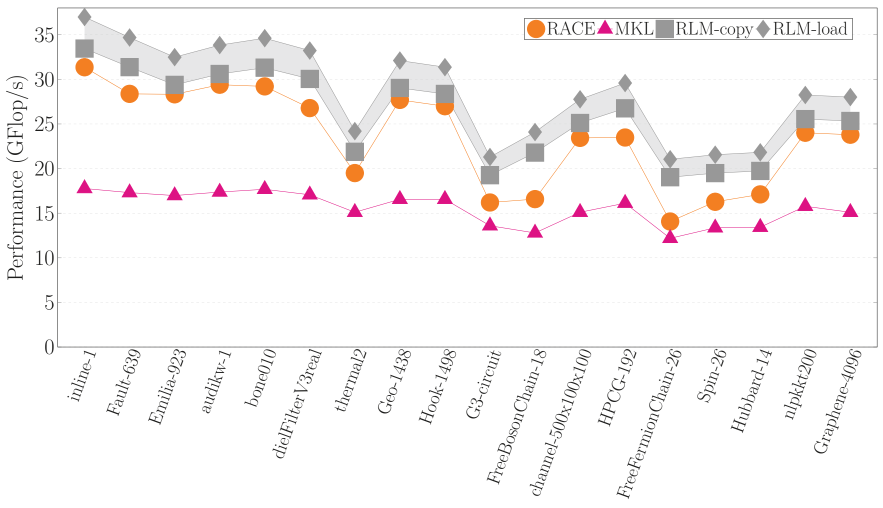
RACE is a sequential, recursive, level-based algorithm that is applicable to general distance-k dependencies. It is currently limited to matrices with symmetric structure (undirected graph), but possibly nonsymmetric entries. The algorithm comprises four steps: level construction, permutation, distance-k coloring, and load balancing. If these steps do not generate sufficient parallelism, recursion on sub-graphs can be applied. Using RACE implies a pre-processing and a processing phase. In pre-processing, the user supplies the matrix, the kernel requirements (e.g., distance-1 or distance-2) and hardware settings (number of threads, affinity strategy). The library generates a permutation and stores the recursive coloring information in a level tree. It also creates a pool of pinned threads to be used later. In the processing phase, the user provides a sequential kernel function which the library executes in parallel as a callback using the thread pool.

Figure 9 shows the performance of SymmSpMV on a 24-core Intel Xeon Skylake CPU for a range of sparse symmetric matrices. In Fig. 9a we compare RACE against Intel's implementation in the MKL library, and with roofline limits obtained via bandwidth measurements using array copy and read-only kernels, respectively. RACE outperforms MKL by far. In Fig. 9b we compare against standard multicoloring (MC) and algebraic block multicoloring (ABMC). The advantage of RACE is especially pronounced with large matrices, where data traffic and locality of access is pivotal. One has to be aware that some algorithms may exhibit a change in convergence behavior due to the reordering. This has to be taken into account when benchmarking whole program performance instead of kernels. Details can be found in [1].

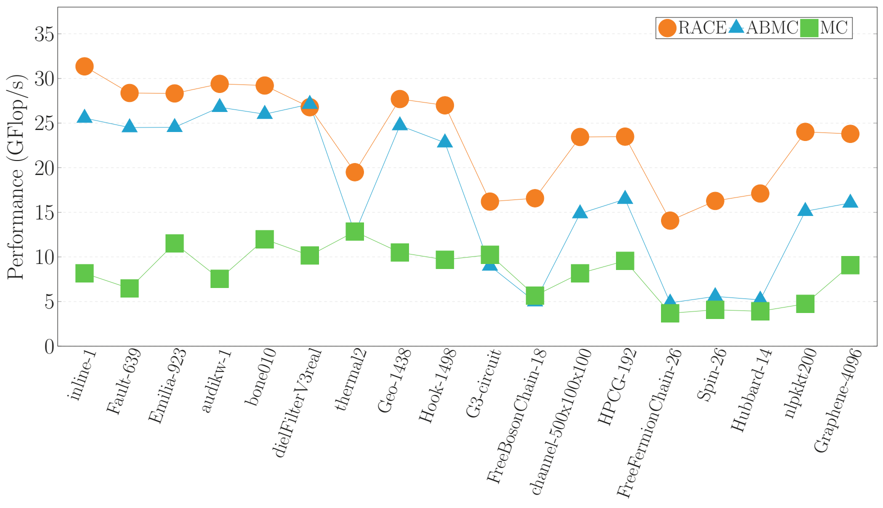
In order to show the advantages of RACE in the context of a relevant algorithm, we chose FEAST [63] for computing inner eigenvalues. The hot spot of the algorithm (more than 95%) is a solver for shifted linear systems ($A - \sigma I = b$). These systems are, however, highly ill-conditioned, posing severe convergence problems for most linear iterative solvers. We use the FEAST implementation of Intel MKL, which by default employs the PARDISO direct solver [69], but its Reverse Communication Interface (RCI) allows us to plug our CGMN implementation instead. In the following experiment we find ten inner eigenvalues of a simple discrete Laplacian matrix to an accuracy of 10^{-8}. Figure 10 shows the measured time and memory footprint of the default MKL version (using PARDISO) and the CGMN versions parallelized using both RACE and ABMC for different matrix sizes. ABMC is a factor of $4\times$ slower than RACE. The time required by the default MKL with PARDISO is smaller than with CGMN using RACE for small sizes; however, the gap gets smaller as the size grows due to the direct solvers having a higher time complexity (here $\approx O(n^2)$) compared to iterative methods ($\approx O(n^{1.56})$). Moreover, the direct solver requires more memory, and the memory requirement grows much faster (see Fig. 10b) than with CGMN. In our experiment the direct solver ran out of memory at problem sizes beyond 140^3, while CGMN using RACE used less than 10% of space at this point. Thus, CGMN with RACE can solve much larger problems compared to direct solvers, which is a major advantage in fields like quantum physics.

4 Hardware Efficiency and Scalability

In this section we showcase performance and parallel efficiency numbers for library components developed in ESSEX-II that are of paramount importance for the application work packages: GPGPU-based tall and skinny matrix-matrix multiplication and the computation of inner eigenvalues using polynomial filter techniques.



(a)



(b)

Fig. 9 SymmSpMV performance of RACE compared to other methods. The roofline model for SymmSpMV is shown in Fig. 9a for reference. Representative matrices from [17] and ScaMaC (see Sect. 5.5) were used. Note that the matrices are ordered according to increasing number of rows. (One Skylake Platinum 8160 CPU [24 threads]). (**a**) Performance of RACE compared with MKL. (**b**) Performance of RACE compared to other coloring approaches

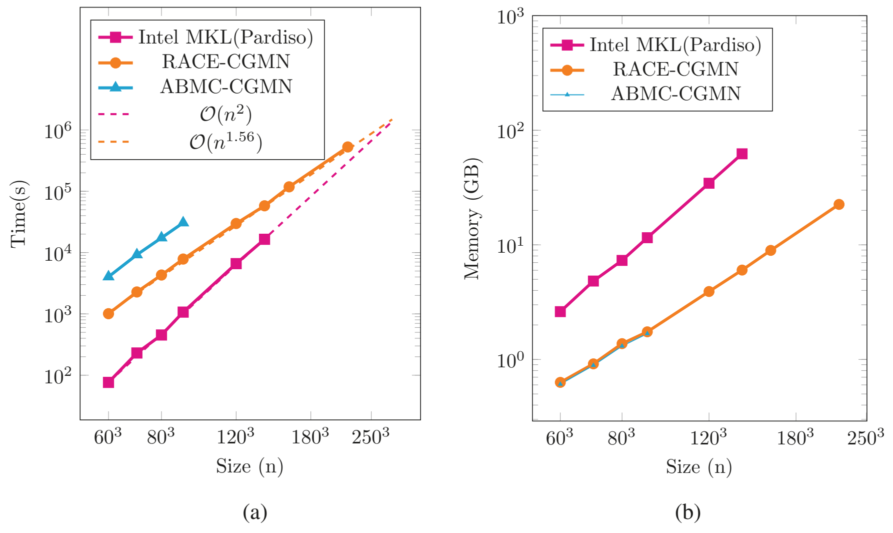


Fig. 10 Comparison of FEAST with default MKL direct solver and iterative solver CGMN, parallelized using RACE. (One Skylake Platinum 8160 CPU [24 threads]). (**a**) Time to solution. (**b**) Memory requirement

4.1 Tall and Skinny Matrix-Matrix Multiplication (TSMM) on GPGPUs

Orthogonalization algorithms frequently require the multiplication of matrices that are strongly nonsquare. Vendor-supplied optimized BLAS libraries often yield sub-optimal performance in this case. "Sub-optimal" is a well-defined term here since the multiplication of an $M \times K$ matrix A with a $K \times N$ matrix B with $K \gg M, N$ and small M, N is a memory-bound operation: At $M = N$, its computational intensity is just $M/8$ flop/byte. In ESSEX-I, efficient implementations of TSMM on multicore CPUs were developed [51].

The naive roofline model predicts memory-bound execution for $M \lesssim 64$ on a modern Volta-class GPGPU. See Fig. 11 for a comparison of optimal (roofline) performance and measured performance for TSMM on an Nvidia Tesla V100 GPGPU using the cuBLAS library.[6] We have developed an implementation of TSMM for GPGPUs [20], investigating various optimization techniques such as different thread mappings, overlapping long-latency loads with computation via leapfrogging[7] and unrolling, options for global reductions, and register tiling. Due

[6]https://docs.nvidia.com/cuda/cublas (May 2019).

[7]Leapfrogging in this context means that memory loads to operands are initiated one loop iteration before the data is actually needed, allowing for improved overlap between data transfers and computations.

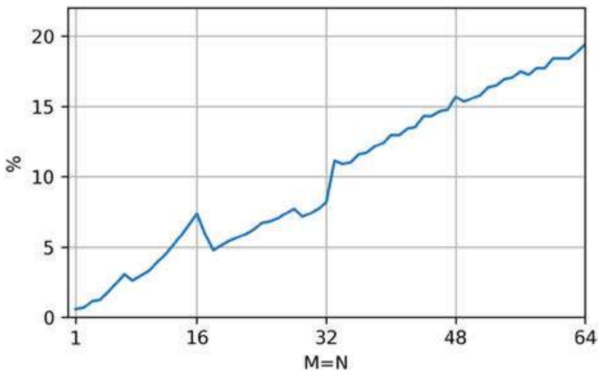


Fig. 11 Percentage of roofline predicted performance achieved by cuBLAS for the range $M = N \in [1, 64]$ on a Tesla V100 with 16 GB of memory. (From [20])

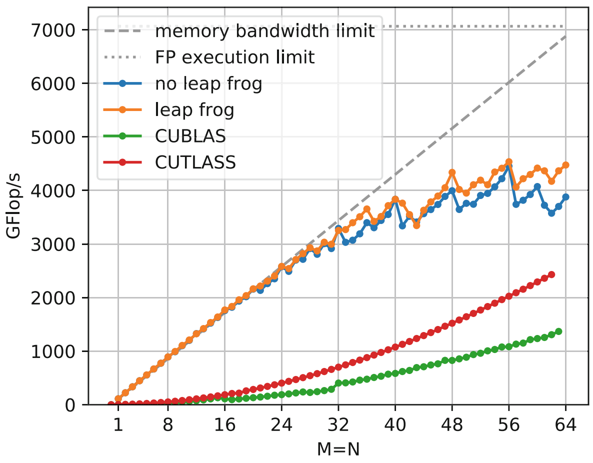


Fig. 12 Best achieved performance for each matrix size with $M = N$ in comparison with the roofline limit, cuBLAS and CUTLASS, with $K = 2^{23}$. (From [20])

to the large and multi-dimensional parameter space, the kernel code is generated using a python script.

Figure 12 shows a comparison between our best implementations obtained via parameter search (labeled "leap frog" and "no leap frog," respectively) with cuBLAS and CUTLASS,[8] which is a collection of CUDA C++ template abstractions for high-performance matrix multiplications. Up to $M = N = 36$, our implementation stays within 95% of the bandwidth limit. Although the performance

[8]https://github.com/NVIDIA/cutlass (May 2019).

levels off at larger M, N, which is due to insufficient memory parallelism, it is still significantly better than with cuBLAS or CUTLASS.

4.2 BEAST Performance and Scalability on Modern Hardware

4.2.1 Node-Level Performance

Single-device benchmark tests for BEAST-P were performed on an Intel Knights Landing (KNL), an Nvidia Tesla P100, and an Nvidia Tesla V100 accelerator, comparing implementations based on vendor libraries (MKL and cuBLAS/cuSPARSE, respectively) with two versions based on GHOST: one with and one without tailored fused kernels. The GPGPUs showed performance levels expected from a bandwidth-limited code, while on KNL the bottleneck was located in the core (see Fig. 13a). Overall, the concept of fused optimized kernels provided speedups of up to 2× compared to baseline versions. Details can be found in [56].

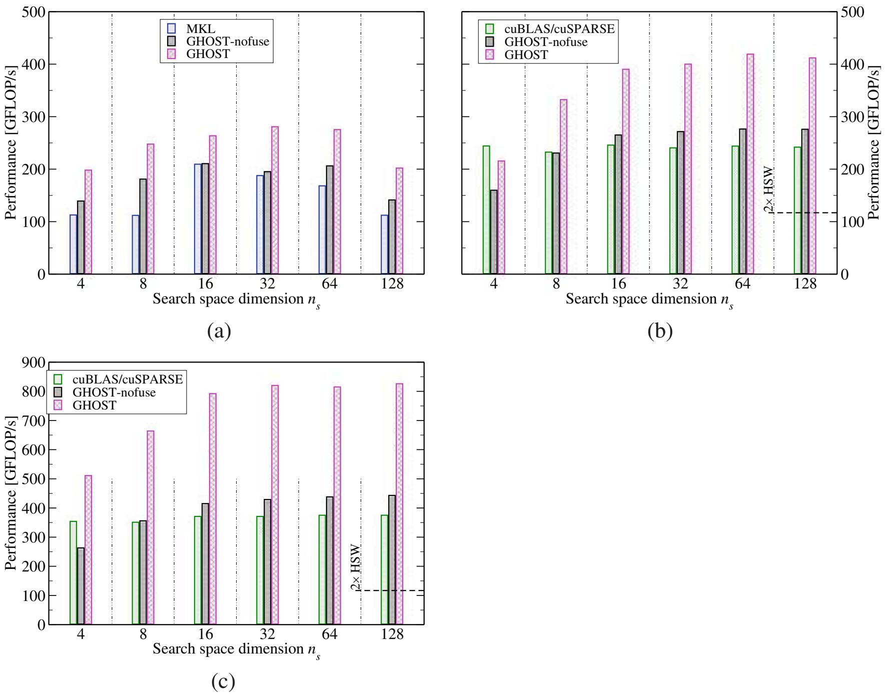


Fig. 13 BEAST-P performance for a topological insulator problem of dimensions 128×64×64 with $n_p = 500$ using different implementations on KNL, P100, and V100. Performance of a dual Xeon E5-2697v3 node (Haswell) is shown for reference. Note the different y axis scaling of the V100 results. (From [56]; for details see therein). (**a**) KNL. (**b**) P100. (**c**) V100

4.2.2 Massively Parallel Performance

Scaling tests for BEAST-P were performed on the "Oakforest-PACS" (OFP) at the University of Tokyo, "Piz Daint" at CSCS in Lugano, and on the "SuperMUC-NG" (SNG) at Leibniz Supercomputing Centre (LRZ) in Garching.[9] While the OFP nodes comprise Intel "Knights Landing" (KNL) many-core CPUs, SNG has CPU-only dual-socket nodes with Intel Skylake-SP, and Piz Daint is equipped with single-socket Xeon "Haswell" nodes, each of which has an Nvidia Tesla P100 accelerator attached. Weak and strong scaling tests were done with topological insulator (TI) matrices generated by the ScaMaC library. Flops were calculated for the computation of the approximate eigenspace, U, averaged over the four BEAST iterations it took to find the 148 eigenvalues in each interval to a tolerance of 1×10^{-10}. The subspace contained 256 columns, and spMMVs were performed in blocks of size 32 for best performance. Optimized coefficients [25] were used for the Chebyshev polynomial approximation, resulting in a lower overall required polynomial degree. Weak and strong scaling results are shown in Fig. 14a through d.

OFP and SNG show similar weak scaling efficiency due to comparable single-node performance and network characteristics. Piz Daint, owing to its superior single-node performance of beyond 400 Gflop/s, achieves only 60% of parallel efficiency at 2048 nodes. A peculiar observation was made on the CPU-only SNG system: Although the code runs fastest with pure OpenMP on a single node (223 Gflop/s), scaled performance was observed to be better with one MPI process per socket. The ideal scaling and efficiency numbers in Fig. 14a–c use the best value on the smallest number of nodes in the set as a reference. The largest matrix on SNG had 6.6×10^9 rows.

5 Scalable and Sustainable Software

It was a central goal of the ESSEX-II project to consolidate our software efforts and provide a library of solvers for sparse eigenvalue problems on extreme-scale HPC systems. This section gives an overview of the status of our software, most of which is now publicly available under a three-clause BSD license. Many of the efforts have been integrated in the PHIST library so that they can easily be used together, and we made part of the software available in larger contexts like Spack [27] and the extreme-scale scientific software development kit xSDK [11]. The xSDK is an effort to define common standards for high-performance, scientific software in terms of software engineering and interoperability.

[9] Runs on OFP and SNG were made possible during the "Large-scale HPC Challenge" Project on OFP and the "Friendly-User Phase" of SNG.

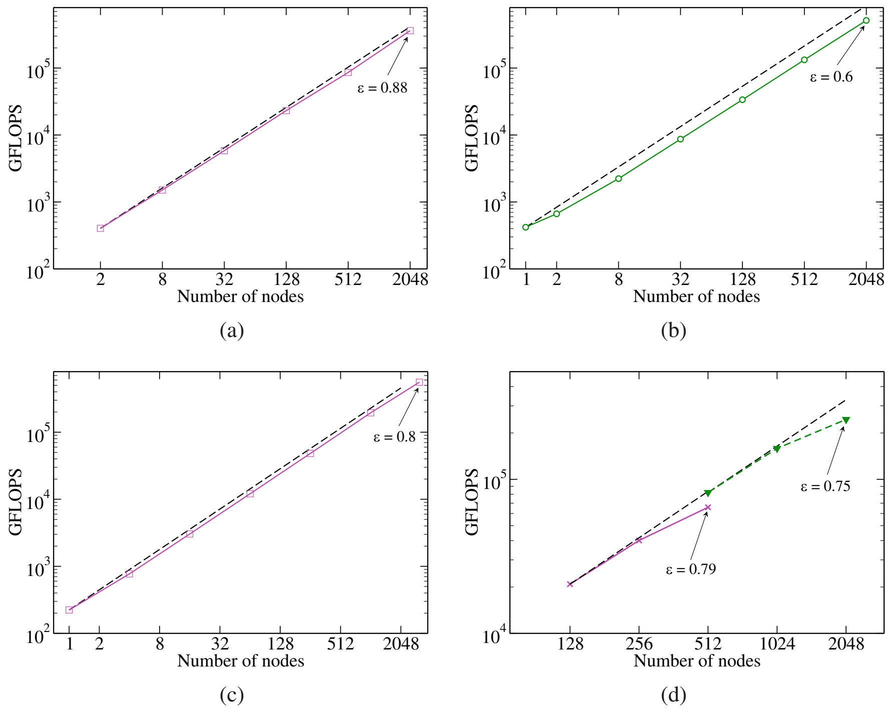


Fig. 14 Weak scaling of BEAST-P on OFP, Piz Daint, and SNG, and strong scaling on SNG. Dashed lines denote ideal scaling with respect to the smallest number of nodes in the set. (**a**) Weak scaling of BEAST-P on OFP for problems of size 2^{20} (2 nodes) to 2^{30} (2048 nodes, about one quarter of the full machine). (**b**) Weak scaling of BEAST-P on Piz Daint for problems of size 2^{21} (1 node) to 2^{32} (2048 nodes). (From [56]). (**c**) Weak scaling of BEAST-P on SNG for problems of size 2^{21} (1 node) to 1.53×2^{32} (3136 nodes, about half of the full machine). (**d**) Strong scaling of BEAST-P on SNG for problems of size 2^{28} (crosses) and 2^{30} (triangles)

The current status of the software developed in the ESSEX-II project is summarized as follows.

- BEAST is available via bitbucket,[10] and can be compiled either using the PHIST kernel interface or the GHOST library directly. The former allows using it with any backend supported by PHIST.
- CRAFT is available stand-alone[11] or (in a fixed version) as part of PHIST.
- ScaMaC is available stand-alone[12] or (in a fixed version) as part of PHIST.

[10]https://bitbucket.org/essex/beast/.

[11]https://bitbucket.org/essex/craft/.

[12]https://bitbucket.org/essex/matrixcollection/.

- GHOST is available via bitbucket.[13] The functionality which is required to provide the PHIST interface can be tested via PHIST. Achieving full (or even substantial) test coverage of the GHOST-functionality would require a very large number of tests (in addition to what the PHIST interface provides, GHOST allows mixing data types, and it uses automatic code generation, which leads to an exponentially growing number of possible code paths with every new kernel, supported processor and data type). It is, however, possible to create a basic GHOST installation via the Spack package manager (since March 2018, commit `bcde376`).
- PHIST is available via bitbucket[14] and Spack (since commit `2e4378b`). Furthermore, PHIST 1.7.5 is part of xSDK 0.4.0. The version distributed with the xSDK is restricted to use the Tpetra kernels to maximize the interoperability of the package.

5.1 PHIST and the Block-ILU

In ESSEX-I we addressed mostly node-level performance [77] on multi-core CPUs. The main publication of ESSEX-II concerning the PHIST library [78] presents performance results for the block Jacobi–Davidson QR (BJDQR) solver on various platforms, including recent CPUs, many-core processors and GPUs. It was also shown in this work that the block variant has a clear performance advantage over the single-vector algorithm in the strong scaling limit. The reason is that, while the number of matrix-vector multiplications increases with the block size (see also [64]), the total number of reductions decreases. In order to demonstrate the performance portability of PHIST, we show in Fig. 15 a weak scaling experiment on the recent SuperMUC-NG machine.

For the block size 4, we roughly match the performance it achieves in the memory-bounded HPCCG benchmark (207 TFlop/s),[15] but using only half of the machine. This gives a clear indication that our node-level performance engineering and multi-node implementation are highly successful: after all, we do not optimize for the specific operator application (a simple structured grid, 3D Laplace operator), which the HPCCG code does. On the other hand, we have an increased computational intensity for some of the operations due to the blocking, which increases the performance over a single-vector CG solver. The single-vector BJDQR solver achieves 98 TFlop/s on half of the machine.

[13] https://bitbucket.org/essex/ghost/.

[14] https://bitbucket.org/essex/phist.

[15] See https://www.top500.org/system/179566.

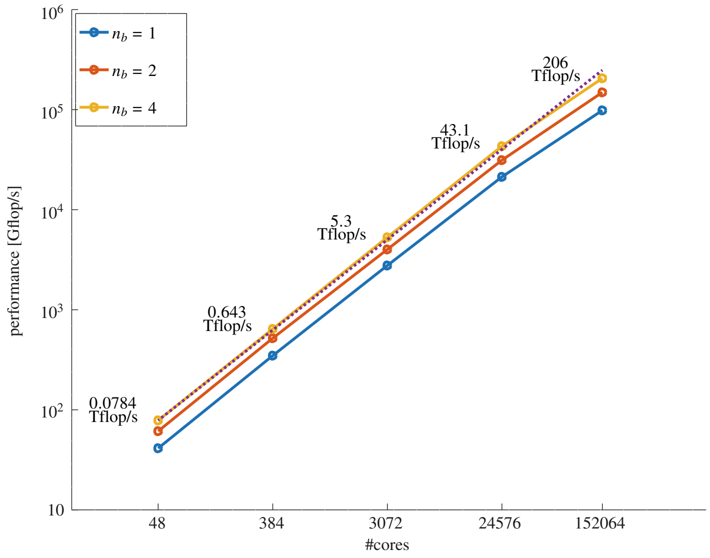


Fig. 15 Weak scaling behavior of the PHIST BJDQR solver for a symmetric PDE benchmark problem and different block sizes

5.1.1 Integration of the Block-ILU Preconditioning Technique

Initial steps have been taken to make the Block-ILU preconditioner (cf. Sect. 3.1) available via the PHIST preconditioning interface. At the time of writing, there is an experimental implementation of a block CRS sparse matrix format in the PHIST builtin kernel library, including parallel conversion and matrix-vector product routines and the possibility to construct and apply the block Cholesky preconditioner. Furthermore, the interfaces necessary to allow using the preconditioner within the BJDQR eigensolver have been implemented. These features are available for experimenting in a branch of the PHIST git repository because they do not yet meet the high demands on maintainability (especially unit testing) and documentation of a publicly available library. Integration of the method with the BEAST eigensolver is not yet possible because the builtin kernel library does not support complex arithmetic. As mentioned in Sect. 3.2, the complex version will be integrated directly into the BEAST software, instead.

5.2 BEAST

BEAST combines implementations of spectral filtering methods for Rayleigh–Ritz type subspace iteration in a generalized framework to provide facilities for improving performance and robustness. The algorithmic foundation allows for the solution of interior Hermitian definite eigenproblems of standard and generalized form via an iterative eigensolver, unveiling all eigenpairs in one or many specified intervals. The software is designed as hybrid parallel library, written in C/C++, and relying on GHOST and PHIST to provide basic operations, parallelism, and data types. Beyond the excellent scalability of the underlying kernel libraries, multiple additional levels of parallelism allow for computing larger portions of the spectrum and/or utilizing a larger number of computing cores. The inherent ability of the underlying algorithm to compute separate intervals independently offers wide potential but requires careful handling of cross-interval interactions to ensure the desired quality of results, which is well supported by BEAST.

The BEAST library interface comes in variations for the common floating point formats (real and complex, single and double precision) for standard and generalized eigenproblems. Additionally, the software offers the possibility to switch precisions on-the-fly, from single to double precision, in order to further improve performance. While BEAST offers an algorithm for standard eigenproblems that completely bypasses the need for linear system solves, other setups typically require a suitable linear solver. Besides a builtin parallel sparse direct solver for banded systems, BEAST includes interfaces to MUMPS and Strumpack, as well as a flexible callback-driven interface for the inclusion of arbitrary linear solvers. It also interfaces with CRAFT and ScaMaC, which provide fault tolerance and dynamic matrix generation, respectively. While working out of the box for many problems, BEAST offers a vast amount of options to tweak the software for the specific problem at hand. A builtin command line parser allows for easy modification. The included application bundles the several capabilities of BEAST in form of a stand-alone tool that reads or generates matrices and solves the specified eigenproblem. As such, it acts as comprehensive example for the usage of BEAST.

The library is still in a development state, and interface and option sets may change. A more comprehensive overview over a selection of features is provided in Sect. 3.2.

5.3 CRAFT

The CRAFT library [73] covers two essential aspects of fault tolerance namely communication, and data recovery of an MPI application in case of process-failures.

In the Checkpoint/Restart part of the library, it provides an easier and extensible interface for making application-level checkpoint/restart. A CRAFT checkpoint can be defined simply by defining a `Checkpoint` object and adding the restart-relevant data in it, as shown in Listing 1. By default, the `Checkpoint::add()` function supports the most frequently used data formats, e.g., "plain old data" (POD), i.e., `int`, `double`, `float`, etc., POD 1D- and 2D-arrays, MPI data-types, etc.. However, it can be easily extended to support any user defined data-types. The `Checkpoint::read()`, `write()` and `update()` methods can then be used to read/write all added checkpoint's data. The library supports asynchronous-

```
#include <mpi.h>
#include <craft.h>
int main(int argc, char* argv[]){
  ...
  size_t n=5, myrank, iteration=1, cpFreq=10;
  double dbl = 0.0;
  int * dataArr   = new int[n];
  MPI_Comm FT_Comm;
  MPI_Comm_dup (MPI_COMM_WORLD, &FT_Comm);
  AFT_BEGIN (FT_Comm, &myrank, argv);
  ********************************************
  Checkpoint  myCP ("myCP", FT_Comm);           * //define checkpoint
  myCP.add ("dbl", &dbl);                        *
  myCP.add ("iteration", &iteration);            *
  myCP.add ("dataArr", dataArr, &n);             AFT Zone
  myCP.commit();                                 *
  myCP.restartIfNeeded (&iteration);             *
  for(; iteration <= 100 ; iteration++){         *
    Computation_communication();                 *
    modifyData(&dbl, dataArr);                   *
    myCP.updateAndWrite (iteration, cpFreq);     *
  }                                              *
  ...                                            *
  ********************************************
  AFT_END();
}
```

Listing 1 A toy-code that demonstrates the simplicity of CRAFT's checkpoint/restart and automatic fault tolerace features in a typical iterative-style scientific application

checkpointing as well as node-level checkpointing using the SCR library [68]. Moreover it supports multi-staged, nested-, and signal-checkpointing.

The Automatic Fault Tolerance (AFT) part of CRAFT provides an easier interface for a dynamic process-failure recovery and management. CRAFT uses the ULFM-MPI implementation for process-failure detection, propagation, and communication recovery procedures, however it considerably reduces the user's effort by hiding these details behind `AFT_BEGIN()` and `AFT_END()` functions as shown in Listing 1. After a process failure, the library recovers the broken communicator (shrinking or non-shrinking by process-spawning), and returns the control back to the program at `AFT_BEGIN()`, where the data can be recovered. Both of these CRAFT functionalities are designed to complement each other, however they can be used independently as well. For detailed explanation of the features included in CRAFT, check [73]. Moreover, the library is available at [72].

5.4 *CRAFT Benchmark Application*

Within the scope of ESSEX, we have integrated CRAFT in the GHOST and PHIST libraries, and the BEAST algorithm.

Figure 16 shows a benchmark comparing the overhead of three different checkpointing strategies for the Lanczos algorithm (GHOST-based eigensolver), Jacobi–Davidson (PHIST-based eigensolver), and the BEAST algorithm. The important

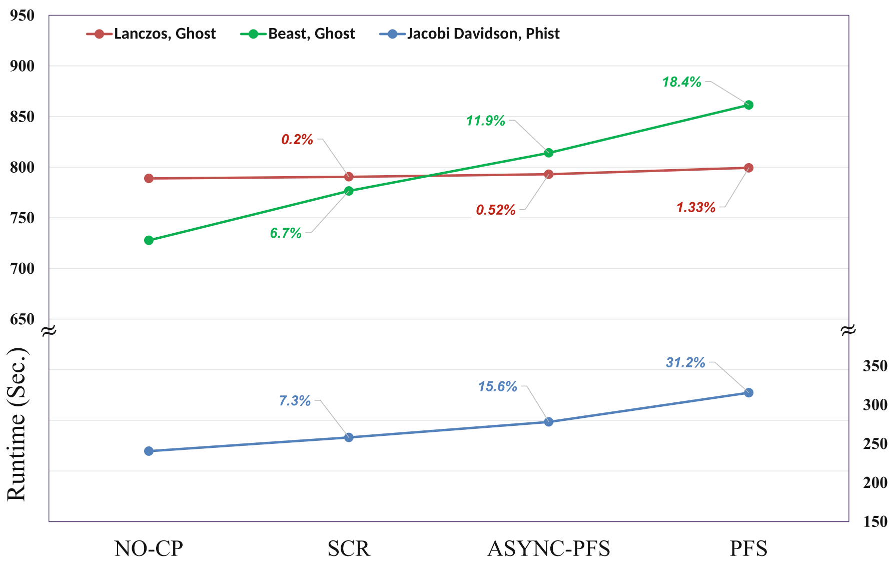


Fig. 16 CRAFT checkpointing overhead comparison for the Lanczos, Jacobi–Davidson, and BEAST eigenvalue solvers using three checkpointing methods of CRAFT, namely, node-level checkpointing with SCR, asynchronous PFS, and synchronous PFS checkpoints. The overhead for each checkpoint case is shown as a percentage. (Number of nodes=128, number of processes=256, Intel MPI)

parameters for these benchmarks are listed in Table 1. The benchmark shows that the node-level and asynchronous checkpointing significantly reduces the checkpoint overhead despite a very high checkpoint frequency.

The benchmark presented in Fig. 17 demonstrates the overhead caused by checkpoint/restart as well as by the communication recovery after process failures for the Lanczos application. The first two bars, namely ‘No CP Intel MPI’ and ‘No CP ULFM-MPI’ show the runtime between non-fault-tolerant (Intel-MPI) vs. a fault-tolerant MPI implementation (ULFM-MPI), and creates a baseline for ULFM-MPI implementation without any failures. The next two groups of bars show the application runtime with 0-,1-, and 2-failures with checkpoints taken on PFS- and node-level. The failures are triggered at the mid-point of two successive checkpoints from within the application to have a deterministic re-computation time, where each failure simulates a complete node-crash (2 simultaneous process failures) and recovery is performed in a non-shrinking fashion on spare nodes. The largest contribution to the overhead is caused by the re-computation part, whereas the communication repair overhead takes an average of ≈ 2.6 s only.

Besides ESSEX, CRAFT has been utilized in [22] to create a process-level fault tolerant FEM code based on the shrinking recovery style. Moreover, CRAFT has been recently integrated in the EXASTEEL [21] project.

Table 1 The parameter values for Lanczos, JD, and Beast benchmarks

Lanczos parameters			
Matrix	Graphene-3000-3000	Number of rows and columns	$9.0 \cdot 10^8$
Number of non-zeros	$11.7 \cdot 10^9$	Global checkpoint size	≈14.4 GB
Number of iterations	3000	Checkpoint frequency	500
Jacobi–Davidson parameters (using Phist)			
Matrix	spinSZ30	Number of rows and columns	$1.6 \cdot 10^8$
Number of non-zeros	$2.6 \cdot 10^9$	Number of sought eigenvalues	20
Number of checkpoints	10		
Global checkpoint size	≈32 GB	Backend support library	GHOST
Beast parameters			
Matrix	tgraphene: 12000, 12000, 0	Number rows and columns	$1.44 \cdot 10^8$
Beast iterations	9	Checkpoint frequency	2
Global checkpoint size	≈65 GB	Backend support library	GHOST

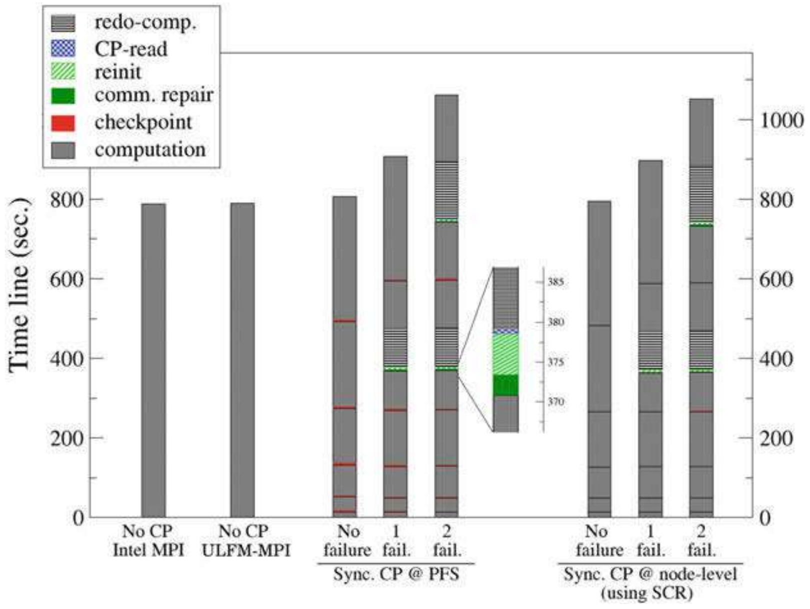


Fig. 17 Lanczos application with various checkpoint/restart and process failure recovery scenarios using 128 nodes (256 processes) on the RRZE Emmy cluster. On average the communication recovery time is 2.6 s (ULFM-MPI v1.1)

5.5 ScaMaC

Sparse matrices are central objects in the ESSEX project because of its focus on large-scale numerical linear algebra problems. A sparse matrix, whether derived from the Hamiltonian of a quantum mechanical system, from the Laplacian in a partial differential equation, or simply given as an abstract entity with unknown properties, defines a problem to be solved. The solution may then consist of a set of eigenvalues and eigenvectors computed with the BEAST or Jacobi–Davidson algorithms or, more moderately, of an estimate of some matrix norm or the spectral radius.

Testing and benchmarking of linear algebra algorithms, but also of computational kernels such as spMVM, requires matrices of different type and different size. Standard collections such as the Matrix Market [58] or Florida Sparse Matrix Collection [17] cover a wide range of examples, but mainly provide matrices of fixed moderate size. As algorithms and implementations improve, such matrices become readily too small and limited to serve as realistic test and benchmark cases.

We therefore decided in the ESSEX project to establish a collection of *scalable* matrices—the ScaMaC. Every matrix in ScaMaC is parameterized by individual parameters that allow the user to scale up the matrix dimension and to modify other, for example spectral, properties of the matrix. ScaMaC includes simple test and benchmark matrices but also 'real-world' matrices from research studies and applications. A major goal of ScaMaC is to provide a flexible yet generic interface for matrix generation, together with the necessary infrastructure to allow for immediate access to the collection irrespective of the concrete usage case.

The ScaMaC approach to matrix generation is straightforward and simple: Matrices are generated row-by-row (or column-by-column). The entire complexity of the actual generation technique, which depends on the specific matrix example, is encapsulated in a `ScamacGenerator` type and hidden from the user. ScaMaC provides routines to create and destroy such a matrix generator, to query matrix parameters prior to the actual matrix generation, and to obtain each row of the matrix. The ScaMaC interface is entirely generic and identical for all matrices in the collection.

A minimal code example is given in Fig. 18. In this example, the matrix and its parameters are set by parsing an argument string of the form `"MatrixName, parameter=...,..."` in line 3, before all rows are generated in the loop in lines 12–17. As this examples shows, parallelization of matrix generation is not part of the ScaMaC, but lies within the responsibility of the calling program. All ScaMaC routines are thread-safe and can be embedded directly into MPI processes and OpenMP threads. This approach guarantees full flexibility for the user and is easily integrated into existing parallel matrix frameworks such as PETSc or Trilinos. Both BEAST and PHIST provide direct access to the ScaMaC, therefore freeing the user from any additional considerations when using ESSEX software.

```
// step 1: obtain a generator - per process
ScamacGenerator * my_gen;
err = scamac_parse_argstr("Hubbard,n_sites=20", &my_gen, &errstr);
err = scamac_generator_finalize(my_gen);
    ................
// step 2: allocate workspace - per thread
ScamacWorkspace * my_ws;
err = scamac_workspace_alloc(my_gen, &my_ws);
    ................
// step 3: generate the matrix row by row
ScamacIdx nrow = scamac_generator_query_nrow(my_gen);
for (idx=0; idx<nrow; idx++) { // parallelize loop with OpenMP, MPI, ...
    // obtain the column indices and values of one row
    err = scamac_generate_row(my_gen, my_ws, idx, SCAMAC_DEFAULT, &nz, cind, val);
    // store or process the row
        ................
}
// step 4: clean up
err = scamac_workspace_free(my_ws);     // in each thread
err = scamac_generator_destroy(my_gen); // in each process
// step 5: use matrix
    ................
```

Fig. 18 Code example for row-by-row matrix generation with the generic ScaMaC generators

ScaMaC is written in plain C. Auto-generated code is included already in the release, such that requirements at compile time are minimal. Interoperability with other programming languages is straightforward, e.g., by using the ISO C bindings of the FORTRAN 2003 standard. Runtime requirements are equally minimal. Matrix generation has negligible memory overhead, requiring only a few KiB workspace to store lookup tables and similar information.

The key feature of ScaMaC is scalability, since the matrix rows (or columns) can be generated independently and in arbitrary order. For example at Oakforest-PACS (see Sect. 4.2.2), a `Hubbard` matrix (see below) with dimension $\geq 9 \times 10^9$ and $\geq 1.5 \times 10^{11}$ non-zeros is generated in less than a minute, using 2^{10} MPI processes each of which generates an average of 1.5×10^5 rows per second. As explained, the task of efficiently storing or using the matrix is left to the calling program.

ScaMaC is accompanied by a small toolkit for exploration of the collection. The toolkit addresses some basic tasks such as querying matrix information or plotting the sparsity pattern, but is not intended to compete with production-level code or full-fledged solver libraries, which the ESSEX project provides with BEAST, GHOST, and PHIST.

At the moment,[16] the matrix generators included in ScaMaC strongly reflect our personal research interests in quantum physics, but the ScaMaC framework is entirely flexible and allows for easy inclusion of new types of matrices, provided that they can be generated in a scalable way. The next update (scheduled for 2020) will extend ScaMaC primarily with matrix generators for standard partial differ-

[16]In version 0.8.2, ScaMaC contains 15 different matrix generators with a total of 95 parameters.

ential equations, including stencil matrices and finite element discretizations for advection-diffusion and elasticity problems, wave propagation, and the Schrödinger equation. Additional examples that are well suited for scalable generation are regular and irregular graph Laplacians, which have gained renewed interest in the context of machine learning [14, 59].

To obtain an idea of the 'real-world' application matrices already contained in ScaMaC, consider two examples: The celebrated Hubbard model of condensed matter physics (`Hubbard`) [34] and a theoretical model for excitons in the cuprous oxide from our own research in this field (`Exciton`) [2]. These matrices appear as Hamiltonians in the Schrödinger equation, and thus are either symmetric real (`Hubbard`) or Hermitian complex (`Exciton`). The respective application requires a moderate number (typically, 10–1000) of extremal or interior eigenpairs, which is less than 0.1% of the spectrum. Other ScaMaC generators provide general (non-symmetric or non-Hermitian) matrices, with a variety of sparsity patterns, spectral properties, etc. All generators depend on a number of application-specific parameters,[17] which are partly listed in Table 2 for the `Hubbard` and `Exciton` generator.

For the `Hubbard` example, two parameters determine the matrix dimension and sparsity pattern: n_fermions gives the number electrons per spin orientation (up or down), n_sites the number of orbitals occupied by the electrons. In terms of these parameters, the matrix dimension is $D = \binom{\text{n\_sites}}{\text{n\_fermions}}^2$. This dependency results in the rapid growth of D shown in Table 3. In the physically very interesting case of half-filling (n_fermions = n_sites/2 = n) we have asymptotically $D \simeq 2^n/\sqrt{(\pi/2)n}$, that is, exponential growth of D.

The `Exciton` example has the more moderate dependence $D = 3(2\text{L} + 1)^3$ (see Table 3). Here, the parameter L is a geometric cutoff that limits the maximal distance between the electron and hole that constitute the exciton. This example has a number of other parameters that are adapted literally from [2]. These parameters enter into the matrix entries, and thus affect the matrix spectrum and, finally, the algorithmic hardness of computing the eigenvalues of interest that determine the physical properties of the exciton.

Both `Hubbard` and `Exciton` are examples of difficult matrices, albeit for different reasons. For `Hubbard`, one unresolved challenge is to compute multiple interior eigenvalues for large n_fermions, n_sites, which becomes extremely difficult because of the rapid growth of the matrix dimension (specialized techniques for the Hubbard model such as the density-matrix renormalization group [70] cannot compute interior eigenvalues). Due to the irregular sparsity pattern of the `Hubbard` matrices (see Fig. 19 below), already the communication overhead of spMVM poses a serious obstacle to scalability and parallel efficiency. For `Exciton`, which are essentially stencil-like matrices of moderate size, the challenge is to compute some hundred eigenvalues out of a strongly clustered spectrum. Here, it is the

[17] For a full list of generators and parameters, consult the ScaMaC documentation included with the code, or at https://alvbit.bitbucket.io/scamac_docs/_matrices_page.html.

Table 2 Parameters of the `Hubbard` and `Exciton` matrix generator in the ScaMaC

`Hubbard`			`Exciton`		
matrix type: symmetric real			matrix type: Hermitian complex		
int	n_sites	number of sites	int	L	cube length
int	n_fermions	number of fermions	double	so	spin orbit
double	t	hopping strength	double	ex	exchange
double	U	Hubbard interaction	double	mlh	mass light hole
	⋮		double	mhh	mass heavy hole
double	ranpot	random potential	double	me	mass electron
rngseed	seed	random seed	double	eps	dielectric constant
				⋮	

Table 3 Matrix dimension D for the `Hubbard` and `Exciton` example, as a function of the respective parameter n_sites (and default value n_fermions = 5) or L

`Hubbard` n_sites	D	`Exciton` L	D
10	63 504	10	27 783
15	9 018 009	20	206 763
20	240 374 016	50	3 090 903
25	2 822 796 900	100	24 361 803
30	20 307 960 036	150	81 812 703
40	432 974 528 064	200	193 443 603

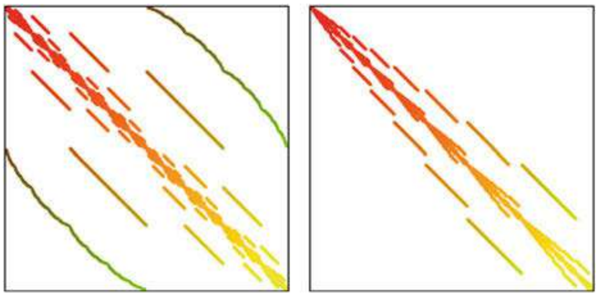


Fig. 19 Sparsity pattern of the Hubbard (`Hubbard,n_sites=40,n_fermions=20`) and spin chain (`SpinChainXXZ,n_sites=32,n_up=8`) example

poor convergence of iterative eigenvalue solvers for nearly degenerate eigenvalues that renders this problem hard. Thanks to the algorithmic advances in the ESSEX project, we now have reached a position that allows for future progress on these problems.

ScaMaC comes with several convenient features. For example, the `Hubbard` matrix includes the parameter `ranpot` to switch on a random potential. Random numbers in ScaMaC are entirely reproducible, and independent of the number of threads or processes that call the ScaMaC routines, or of the order in which the matrix rows are generated. An identical random seed gives the same matrix under all circumstances. In particular, individual matrix rows can be reconstructed at any time, which simplifies a fault-tolerant program design (see Sect. 5.3). Another feature is the possibility to effortlessly generate the (conjugate) transpose of non-symmetric (non-Hermitian) matrices, which is considerably easier than constructing the transpose of a (distributed) sparse matrix after generation.

6 Application Results

6.1 *Eigensolvers in Quantum Physics: Graphene, Topological Insulators, and Beyond*

Because of the linearity of the Schrödinger equation, quantum physics is a paradigm for numerical linear algebra applications. Historically, some application cases, such as the computation of the ground state (i.e. of the eigenvector to the minimal eigenvalue), have received so much attention that only gradual progress remains possible nowadays. In the ESSEX project we instead address two major cases where novel algorithmic improvements and systematic utilization of large-scale computing resources through state-of-the-art implementations still result in substantial qualitative progress. These two cases are the computation of (i) extreme eigenvalues with high degeneracy, which is addressed with a block Jacobi–Davidson algorithm, (ii) multiple interior eigenvalues, which is addressed by various filter diagonalization techniques. Application case (i) has been documented in [64], including the example of spin chain matrices (`SpinChainXXZ` in the ScaMaC). For application case (ii) the primary quantum physics example are graphene [13] and topological insulators [31] (`Graphene` and `TopIns` in the ScaMaC). For these examples, eigenvalues towards the center of the spectrum, near the Fermi energy of the material, are those of interest. This situation is similar to applications in quantum chemistry and density functional theory, but in our case the matrices represent a full (disordered or structured) two or three-dimensional domain, and are usually larger than those considered elsewhere [57].

Starting with the paper [62] on Chebyshev filter diagonalization (ChebFD) and culminating in the BEAST software package (see Sect. 3.2), the computation of interior eigenvalues of large-to-huge graphene and topological insulator matrices has been successfully demonstrated with ESSEX algorithms, using polynomial filters derived from Chebyshev polynomials. Already with the simple ChebFD algorithm we could compute $N_T \simeq 100$ eigenvectors from the center of the spectrum of a matrix with dimension $D \simeq 10^9$ (i.e. an effective problem size $N_T \times D \simeq 10^{11}$), in order to understand the electronic properties of a structured topological insulator (see Figure 13 in [62]). With improved filter coefficients and a more sophisticated implementation, the polynomial filters in the (P-) BEAST package deal with such problems at reduced computational cost (see Sect. 3.2).

Such large-scale computations heavily rely on the optimized spMMVM and TSMM kernels of the GHOST library (see Sect. 5).

To appreciate the numerical progress reflected in these numbers one should note the different scaling of the numerical effort N_{MVM} (measured in terms of the dominant operation of spMVM) for the computation of extreme and interior eigenvalues (cf. the discussion in [62]). In an idealized situation with equidistant eigenvalues, we have roughly $N_{\mathrm{MVM}} \sim D^{1/2}$ for extreme but $N_{\mathrm{MVM}} \sim D$ for interior eigenvalues. For the $D \simeq 10^9$ example, we have to compensate for a factor 10^4–10^5 to enable computation of interior instead of extreme eigenvalues.

Algorithm and software development in ESSEX has been to a large degree application-driven. Now, at the end of the ESSEX project, where the algorithms for our main application cases have become available, we follow two ways to go beyond the initial quantum physics applications. First, entirely new applications can now be addressed with ESSEX software, extending our efforts to non-linear and non-Hermitian problems (see Sect. 6.2). Second, relevant applications such as the `Hubbard` and `Exciton` examples (see Sect. 5.5) still fit into the two major application areas already addressed in ESSEX, but further increase the computational complexity. For `Exciton`, the strongly clustered spectrum with many nearly-degenerate eigenvalues leads to a numerical effort $N_{\mathrm{MVM}} \gg D^{1/2}$ already for extreme eigenvalues. For `Hubbard`, the huge matrix dimension D is a serious obstacle for the computation of interior eigenvalues.

The `Hubbard` matrices also hint at an application-specific issue of general interest that we encountered but could not solve within ESSEX. Specifically, it is the complicated sparsity pattern of many of our quantum physics matrices (see Fig. 19) that adversely affects the parallel efficiency of distributed spMVM, and thus of our entire software solutions. Node-level performance engineering is here easily overcompensated by communication overhead. Unfortunately, the communication overhead is not reduced by standard matrix reordering strategies [61, 71, 84]. This problem can be partially alleviated by overlapping communication with computation, as in the spMMVM (see Sect. 2), but a full solution to restore parallel efficiency is not yet available. Clearly, our different application scenarios still provide enough incentive to think about future numerical, algorithmic, and computational developments beyond the ESSEX project.

6.2 New Applications in Nonlinear Dynamical Systems

The block Jacobi–Davidson QR eigensolver in PHIST is capable of solving non-symmetric and generalized eigenvalue problems of the form

$$\mathbf{A}x = \lambda \mathbf{B}x, \tag{1}$$

where **B** should be symmetric and positive definite. In [75], we exploited several unique features of this implementation to study the linear stability of a three-dimensional reaction-diffusion equation: the Jacobian is non-symmetric, the preconditioner was implemented in Epetra (which can be used directly as a backend for PHIST), and the high degree of symmetry in the model yields eigenvalues with high geometric multiplicity (up to 24). We therefore use a relatively large block size of 8 for these computations to achieve convergence to the desired 20–50 eigenpairs $(\lambda_i, \mathrm{x}_i)$, with the real part of λ_i near 0. In a recent Ph.D. thesis [74], the solver was also used for studying the linear stability of incompressible flow problems. Here **B** is in fact only semi-definite, and the preconditioner has to make sure that the solution stays in the 'divergence-free space', in which the velocity field satisfies $\nabla \cdot \mathrm{u} = 0$ and **B** induces a norm.

Another ongoing effort concerning dynamical systems is the use of PHIST to parallelize the dynamical systems analysis tool PyNCT, which has as its main application the study of superconductors [82]. We have taken first steps to use PHIST as backend for the Python-based algorithms in PyNCT. Furthermore, it is possible to solve the eigenvalue problems arising in PyNCT directly by the BJDQR method in PHIST. Our goal here is the scalable parallel and fully automatic computation of bifurcation diagrams using PyNCT and any backend supported by PHIST.

The Statistical Learning Lab led by Dr. Marina Meila at the University of Washington started to use the PHIST eigensolver to compute spectral gaps for Laplacian matrices obtained from conformation trajectories in molecular dynamics simulations, and other scientific data [14, 59]. These are symmetric positive definite matrices whose dimensions equal the number of simulation steps, typically of the order of $n = 10^6$. When the data intrinsic dimension d is fixed, and much smaller than n (in our examples $d < 10$), the Laplacian is a sparse matrix. The sparsity pattern is not regular, and it is data dependent, as it reflects the neighborhood relationships in the data. Hence, in densely sampled regions rows will have many more non-zeros than in the sparsely populated regions of the data. In a manifold embedding algorithm, the eigengaps identify the optimal number of coordinates in which to embed the data. Furthermore, for data sizes $n \gg 10^6$, PHIST is used to compute the diffusion map embedding itself for the higher frequency coordinates for which existing methods are prohibitively slow.

7 International Collaborations

The internationalisation effort in the second phase of SPPEXA has fostered the ESSEX-II activities in several directions. First and foremost it amplified the scientific expertise in the project. Soon it became clear that complementing knowledge and developments could be leveraged across the partners. A specific benefit of the collaboration between German and Japanese partners is their very different background in terms of HPC infrastructures.Through close personal collaboration within the project all partners could easily access and use latest supercomputers on either side (see Sect. 4.2 and note that the BEAST framework has also been ported to the K-computer). Together with the joint collaboration on scientific problems and software development a steady exchange evolved with many personal research visits which also opened up collaboration with partners not involved directly in ESSEX-II.

The results described in Sect. 3.1 on preconditioners are a direct result of the collaboration of ESSEX-II with the ppOpen-HPC[18] project led by Univ. of Tokyo. On the other hand, the CRAFT library developed at Univ. of Erlangen is utilized in an FEM code of Univ. of Tokyo and is part of a follow on JHPCN project with the German partner involved as associated partner.

Collaboration between Japanese and German working groups made possible the expansion of the BEAST framework for projection based eigensolvers to include Sakurai–Sugiura methods. Various numerical and theoretical issues associated with the implementation of the solver within an iterative framework were resolved, and new ideas explored during research visits. Results based on this collaboration have so far been presented in multiple conferences and a paper in preparation [35].

The linear systems arising from numerical quadrature in the BEAST-C and BEAST-M framework were used in the testing and development of a Block Cholesky-based ILU preconditioner. The integration of an interface to this solver into BEAST has begun. Examining strategies and expectations for solving these extreme ill-conditioned problems was a point of intense discussion and collaboration between working groups. One results was the development of RACE (see Sect. 3.4). Beyond the discussion between several Japanese and German ESSEX-II partners also a strong collaboration with the Swiss partner (O. Schenk) of EXASTEEL-II evolved, who is an expert on direct solvers and graph partitioning. In this context also a collaboration with T. Iwashita (Hokkaido Univ., Japan) started in terms of hardware efficient coloring.

Throughout the project, the variety of large matrices continuously added to the ScaMaC library allowed for testing with a variety of realistically challenging problems of both real and complex types in all ESSEX-II working groups.

Acknowledgments The project is funded by the German DFG priority programme 1648 (Software for Exascale Computing) under the ESSEX-I/II (Equipping Sparse Solvers for Exascale) projects. We are grateful for computer time granted on the LRZ SuperMUC and SuperMUC-NG, the CSCS Piz Daint, and the OakForest PACS systems.

References

1. Alappat, C.L., Hager, G., Schenk, O., Thies, J., Basermann, A., Bishop, A.R., Fehske, H., Wellein, G.: A recursive algebraic coloring technique for hardware-efficient symmetric sparse matrix-vector multiplication. Accepted for publication in ACM Transactions on Parallel Computing. Preprint: http://arxiv.org/abs/1907.06487

[18]http://ppopenhpc.cc.u-tokyo.ac.jp/ppopenhpc/.

2. Alvermann, A., Fehske, H.: Exciton mass and exciton spectrum in the cuprous oxide. J. Phys. B **51**(4), 044001 (2018). https://doi.org/10.1088/1361-6455/aaa060. http://stacks.iop.org/0953-4075/51/i=4/a=044001
3. Alvermann, A., Basermann, A., Fehske, H., Galgon, M., Hager, G., Kreutzer, M., Krämer, L., Lang, B., Pieper, A., Röhrig-Zöllner, M., Shahzad, F., Thies, J., Wellein, G.: ESSEX: equipping sparse solvers for exascale. In: Lopes, L. et al. (eds.) Euro-Par 2014: Parallel Processing Workshops. Lecture Notes in Computer Science, vol. 8806, pp. 577–588. Springer, Berlin (2014). http://doi.org/10.1007/978-3-319-14313-2_49
4. Alvermann, A., Basermann, A., Bungartz, H.J., Carbogno, C., Ernst, D., Fehske, H., Futamura, Y., Galgon, M., Hager, G., Huber, S., Huckle, T., Ida, A., Imakura, A., Kawai, M., Köcher, S., Kreutzer, M., Kus, P., Lang, B., Lederer, H., Manin, V., Marek, A., Nakajima, K., Nemec, L., Reuter, K., Rippl, M., Röhrig-Zöllner, M., Sakurai, T., Scheffler, M., Scheurer, C., Shahzad, F., Simoes Brambila, D., Thies, J., Wellein, G.: Benefits from using mixed precision computations in the ELPA-AEO and ESSEX-II eigensolver projects. Jpn. J. Ind. Appl. Math. **36**, 699–717 (2019). https://doi.org/10.1007/s13160-019-00360-8
5. Anzt, H., Tomov, S., Dongarra, J.: Accelerating the LOBPCG method on GPUs using a blocked sparse matrix vector product. University of Tennessee Innovative Computing Laboratory Technical Report UT-EECS-14-731 (2014). http://www.eecs.utk.edu/resources/library/589
6. Anzt, H., Tomov, S., Dongarra, J.: Implementing a sparse matrix vector product for the SELL-C/SELL-C-σ formats on NVIDIA GPUs. University of Tennessee Innovative Computing Laboratory Technical Report UT-EECS-14-727 (2014). http://www.eecs.utk.edu/resources/library/585
7. Asakura, J., Sakurai, T., Tadano, H., Ikegami, T., Kimura, K.: A numerical method for nonlinear eigenvalue problems using contour integrals. JSIAM Lett. **1**, 52–55 (2009). https://doi.org/10.14495/jsiaml.1.52
8. Asakura, J., Sakurai, T., Tadano, H., Ikegami, T., Kimura, K.: A numerical method for polynomial eigenvalue problems using contour integral. Jpn. J. Ind. Appl. Math. **27**(1), 73–90 (2010). https://doi.org/10.1007/s13160-010-0005-x
9. Balay, S., Abhyankar, S., Adams, M.F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W.D., Karpeyev, D., Kaushik, D., Knepley, M.G., May, D.A., McInnes, L.C., Mills, R.T., Munson, T., Rupp, K., Sanan, P., Smith, B.F., Zampini, S., Zhang, H., Zhang, H.: PETSc Web page (2019). https://www.mcs.anl.gov/petsc
10. Barel, M.V., Kravanja, P.: Nonlinear eigenvalue problems and contour integrals. J. Comput. Appl. Math. **292**, 526–540 (2016). https://doi.org/10.1016/j.cam.2015.07.012
11. Bartlett, R., Demeshko, I., Gamblin, T., Hammond, G., Heroux, M., Johnson, J., Klinvex, A., Li, X., McInnes, L., Moulton, J.D., Osei-Kuffuor, D., Sarich, J., Smith, B., Willenbring, J., Yang, U.M.: xSDK foundations: toward an extreme-scale scientific software development kit. Supercomput. Front. Innov. Int. J. **4**(1), 69–82 (2017). https://doi.org/10.14529/jsfi170104
12. Beyn, W.J.: An integral method for solving nonlinear eigenvalue problems. Linear Algebra Appl. **436**(10), 3839–3863 (2012). https://doi.org/10.1016/j.laa.2011.03.030. Special Issue dedicated to Heinrich Voss's 65th birthday
13. Castro Neto, A.H., Guinea, F., Peres, N.M.R., Novoselov, K.S., Geim, A.K.: The electronic properties of graphene. Rev. Mod. Phys. **81**, 109–162 (2009). https://doi.org/10.1103/RevModPhys.81.109
14. Chen, Y.C., Meilă, M.: Selecting the independent coordinates of manifolds with large aspect ratios (2019, e-prints). arXiv:1907.01651.
15. Chen, H., Imakura, A., Sakurai, T.: Improving backward stability of Sakurai-Sugiura method with balancing technique in polynomial eigenvalue problem. Appl. Math. **62**(4), 357–375 (2017). https://doi.org/10.21136/AM.2017.0016-17
16. Chen, H., Maeda, Y., Imakura, A., Sakurai, T., Tisseur, F.: Improving the numerical stability of the Sakurai-Sugiura method for quadratic eigenvalue problems. JSIAM Lett. **9**, 17–20 (2017). https://doi.org/10.14495/jsiaml.9.17
17. Davis, T.A., Hu, Y.: The University of Florida sparse matrix collection. ACM Trans. Math. Softw. **38**(1), 1:1–1:25 (2011). https://doi.org/10.1145/2049662.2049663

18. Druskin, V., Knizhnerman, L.: Two polynomial methods to compute functions of symmetric matrices. U.S.S.R. Comput. Maths. Math. Phys. **29**(6), 112–121 (1989). https://doi.org/10.1016/S0041-5553(89)80020-5
19. Edwards, H.C., Trott, C.R., Sunderland, D.: Kokkos: enabling manycore performance portability through polymorphic memory access patterns. J. Parallel Distrib. Comput. **74**(12), 3202–3216 (2014). https://doi.org/10.1016/j.jpdc.2014.07.003. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing
20. Ernst, D., Hager, G., Thies, J., Wellein, G.: Performance engineering for a tall & skinny matrix multiplication kernel on GPUs. In: Proceedings PPAM'19: the 13h International Conference on Parallel Processing and Applied Mathematics, Bialystok, Poland, September 8–11, 2019. https://doi.org/10.1007/978-3-030-43229-4_43
21. EXASTEEL project website: www.numerik.uni-koeln.de/14426.html
22. Fukasawa, T., Shahzad, F., Nakajima, K., Wellein, G.: pFEM-CRAFT: a library for application-level fault-resilience based on the CRAFT framework. In: Poster at the 2018 SIAM Conference on Parallel Processing for Scientific Computing (SIAM PP18), Tokyo (2018)
23. Galgon, M., Krämer, L., Lang, B., Alvermann, A., Fehske, H., Pieper, A.: Improving robustness of the FEAST algorithm and solving eigenvalue problems from graphene nanoribbons. PAMM **14**(1), 821–822 (2014). http://doi.org/10.1002/pamm.201410391
24. Galgon, M., Krämer, L., Thies, J., Basermann, A., Lang, B.: On the parallel iterative solution of linear systems arising in the FEAST algorithm for computing inner eigenvalues. Parallel Comput. **49**, 153–163 (2015)
25. Galgon, M., Krämer, L., Lang, B., Alvermann, A., Fehske, H., Pieper, A., Hager, G., Kreutzer, M., Shahzad, F., Wellein, G., Basermann, A., Röhrig-Zöllner, M., Thies, J.: Improved coefficients for polynomial filtering in ESSEX. In: Sakurai, T., Zhang, S.L., Imamura, T., Yamamoto, Y., Kuramashi, Y., Hoshi, T. (eds.) Eigenvalue Problems: Algorithms, Software and Applications in Petascale Computing, pp. 63–79. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62426-6_5
26. Galgon, M., Krämer, L., Lang, B.: Improving projection-based eigensolvers via adaptive techniques. Numer. Linear Algebra Appl. **25**(1), e2124 (2018). http://dx.doi.org/10.1002/nla.2124
27. Gamblin, T., LeGendre, M.P., Collette, M.R., Lee, G.L., Moody, A., de Supinski, B.R., Futral, W.S.: The Spack package manager: bringing order to HPC software chaos (2015). LLNL-CONF-669890
28. Giorgi, P., Vialla, B.: Generating optimized sparse matrix vector product over finite fields. In: Proceedings of ICMS 2014: Fourth International Congress on Mathematical Software, Seoul. Lecture Notes in Computer Science, vol. 8592, pp. 685–690. Springer, Berlin (2014). http://www.lirmm.fr/~giorgi/icms2014-giovia.pdf
29. Gordon, D., Gordon, R.: CGMN revisited: robust and efficient solution of stiff linear systems derived from elliptic partial differential equations. ACM Trans. Math. Softw. **35**(3), 18:1–18:27 (2008). https://doi.org/10.1145/1391989.1391991
30. Guettel, S., Polizzi, E., Tang, P., Viaud, G.: Zolotarev quadrature rules and load balancing for the FEAST eigensolver. SIAM J. Sci. Comput. **37**(4), A2100–A2122 (2015). https://doi.org/10.1137/140980090
31. Hasan, M.Z., Kane, C.L.: Colloquium: topological insulators. Rev. Mod. Phys. **82**, 3045–3067 (2010). https://doi.org/10.1103/RevModPhys.82.3045
32. Hasegawa, T., Imakura, A., Sakurai, T.: Recovering from accuracy deterioration in the contour integral-based eigensolver. JSIAM Lett. **8**, 1–4 (2016). https://doi.org/10.14495/jsiaml.8.1
33. Heroux, M.A., Bartlett, R.A., Howle, V.E., Hoekstra, R.J., Hu, J.J., Kolda, T.G., Lehoucq, R.B., Long, K.R., Pawlowski, R.P., Phipps, E.T., Salinger, A.G., Thornquist, H.K., Tuminaro, R.S., Willenbring, J.M., Williams, A., Stanley, K.S.: An overview of the Trilinos project. ACM Trans. Math. Softw. **31**(3), 397–423 (2005). http://doi.acm.org/10.1145/1089014.1089021
34. Hubbard, J., Flowers, B.H.: Electron correlations in narrow energy bands. Proc. Roy. Soc. Lond. A **276**(1365), 238–257 (1963). https://doi.org/10.1098/rspa.1963.0204

35. Huber, S., Futamura, Y., Galgon, M., Imakura, A., Lang, B., Sakurai, T.: Flexible subspace iteration with moments for an effective contour-integration based eigensolver (2019, in preparation)
36. Ikegami, T., Sakurai, T.: Contour integral eigensolver for non-hermitian systems: a Rayleigh-Ritz-type approach. Taiwan. J. Math. **14**(3A), 825–837 (2010). http://www.jstor.org/stable/43834819
37. Ikegami, T., Sakurai, T., Nagashima, U.: A filter diagonalization for generalized eigenvalue problems based on the Sakurai–Sugiura projection method. J. Comput. Appl. Math. **233**(8), 1927–1936 (2010). https://doi.org/10.1016/j.cam.2009.09.029
38. Imakura, A., Sakurai, T.: Block Krylov-type complex moment-based eigensolvers for solving generalized eigenvalue problems. Numer. Algorithms **75**(2), 413–433 (2017). https://doi.org/10.1007/s11075-016-0241-5
39. Imakura, A., Sakurai, T.: Block SS–CAA: a complex moment-based parallel nonlinear eigensolver using the block communication-avoiding Arnoldi procedure. Parallel Comput. **74**, 34–48 (2018). https://doi.org/10.1016/j.parco.2017.11.007. Parallel Matrix Algorithms and Applications (PMAA'16)
40. Imakura, A., Du, L., Sakurai, T.: A block Arnoldi-type contour integral spectral projection method for solving generalized eigenvalue problems. Appl. Math. Lett. **32**, 22–27 (2014). https://doi.org/10.1016/j.aml.2014.02.007
41. Imakura, A., Du, L., Sakurai, T.: Error bounds of Rayleigh–Ritz type contour integral-based eigensolver for solving generalized eigenvalue problems. Numer. Algorithms **71**(1), 103–120 (2016). https://doi.org/10.1007/s11075-015-9987-4
42. Imakura, A., Du, L., Sakurai, T.: Relationships among contour integral-based methods for solving generalized eigenvalue problems. Jpn. J. Ind. Appl. Math. **33**(3), 721–750 (2016). https://doi.org/10.1007/s13160-016-0224-x
43. Imakura, A., Futamura, Y., Sakurai, T.: An error resilience strategy of a complex moment-based eigensolver. In: Sakurai, T., Zhang, S.L., Imamura, T., Yamamoto, Y., Kuramashi, Y., Hoshi, T. (eds.) Eigenvalue Problems: Algorithms, Software and Applications in Petascale Computing, pp. 1–18. Springer, Cham (2017)
44. Iwashita, T., Nakashima, H., Takahashi, Y.: Algebraic block multi-color ordering method for parallel multi-threaded sparse triangular solver in ICCG method. In: Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IPDPS'12, pp. 474–483. IEEE Computer Society, Washington (2012). https://doi.org/10.1109/IPDPS.2012.51
45. Kawai, M., Iwashita, T., Nakashima, H., Marques, O.: Parallel smoother based on block red-black ordering for multigrid Poisson solver. In: High Performance Computing for Computational Science – VECPAR 2012, pp. 292–299 (2013)
46. Kawai, M., Ida, A., Nakajima, K.: Hierarchical parallelization of multi-coloring algorithms for block IC preconditioners. In: 2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 138–145. IEEE, Piscataway (2017). https://doi.org/10.1109/HPCC-SmartCity-DSS.2017.18
47. Kawai, M., Ida, A., Nakajima, K.: Modified IC preconditioner of CG method for ill-conditioned problems (in Japanese). Tech. Rep. vol. 2017-HPC-158, No.9, IPSJ SIG (2017)
48. Kawai, M., Ida, A., Nakajima, K.: Higher precision for block ILU preconditioner. In: CoSaS2018. FAU (2018)
49. Krämer, L.: Integration based solvers for standard and generalized Hermitian eigenvalue problems. Ph.D. Thesis, Bergische Universität Wuppertal (2014). http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:hbz:468-20140701-112141-6
50. Krämer, L., Di Napoli, E., Galgon, M., Lang, B., Bientinesi, P.: Dissecting the FEAST algorithm for generalized eigenproblems. J. Comput. Appl. Math. **244**, 1–9 (2013)
51. Kreutzer, M.: Performance engineering for exascale-enabled sparse linear algebra building blocks. Ph.D. Thesis, FAU Erlangen-Nürnberg, Technische Fakultät, Erlangen (2018). https://doi.org/10.25593/978-3-96147-104-1

52. Kreutzer, M., Hager, G., Wellein, G., Fehske, H., Bishop, A.: A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units. SIAM J. Sci. Comput. **36**(5), C401–C423 (2014). https://doi.org/10.1137/130930352
53. Kreutzer, M., Pieper, A., Hager, G., Wellein, G., Alvermann, A., Fehske, H.: Performance engineering of the Kernel Polynomal Method on large-scale CPU-GPU systems. In: 2015 IEEE International Parallel and Distributed Processing Symposium, pp. 417–426 (2015). https://doi.org/10.1109/IPDPS.2015.76
54. Kreutzer, M., Thies, J., Pieper, A., Alvermann, A., Galgon, M., Röhrig-Zöllner, M., Shahzad, F., Basermann, A., Bishop, A.R., Fehske, H., Hager, G., Lang, B., Wellein, G.: Performance engineering and energy efficiency of building blocks for large, sparse eigenvalue computations on heterogeneous supercomputers. In: Bungartz, H.J., Neumann, P., Nagel, W.E. (eds.) Software for Exascale Computing—SPPEXA 2013–2015, pp. 317–338. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40528-5_14
55. Kreutzer, M., Thies, J., Röhrig-Zöllner, M., Pieper, A., Shahzad, F., Galgon, M., Basermann, A., Fehske, H., Hager, G., Wellein, G.: GHOST: building blocks for high performance sparse linear algebra on heterogeneous systems. Int. J. Parallel Program. **45**(5), 1046–1072 (2017). https://doi.org/10.1007/s10766-016-0464-z
56. Kreutzer, M., Ernst, D., Bishop, A.R., Fehske, H., Hager, G., Nakajima, K., Wellein, G.: Chebyshev filter diagonalization on modern manycore processors and GPGPUs. In: Yokota, R., Weiland, M., Keyes, D., Trinitis, C. (eds.) High Performance Computing, pp. 329–349. Springer, Cham (2018). https://dx.doi.org/10.1007/978-3-319-92040-5_17
57. Li, R., Xi, Y., Erlandson, L., Saad, Y.: The Eigenvalues Slicing Library (EVSL): algorithms, implementation, and software (preprint). http://www-users.cs.umn.edu/~saad/software/EVSL/index.html
58. Matrix Market: https://math.nist.gov/MatrixMarket/. Accessed 26 July 2019
59. Meila, M., Koelle, S., Zhang, H.: A regression approach for explaining manifold embedding coordinates (2018, e-prints). arXiv:1811.11891
60. Müthing, S., Ribbrock, D., Göddeke, D.: Integrating multi-threading and accelerators into DUNE-ISTL. In: Abdulle, A., Deparis, S., Kressner, D., Nobile, F., Picasso, M., Quarteroni, A. (eds.) Numerical Mathematics and Advanced Applications – ENUMATH 2013. Lecture Notes in Computational Science and Engineering, vol. 103, pp. 601–609. Springer, Berlin (2014). https://doi.org/10.1007/978-3-319-10705-9_59
61. ParMETIS - parallel graph partitioning and fill-reducing matrix ordering. http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview
62. Pieper, A., Kreutzer, M., Alvermann, A., Galgon, M., Fehske, H., Hager, G., Lang, B., Wellein, G.: High-performance implementation of Chebyshev filter diagonalization for interior eigenvalue computations. J. Comput. Phys. **325**, 226–243 (2016). http://dx.doi.org/10.1016/j.jcp.2016.08.027
63. Polizzi, E.: Density-matrix-based algorithm for solving eigenvalue problems. Phys. Rev. B **79**(11), 115112 (2009). https://doi.org/10.1103/PhysRevB.79.115112
64. Röhrig-Zöllner, M., Thies, J., Kreutzer, M., Alvermann, A., Pieper, A., Basermann, A., Hager, G., Wellein, G., Fehske, H.: Increasing the performance of the Jacobi-Davidson method by blocking. SIAM J. Sci. Comput. **37**(6), 206–239 (2015). http://dx.doi.org/10.1137/140976017
65. Sakurai, T., Sugiura, H.: A projection method for generalized eigenvalue problems using numerical integration. J. Comput. Appl. Math. **159**(1), 119–128 (2003). https://doi.org/10.1016/S0377-0427(03)00565-X. Sixth Japan-China Joint Seminar on Numerical Mathematics; In Search for the Frontier of Computational and Applied Mathematics toward the 21st Century

66. Sakurai, T., Asakura, J., Tadano, H., Ikegami, T.: Error analysis for a matrix pencil of Hankel matrices with perturbed complex moments. JSIAM Lett. **1**, 76–79 (2009). https://doi.org/10.14495/jsiaml.1.76
67. Sakurai, T., Futamura, Y., Imakura, A., Imamura, T.: Scalable Eigen-Analysis Engine for Large-Scale Eigenvalue Problems, pp. 37–57. Springer, Singapore (2019). https://doi.org/10.1007/978-981-13-1924-2_3
68. Sato, K., et al.: Design and modeling of a non-blocking checkpointing system. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pp. 19:1–19:10. IEEE Computer Society Press, Los Alamitos (2012)
69. Schenk, O., Gärtner, K., Fichtner, W.: Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors. BIT Numer. Math. **40**(1), 158–176 (2000). https://doi.org/10.1023/A:1022326604210
70. Schollwöck, U.: The density-matrix renormalization group. Rev. Mod. Phys. **77**, 259–315 (2005)
71. SCOTCH: Static mapping, graph, mesh and hypergraph partitioning, and parallel and sequential sparse matrix ordering package. http://www.labri.fr/perso/pelegrin/scotch/
72. Shahzad, F.: Checkpoint/restart and automatic fault tolerance (CRAFT) library. https://bitbucket.org/essex/craft. Accessed 27 July 2017
73. Shahzad, F., Thies, J., Kreutzer, M., Zeiser, T., Hager, G., Wellein, G.: CRAFT: a library for easier application-level checkpoint/restart and automatic fault tolerance. IEEE Trans. Parallel Distrib. Syst. **30**(3), 501–514 (2019). https://doi.org/10.1109/TPDS.2018.2866794
74. Song, W.: Matrix-based techniques for (flow-)transition studies. Ph.D. Thesis, University of Groningen (2019). https://elib.dlr.de/125176/
75. Song, W., Wubs, F.W., Thies, J., Baars, S.: Numerical bifurcation analysis of a 3D Turing-type reaction-diffusion model. Commun. Nonlinear Sci. Numer. Simul. **60**, 145-164 (2018). https://doi.org/10.1016/j.cnsns.2018.01.003
76. Tang, P.T.P., Polizzi, E.: FEAST as a subspace iteration eigensolver accelerated by approximate spectral projection. SIAM J. Matrix Anal. Appl. **35**(2), 354–390 (2014). https://doi.org/10.1137/13090866X
77. Thies, J., Galgon, M., Shahzad, F., Alvermann, A., Kreutzer, M., Pieper, A., Röhrig-Zöllner, M., Basermann, A., Fehske, H., Hager, G., Lang, B., Wellein, G.: Towards an exascale enabled sparse solver repository (2016). In: Software for Exascale Computing – SPPEXA 2013-2015, Volume 113 of the series Lecture Notes in Computational Science and Engineering, 295-316 (2016). http://doi.org/10.1007/978-3-319-40528-5_13. Preprint: https://elib.dlr.de/100211/
78. Thies, J., Röhrig-Zöllner, M., Overmars, N., Basermann, A., Ernst, D., Hager, G., Wellein, G.: PHIST: a pipelined, hybrid-parallel iterative solver toolkit. Accepted for publication in ACM Trans. Math. Softw. Preprint: https://elib.dlr.de/123323/
79. Van der Vorst, H.A.: Iterative Krylov methods for large linear systems, vol. 13. Cambridge University Press, Cambridge (2003)
80. ViennaCL - the Vienna computing library. http://viennacl.sourceforge.net/doc/index.html
81. Weiße, A., Wellein, G., Alvermann, A., Fehske, H.: The kernel polynomial method. Rev. Mod. Phys. **78**, 275–306 (2006). https://link.aps.org/doi/10.1103/RevModPhys.78.275
82. Wouters, M., Vanroose, W.: Automatic exploration techniques for the numerical bifurcation study of the Ginzburg-Landau equation. SIAM J. Dynam. Syst. (2019, submitted). Preprint: https://arxiv.org/abs/1903.02377
83. Yokota, S., Sakurai, T.: A projection method for nonlinear eigenvalue problems using contour integrals. JSIAM Lett. **5**, 41–44 (2013). https://doi.org/10.14495/jsiaml.5.41
84. Zoltan: Parallel partitioning, load balancing and data-management services. http://www.cs.sandia.gov/zoltan/Zoltan.html

ExaDG: High-Order Discontinuous Galerkin for the Exa-Scale

Daniel Arndt, Niklas Fehn, Guido Kanschat, Katharina Kormann, Martin Kronbichler, Peter Munch, Wolfgang A. Wall, and Julius Witte

Abstract This text presents contributions to efficient high-order finite element solvers in the context of the project ExaDG, part of the DFG priority program 1648 *Software for Exascale Computing* (SPPEXA). The main algorithmic components are the matrix-free evaluation of finite element and discontinuous Galerkin operators with sum factorization to reach a high node-level performance and parallel scalability, a massively parallel multigrid framework, and efficient multigrid smoothers. The algorithms have been applied in a computational fluid dynamics context. The software contributions of the project have led to a speedup by a factor $3-4$ depending on the hardware. Our implementations are available via the deal.II finite element library.

D. Arndt
Oak Ridge National Laboratory, Oak Ridge, TN, USA
e-mail: arndtd@ornl.gov

N. Fehn · M. Kronbichler (✉) · W. A. Wall
Technical University of Munich, Garching, Germany
e-mail: fehn@lnm.mw.tum.de; kronbichler@lnm.mw.tum.de; wall@lnm.mw.tum.de

K. Kormann
Max Planck Institute for Plasma Physics, Garching, Germany

Technical University Munich, Garching, Germany
e-mail: katharina.kormann@ipp.mpg.de; katharina.kormann@tum.de

P. Munch
Technical University of Munich, Garching, Germany

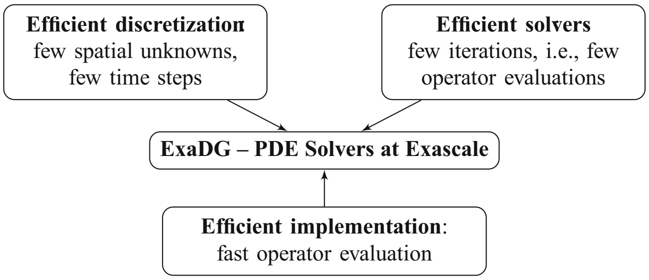
Helmholtz-Zentrum Geesthacht, Geesthacht, Germany
e-mail: munch@lnm.mw.tum.de; peter.muench@hzg.de

G. Kanschat · J. Witte
Heidelberg University, Heidelberg, Germany
e-mail: kanschat@uni-heidelberg.de; julius.witte@iwr.uni-heidelberg.de

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_8

1 Introduction

Exa-scale performance of numerical algorithms is determined by two factors, node-level performance and distributed-memory scalability to thousands of nodes over an Infiniband-type fabric. Additionally, the final application efficiency in terms of time-to-solution is strongly influenced by the choice of numerical methods, where a high sequential efficiency is essential. The project ExaDG aims to bring together these three pillars to create an algorithmic framework for the next generation of solvers for partial differential equations (PDEs). The guiding principles of the project are as follows:


Efficient discretization
few spatial unknowns,
few time steps
Efficient solvers
few iterations, i.e., few
operator evaluations
ExaDG – PDE Solvers at Exascale
Efficient implementation:
fast operator evaluation

If we define the overall goal to be a minimum of computational cost to reach a predefined accuracy, this aim can be split into three components, namely the efficiency of the discretization in terms of the number of degrees of freedom (DoFs) and time steps, the efficiency of the solvers in terms of iteration counts, and the efficiency of the implementation [22]:

$$\begin{aligned} E &= \frac{\text{accuracy}}{\text{computational cost}} \\ &= \underbrace{\frac{\text{accuracy}}{\text{DoFs} \cdot \text{timesteps}}}_{\text{discretization}} \cdot \underbrace{\frac{1}{\text{iterations}}}_{\text{solvers/preconditioners}} \cdot \underbrace{\frac{\text{DoFs} \cdot \text{timesteps} \cdot \text{iterations}}{\text{computational cost}}}_{\text{implementation}} . \end{aligned} \tag{1}$$

We define computational cost as the product of compute resources (cores, nodes) times the wall time resulting in the metric of CPUh, the typical currency of supercomputing facilities.

Regarding the first metric, the type of discretizations in space and time are often the first decision to be made. Numerical schemes that involve as few unknowns and as few time steps as possible to reach the desired accuracy will be more efficient. This goal can be reached by using higher order methods which have a higher resolution capability, especially for problems with a large range of scales and some regularity in the solution [19]. However several possibilities and profound knowledge regarding the performance capability of potential algorithms on modern hardware are still required to select those algorithms and implementations that are optimal with respect to the guiding metric of accuracy versus time-to-solution. High-order (dis-)continuous finite element methods are the basic building block of the ExaDG project due to their generality and geometric flexibility.

Regarding the second metric, solvers are deemed efficient if they keep the number of iterations minimal. We emphasize that "iterations" are defined in a low-level way as the number of operator evaluations, which is also accurate when nesting several iterative schemes within each other. Note that we assume that large-scale systems must be addressed by iterative solvers; in a finite element context sparse direct solvers are not scalable due to fill-in and complex dependencies during factorizations. One class of efficient solvers of particular interest to ExaDG are multigrid methods with suitable smoothers, which have developed to be the gold standard of solvers for elliptic and parabolic differential equations over the last decades. Here, the concept of iterations would accumulate several matrix-vector products within a multigrid cycle that in turn is applied in an outer Krylov subspace solver. Due to the grid transfer and the coarse grid solver, such methods are inherently challenging for highly parallel environments. As part of our efforts in ExaDG, we have developed an efficient yet flexible implementation in the deal.II finite element library [1, 15].

Third, the evaluation of discretized operators and smoothers remains the key component determining computational efficiency of a PDE solver. The relevant metric in this context is the throughput measured as the number of degrees of freedom (unknowns) processed per second (DoFs/s). An important contribution of our efforts is to both tune the implementation of a specific algorithm, but more importantly to also adapt algorithms towards a higher throughput. This means that an algorithm is preferred if it increases the DoFs/s metric, even if it leads to lower arithmetic performance in GFlop/s or lower memory throughput in GB/s. Operator evaluation in PDE solvers only involves communication with the nearest neighbors in terms of a domain decomposition of the mesh, which makes the node-level performance the primary concern in this regard. Since iterative solvers only require the action of the matrix on a vector (and a preconditioner), they are amenable to matrix-free evaluation where the final matrix entries are neither computed nor stored globally in memory in some generic sparse matrix format (e.g., compressed row storage). While matrix-free methods were historically often considered because they lower main memory requirements and allow to fit larger problems in memory [8], their popularity is currently increasing because they need to move less memory: Sparse matrix-vector products are limited by the memory bandwidth on all major computing platforms, so a matrix-free alternative promises to deliver a (much) higher performance.

The outline of this article is as follows. We begin with an introduction of matrix-free algorithms and a presentation of node-level performance results in Sect. 2. In Sect. 3, we describe optimizations of the conjugate gradient method for efficient memory access and communication. Next, we detail our multigrid developments, focusing on performance numbers and the massively parallel setup in Sect. 4 and on the development of better smoothers in Sect. 5. Application results in the field of computational fluid dynamics are presented in Sect. 6, where the efficiency and parallel scalability of our discontinuous Galerkin incompressbile turbulent flow solver are shown. An extension of the kernels to up to 6D PDEs is briefly presented in Sect. 7. We conclude with an outlook in Sect. 8.

2 Node-Level Performance Through Matrix-Free Implementation

An intuitive example of a matrix-free implementation is a finite difference method implemented by its stencil rather than an assembled sparse matrix [33]. For finite element discretizations with sufficient structure of the underlying mesh and low-order shape functions, a small number of stencils allows to represent the operator of a large-scale problem [8]. Such methods are used in the German exascale project TerraNeo, utilizing the regular data structures in hierarchical hybrid grids and embedded into a highly scalable multigrid solver for Stokes systems [31, 32]. By suitable interpolations, the stencils can be extended from the affine coarse grid assumption to also treat smoothly deformed geometries and variable coefficients [7].

For higher-order methods, finite element discretizations lead to fat stencils, making the direct evaluation inefficient even when done through stencils. An alternative matrix-free scheme used in ExaDG is to not compute the explicit DoF coupling and instead turn to integrals underlying the finite element scheme. As an example, we consider the constant-coefficient Laplacian

$$-\nabla^2 u = f \quad \text{in } \Omega, \qquad u = g \quad \text{on } \partial\Omega, \tag{2}$$

whose weak form in a finite-dimensional setting is

$$(\nabla\varphi_i, \nabla u_h)_{\Omega_h} = (\varphi_i, f)_{\Omega_h}, \tag{3}$$

where $u_h(x) = \sum_{j=1:n} \varphi_j(x) u_j$ is the finite element interpolant of the solution with n degrees of freedom, φ_i denotes the test functions with $i = 1, \dots, n$, f is some right hand side, and Ω_h is the finite element representation of the computational domain Ω. The left-hand side of this equation represents a finite element operator, mapping a vector of coefficients $u = [u_i]_i$ to an output vector $v = [v_i]_i$ by evaluating the weak form for all test functions φ_i separately. A matrix-free implementation is obtained by evaluating the element-wise integrals

$$\begin{aligned}\left[(\nabla\varphi_i, \nabla u_h)_{\Omega_h}\right]_{i=1:n} &= \sum_K \int_{\hat K} \left(\mathcal{J}_K^{-\mathsf{T}} \hat\nabla\varphi_i\right)^{\mathsf{T}} \left(\mathcal{J}_K^{-\mathsf{T}} \sum_{j=1}^{n_{\text{dof,ele}}} \hat\nabla\varphi_j u_j^{(K)}\right) \det(\mathcal{J}_K)\, \mathrm{d}\hat{x} \\ &\approx \sum_K I_K^{\mathsf{T}} \left[\sum_{q=1}^{n_q} (\hat\nabla\varphi_{i_K}(\hat x_q))^{\mathsf{T}} \underbrace{\mathcal{J}_K^{-1}\mathcal{J}_K^{-\mathsf{T}} \det(\mathcal{J}_K) w_q}_{\text{physics at quadrature point}} \sum_{j=1}^{n_{\text{dof,ele}}} \hat\nabla\varphi_j(\hat x_q) u_j^{(K)} \right]_{i_K=1:n_{\text{dof,ele}}}\end{aligned} \tag{4}$$

by quadrature on n_q points per cell K. Here, K denotes the elements in the mesh, $\hat{x}$ the coordinates of the reference element $\hat{K} = (0, 1)^d$, $\mathcal{J}_K$ denotes the Jacobian of the mapping from the reference to the real cell, and w_q the quadrature weight. The operator I_K denotes the index mapping from $n_{\text{dof,ele}}$ element-local to global unknowns and defines the element-related unknowns $u^{(K)} = I_K u$.

On element K, the formulation of Eq. (4) consists of two nested sums over the elemental unknowns $u_j^{(K)}$, $j \in n_{\text{dof,ele}}$, and the quadrature points q. The result is tested against all test functions φ_{i_K} on the reference element, which are related to the global test functions φ_i through I_K. Since the metric terms do not depend on the shape function indices i_K and j, and the sum over j does not depend on i_K, the summations in the equation can be broken up into (1) an $dn_q \times n_{\text{dof,ele}}$ matrix operation to evaluate the reference element derivative of $u^{(K)}$ at the quadrature points, (2) the application of metric terms as well as other physics terms at n_q quadrature points, and (3) an $n_{\text{dof,ele}} \times dn_q$ matrix operation to test by all $n_{\text{dof,ele}}$ test functions and perform the summation over the quadrature points. The separation of point-wise physics evaluation at quadrature points is a common abstraction in integration-based matrix-free methods [29, 41, 49, 50].

For high-order finite element methods, the naive evaluation would involve all shape functions at all quadrature points, which is of complexity $\mathcal{O}(k^{2d})$ for polynomials of degree k in d dimensions per element, or $\mathcal{O}(k^d)$ per unknown, similarly to the fat stencil of the final matrix.

At this point, the structure in the reference-cell shape function and quadrature points can be utilized to lower the computational complexity. If the multi-dimensional shape functions are the tensor product of 1D shape functions, and if the quadrature formula is a tensor product of 1D formulas, the so-called sum-factorization algorithm can be used to group common factors along the various dimensions and break down the work into one-dimensional interpolations. Figure 1 visualizes the process of computing the interpolation of nodal values, visualized by black disks, to the values at the quadrature points. Rather than using a naive interpolation of cost $2(k+1)^{2d}$ operations, it can be done in $2d(k+1)^{d+1}$ operations instead. In matrix-vector notation, the interpolation of the gradient with respect to

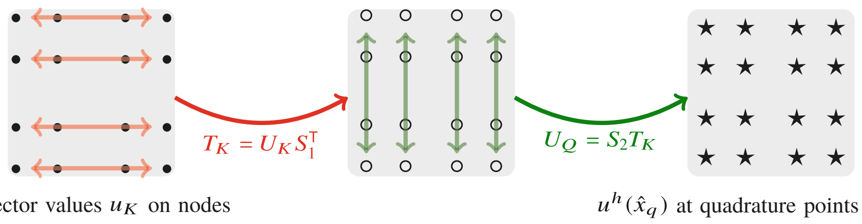

$T_K = U_K S_1^{\mathsf{T}}$
$U_Q = S_2 T_K$
Vector values u_K on nodes
$u^h(\hat{x}_q)$ at quadrature points

Fig. 1 Illustration of sum factorization for interpolation from node values on the left to the values in quadrature points (right)

$\hat{x}$, evaluated at quadrature points, can be written as

$$\begin{bmatrix} \partial u_h/\partial \hat{x}_1 \\ \partial u_h/\partial \hat{x}_2 \\ \vdots \\ \partial u_h/\partial \hat{x}_d \end{bmatrix} = \begin{bmatrix} I \otimes \ldots \otimes I \otimes D_1 \\ I \otimes \ldots \otimes D_2 \otimes I \\ \vdots \\ D_d \otimes I \otimes \ldots \otimes I \end{bmatrix} \left[S_d \otimes \ldots \otimes S_2 \otimes S_1 \right] \boldsymbol{u}^{(K)}. \tag{5}$$

Here, $S_1, \ldots, S_d$ denote the $n_q^{1\text{D}} \times (k+1)$ interpolation matrices from the nodal values to the quadrature points, obtained by evaluating the 1D basis at all 1D quadrature points, and $D_1, \ldots, D_d$ the $n_q^{1\text{D}} \times n_q^{1\text{D}}$ matrices of the derivatives of the Lagrangian basis in quadrature points. In this form, the multiplication by Kronecker matrices is implemented by small matrix-matrix multiplications.

Sum factorization was initially developed in the context of spectral element methods by Orszag [61], see [19] for an overview of the developments. In [12], sum factorization was compared against assembled matrices with the goal to find the best evaluation strategy among assembled matrices and matrix-free schemes. For hexahedral elements considered in this work, the memory consumption and arithmetic complexity indicate that this is the case already for quadratic basis functions [11, 49], with a growing gap for higher polynomial degrees.

2.1 Implementation of Sum Factorization in the deal.II Library

As part of the ExaDG project, we have developed efficient implementations in the deal.II finite element library [1, 4] with the following main features, see [50] for a detailed performance analysis:

- support for both continuous [49] and discontinuous finite elements on uniform and adaptively refined meshes with hanging nodes and deformed elements,
- support for arbitrary polynomial expansions on quadrilateral and hexahedral element shapes as well as tensor product quadrature rules,
- minimization of arithmetic operations by using available symmetries, such as the even-odd decomposition [69] and a switch between the collocation derivative (5) for $n_q^{1\text{D}} \approx k+1$ quadrature points or an alternative variant based on derivatives of the original polynomials as used in [49] and discussed in [29],
- flexible implementation of operations at quadrature points,
- vectorization across several elements to optimally use SIMD units (AVX, AVX-512, AltiVec) of modern processors,
- applicability to modern multi-core CPUs as well as GPUs [51, 57],
- data access optimizations such as element-based loops for DG elements [50, 56],
- and MPI implementation with tight data exchange as well as MPI-only and shared-memory models [43, 48, 54].

The concept of matrix-free evaluation with sum factorization has been widely adopted by now, like in the deal.II [1], DUNE [5, 40, 60], Firedrake [63], mfem [2], Nek5000 [28] or Nektar++ [13] projects. These fast evaluation techniques are directly applicable to explicit time stepping schemes, as we have demonstrated for wave propagation in [42, 53, 65–68] and the compressible Navier–Stokes equations [24]. The proposed developments make matrix-free evaluation of high-order DG operators reach a throughput in unknowns per second almost as high as for optimized 5-wide finite difference stencils in a CFD context [75], despite delivering much higher accuracy.

2.2 *Efficiency of Matrix-Free Implementation*

In Fig. 2, we give an overview of the achieved performance with our framework applied to the discontinuous Galerkin interior penalty (IPDG) discretization of the 3D Laplacian on an affine geometry. The most advanced implementation presented in [50] is used, namely a cell-based loop with a Hermite-like basis for minimal data access [56]. The figure lists the throughput, which is measured by recording the run time of the matrix-vector product in an experiment with around 50 million DoFs (too large to fit into caches), and reporting the normalized quantity DoFs/s obtained by dividing the number of DoFs by the measured run time. The code is run on a single node of six dual-socket HPC systems from the last decade with a shared-memory parallelization with OpenMP, threads pinned to logical cores with the close affinity rule, and using streaming stores to avoid the read-for-ownership data transfer [33] on the result vector. As systems, we consider a 2×8 core AMD Opteron 6128 system from 2010, a 2×8 core Intel Xeon 2680 Sandy Bridge from 2012 (as used in the SuperMUC phase 1 installation in Garching, Germany), a 2×8 core Intel Xeon 2630 v3 (Haswell) representing a medium-core count chip from

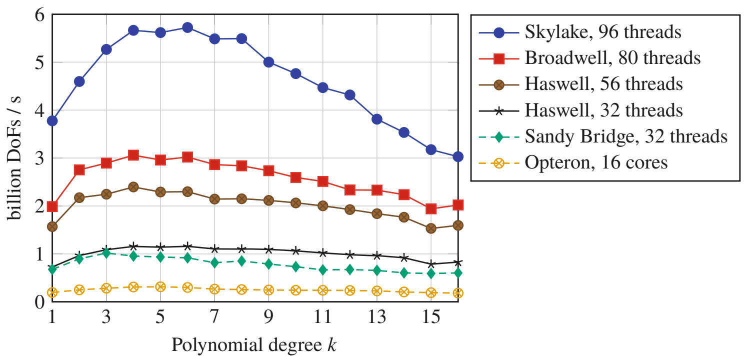


Fig. 2 Throughput of matrix-free evaluation of the IPDG discretization of the 3D Laplacian on an affine grid

2014, a 2 × 14 core Intel Xeon 2697 v3 (Haswell) representing a high-core count chip of the same generation (as used in the SuperMUC phase 2 installation), a 2 × 20 core Intel Xeon 2698 v4 (Broadwell) system from 2016, and a 2 × 24 core Intel Xeon Platinum 8174 from 2017, labeled 'Skylake' in the remainder of this work, and installed in the SuperMUC-NG supercomputer. The chips are operated at 2.0 GHz, 2.7 GHz, 2.4 GHz, 2.6 GHz, 2.2 GHz, and 2.3 GHz, respectively, and all run with fully populated memory interfaces. The Intel machines are run with 2-way hyperthreading, e.g. with 96 threads for the Xeon Platinum Skylake.

The throughput results in Fig. 2 demonstrate the advancements of hardware during the last decade. In particular the increased width of vectorization, from 2 to 4 doubles with Sandy Bridge and from 4 to 8 doubles with Skylake, are clearly visible. Furthermore, the comparison between Sandy Bridge and the smaller Haswell system reveals the benefit of fused multiply-add (FMA) instructions and higher L1 cache bandwidth of the latter: For low polynomial degrees with a modest number of FMA instructions, Sandy Bridge with its higher frequency can approximately deliver the same performance as Haswell. As the polynomial degree is increased, the arithmetic work is increasingly dominated by FMAs in the sum factorization sweeps similar to (5) as shown in [50], and Haswell pulls ahead. Finally, while we observe a throughput of up to 5.7 billion DoFs/s on Skylake (with up to 1.35 TFlop/s for $k = 8$), we observe a relatively strong decrease of performance for polynomial degrees $k \geq 13$: This is because the vectorization across elements leads to an excessive size of the temporary data within sum factorization—here, a different vectorization strategy could lead to better results. However, we consider the polynomial degrees $3 \leq k \leq 8$ most interesting for practical simulations, where almost constant throughput in terms of DoFs/s is reached. This somewhat surprising result, given the expected $\mathcal{O}(1/k)$ complexity of throughput for sum factorization, is because face integrals and memory access with an $\mathcal{O}(1)$ complexity are dominant. Compared to our initial implementation in 2015, which achieved a throughput of 0.32 billion DoFs/s on Sandy Bridge with degree $k = 3$, the progress in software technologies allowed us to reach 1.02 billion DoFs/s on the same system. For Intel Skylake, where memory access is more important, the software progress of our project is more than 4×.

Figure 3 shows the throughput normalized by the number of cores for polynomial degree $k = 4$ over the different hardware generations. For operator evaluation with discontinuous elements and face integrals, approximately 200 floating point operations per unknown are involved with our optimized implementations [50]. At the same time, we must access at least 16 byte (read one double, write one double) plus some neighbors that are not cached, so the arithmetic intensity is around 8–12 Flop/Byte, close to the machine balance of the Skylake Xeon. This means that both memory bandwidth and arithmetic performance are relevant for performance (on one Skylake node, we measured memory throughput of around 160 GB/s, compared to the STREAM limit of 205 GB/s). Likewise, continuous elements evaluated on an affine mesh have seen a considerable increase in throughput per core (arithmetic intensity of 7 Flop/Byte). However, the improvement has been much more modest for continuous elements evaluated on curved elements. In this setting, separate

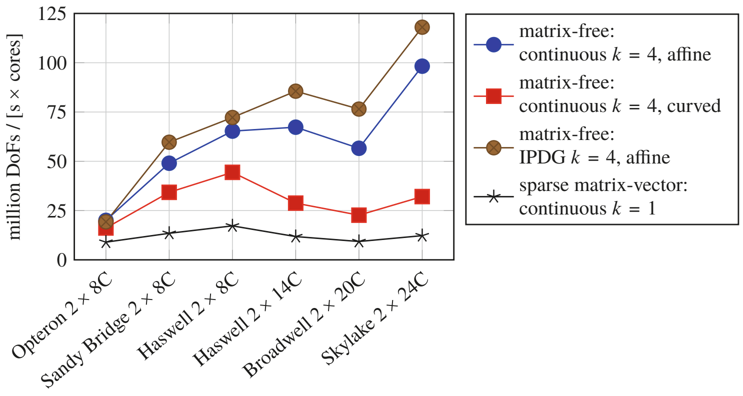


Fig. 3 Evolution of throughput of matrix-vector product per core (computed on a fully populated node and divided by the number of cores) with matrix-free evaluation with $k = 4$ versus a sparse matrix-vector product with continuous linear elements on various hardware

metric terms for all quadrature points and all elements are needed (as opposed to a single term per element in the affine mesh case), reducing the arithmetic intensity to around 1.2 Flop/Byte.[1]

Figure 3 also contains the evolution of performance of a sparse matrix-vector product for tri-linear continuous finite elements. The performance is much lower due to the aforementioned memory bandwidth limit, and has hardly improved per core on Skylake over the dated Opteron architecture. This illustrates the effect of the so-called memory wall. We emphasize that the sparse matrix-vector product for $k = 1$ is more than three times slower than even the matrix-free evaluation for $k = 4$ on curved elements. Hence, high-order methods with matrix-free implementations are faster per unknown on newer hardware, *in addition* to their higher accuracy.

3 Performance-Optimized Conjugate Gradient Methods

The developments of matrix-free implementations presented in the previous section result in a throughput for evaluation of the IPDG operator in Fig. 2 of up to 5.7 billion DoFs/s on Skylake. This is equivalent in time to the mere access of 4.5 doubles per DoF (either reading or writing). In other words, our developments have made the operator evaluation so fast that the matrix-vector product may no longer be the dominant operation in algorithms like the conjugate gradient (CG) method preconditioned by the diagonal, or Chebyshev smoothers. These algorithms

[1]The merged final coefficient tensor $\mathcal{J}_K^{-1}\mathcal{J}_K^{-\mathsf{T}}\det(\mathcal{J}_K)w_q$ is used for the present results, i.e., 6 doubles per quadrature point [29, 51].

involve access to between 6 and 18 vectors for vector updates, the application of the diagonal preconditioner, and inner products. For optimal application performance it is therefore necessary to look into the access to vectors. As proposed in our work [51, 56, 65], merging the vector operations can improve throughput by up to a factor of two, and in particular for the DG case with cell-based loops which allow for a single pass through data [48, 56]. Fusion of different steps of a scheme has also been proposed for explicit time integrators in [14].

For the assessment of optimization opportunities on the algorithm level that goes beyond the matrix-vector product, we consider a high-order finite element benchmark problem suggested by the US exascale initiative "Center for Efficient Exascale Discretization" (CEED). The benchmark involves a continuous finite element discretization of the Laplacian (3), using matrix-free operator evaluation within a conjugate gradient solver preconditioned by the matrix diagonal. In this study, we consider the case BP5 [29], see also https://ceed.exascaleproject.org/bps/, which integrates the weak form (4) of polynomial degree k using a Gauss–Lobatto quadrature formula with $n_q^{\text{1D}} = k + 1$ quadrature points on a cube with deformed elements. While this integration is not exact, it is the typical spectral element setup with an identity interpolation matrix $S_i = I$ in Eq. (5).

Figure 4 lists the contributors to run time for the plain conjugate gradient method preconditioned by the point-Jacobi method as a function of the problem size for the polynomial degree $k = 6$. Here, the metric terms $\mathcal{J}_K$ are computed on the fly from a tri-linear representation of the geometry. Three different performance regimes can be distinguished in the graph: To the left, there is not enough parallelism given the domain decomposition on 48 MPI ranks and batches of 8 elements due to vectorization—indeed, at least 85,000 DoFs are needed to saturate all cores and SIMD lanes. Furthermore, the synchronization barriers due to the inner products in the conjugate gradient method also lead to a slowdown. As the problem size and parallelism increase, the run times decrease significantly and reach a minimum for a problem size around one million DoFs. Here, all data involved in the algorithm

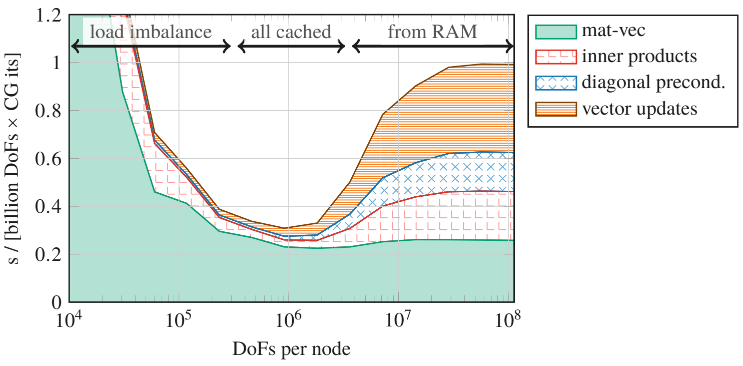


Fig. 4 Breakdown of times per CG iteration in CEED benchmark problem BP5 [29] for the plain conjugate gradient method with $k = 6$ on one node of dual-socket Intel Skylake

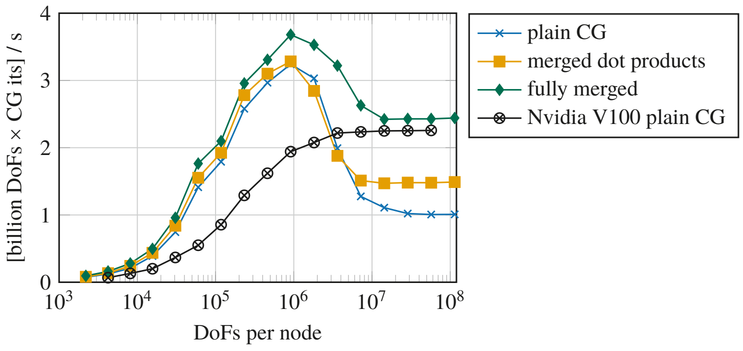


Fig. 5 Study of merged vector operations for conjugate gradient solver for the CEED benchmark problem BP5 [29] on one node of dual-socket Intel Skylake for $k = 6$

fits into the approximately 110 MB of L2+L3 cache on the processors. As the size is further increased, caches are exhausted and most data must be fetched from slow main memory. As a consequence, the run time of the solver increases significantly, and the vector updates, the diagonal preconditioner, and the inner products take a significant share. Note that all vector operations use the hardware optimally with a memory throughput of 205 GB/s.

In order to improve performance, we have therefore developed conjugate gradient implementations with merged vector operations by loop fusion. Figure 5 compares three variants of the conjugate gradient solver: the plain conjugate gradient method runs all vector operations through high-level vector interfaces with separate loops for addition and inner products. In the "merged dot products", we have merged the dot product $p^\mathsf{T} Ap$ following the matrix-vector product into the loop over the elements, and merged the vector updates to the residual and solution with the dot product for $r^\mathsf{T} P^{-1} r$. Here, r denotes the residual vector, p the search direction of the conjugate gradient method, A the matrix operator (represented in a matrix-free way), and P^{-1} the diagonal preconditioner. However, the improvements with this algorithm are relatively modest.

Much more performance can be gained by creating a conjugate gradient variant we call "fully merged": Here, each CG iteration performs a single loop through all vector entries and ideally reads 5 vectors (solution, residual, search direction, temporary vector to hold the matrix-vector product, and diagonal of preconditioner) and writes four (solution, residual, search direction, temporary vector). All vector updates of the previous CG iteration are scheduled before the matrix-vector product and all inner products are scheduled after the matrix-vector product. The vector operations are interleaved with the loop over elements, ensuring that dependencies due to the access pattern of the loop and the MPI communication are fulfilled (this leads to slightly more access in practice). This approach applies the preconditioner several times with partial sums to construct the inner products with a single `MPI_Allreduce`, trading some local computations for the decreased memory access. Of course, fusing the preconditioner into the loop assumes that it is both cheap to apply and does not involve long-range coupling between the DoFs. The results in Fig. 5 show that performance in the saturated limit, i.e., for large sizes beyond 10^7 DoFs, is 2.5 times faster than with the plain CG iteration. Interestingly, this also improves performance for the sizes fitting into caches, which is due to less synchronization and reducing access to the slower L3 cache.

To put the performance of the fully merged case on Intel Skylake into perspective, we compare with executing the plain CG method on an Nvidia V100 GPU using the implementation from [51, 57]: even though the GPU runs with around 700 GB/s of memory throughput, the performance is higher on Intel Skylake with only 200 GB/s from RAM memory because the merged loops significantly increase data locality. Furthermore, on the GPU we do not compute the metric terms on the fly, but load a precomputed tensor $\mathcal{J}_K^{-1}\mathcal{J}_K^{-\mathsf{T}}\det(\mathcal{J}_K)w_q$ which is faster due to reduced register pressure, see also the analysis for BP5 in [71]. We also note that the GPU results with our implementation are faster than an implementation with the OCCA library described in [29] with up to 0.6 billion DoFs/s on a V100 of the Summit supercomputer. The reason is that our implementation uses a continuous finite element storage that does not duplicate the unknowns at shared vertices, edges and faces, which reduces the memory access by about a factor of two. Furthermore, the results from [29] involve a separate gather/scatter step with additional memory transfer to enforce continuity, while this is part of the operator evaluation within a single loop in our code.

Figure 6 lists the achieved throughput with a fully merged conjugate gradient solver for polynomial degrees $k = 2, \ldots, 8$, the most interesting regime for our solvers. We use a tri-linear representation of geometry and compute the geometric

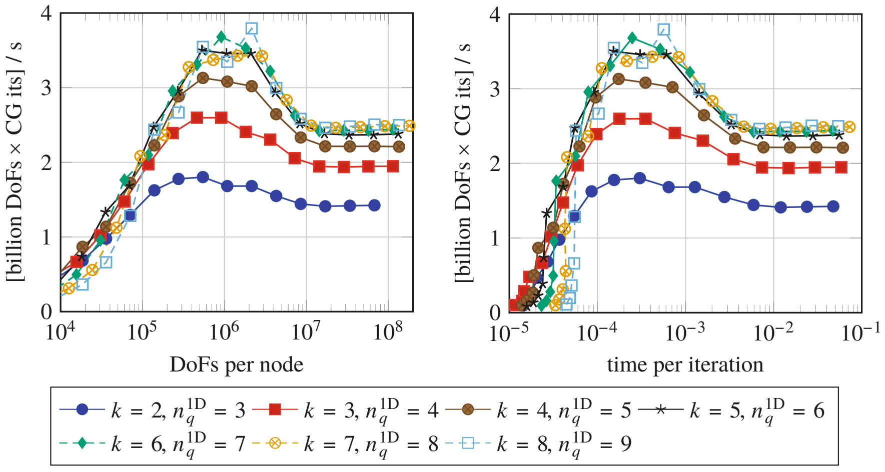


Fig. 6 Throughput of the CEED benchmark problem BP5 [29] on one node of dual-socket Intel Skylake for $k = 2, \ldots, 8$ with fully merged conjugate gradient solver

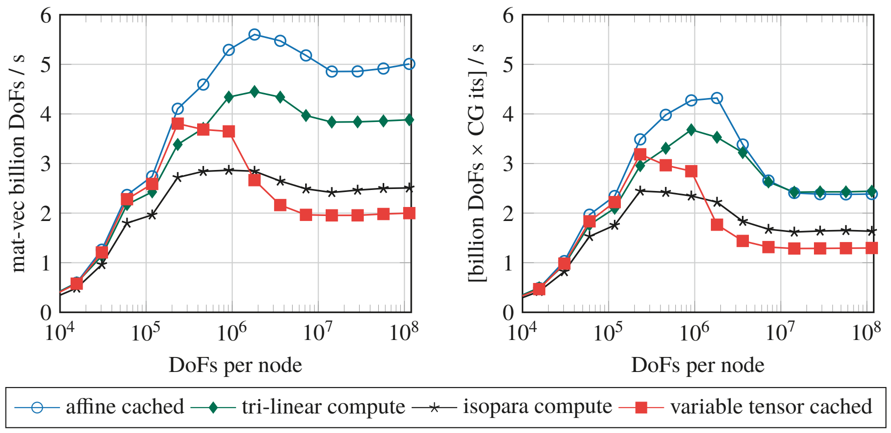


Fig. 7 Study of geometry representation for the CEED benchmark problem BP5 [29] on one node of dual-socket Intel Skylake with $k = 6$ for matrix-vector product only (left) and with fully merged vector operations in the conjugate gradient solver (right)

factors on the fly. Throughput is somewhat lower for quadratic and cubic elements because the geometry data located in the vertices is still noticable.

The results in Fig. 3 motivate the analysis of the representation of the geometry in the matrix-vector product, with results presented in Fig. 7. The figure lists both the throughput of the matrix-vector product in the left panel and the throughput of the complete CG iteration with merged vector operations. Highest performance is obtained for the affine mesh case where our implementation can compress the memory access of the Jacobian. While this case is excluded from the CEED BP5 specification that requires a deformed geometry [29], it is an interesting baseline to compare against. Using separate tensors for each quadrature point, "variable tensor cached", is equally fast as the affine case as long as data fits into caches. However, performance drops once the big geometric arrays must be fetched from main memory. For the case the geometry is computed on the fly from a tri-linear representation of the mesh, i.e., the vertices, the matrix-vector product is slower than the affine variant. For the conjugate gradient solver, however, we observe that the two reach essentially the same performance for five million and more DoFs, as they are both limited by the memory bandwidth from vector access. The "tri-linear compute" case involves a higher Flop/s rate with almost 700 GFlop/s, as compared to the throughput of 330 GFlop/s for the affine mesh case. This means that the merged vector operations allow us to fit additional computations behind the unavoidable memory transfer *without* affecting application performance. Finally, an isoparametric representation of the geometry (labeled "isopara compute" in Fig. 7) can also be computed on the fly by sum factorization from a kth degree polynomial [50]. While this case is obviously slower than the precomputed variable-tensor case from caches, it leverages higher performance when data must be fetched from main memory.

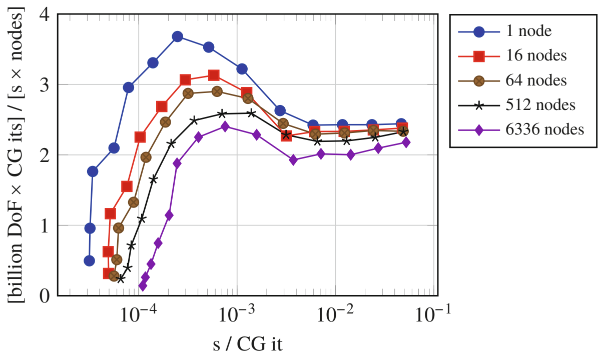


Fig. 8 Throughput of the CEED benchmark problem BP5 [29] with $k = 6$ on up to the full SuperMUC-NG machine. Throughput normalized per node

The tri-linear and isoparametric cases are not equivalent, as only the latter represents higher order curved boundaries. Intermediate polynomial degrees for the geometry are conceivable, which would land between the two in terms of application throughput. To combine the higher performance of the former, we plan to investigate the tradeoffs in more detail in the future, e.g. by using a k-degree representation on a single layer of elements at the boundary and a tri-quadratic representation in the domain's interior.

Finally, Fig. 8 shows the weak scaling of the BP5 benchmark problem up to the full size of the SuperMUC-NG machine with 6336 nodes and 304,128 cores. The data is normalized by reporting the number of DoF per node, so ideal weak scaling would correspond to coinciding lines. While the saturated performance is scaling well, giving a sustained performance of up to 4.4 PFlop/s,[2] most of the in-cache performance advantage is lost due to the communication latency over MPI, see also [62] for limits with MPI in PDE solvers. Defining the strong scaling limit as the point where throughput reduces to 80% of saturated performance [29], it is reached for wall times of 56 µs on 1 node. On 512 nodes, the strong scaling limit is already around 180 µs, whereas it is 245 µs on the full SuperMUC-NG machine. Note that even though most optimizations presented in this section have addressed the node-level performance, we have also considered the strong scaling in our work—indeed, the strong scaling on SuperMUC-NG is excellent with a limit around 5 times lower than the BlueGene-Q results presented in [29].

[2]The LINPACK performance of SuperMUC-NG according to the top500 list is 19.4 PFlop/s. Considering that we use an iterative solver for PDE with optimization of throughput, this is an extremely good value.

4 Geometric Multigrid Methods in Distributed Environments

Multigrid methods are efficient solvers for the linear systems arising from the discretization of elliptic problems, see [30] for a recent efficiency evaluation and [35] for a projection of elliptic solver performance to the exascale setting. They apply simple iterative schemes called smoothers on a hierarchy of coarser problem representations. On each level of the hierarchy, the smoothers address the high-frequency content of the solution by smoothening the error. On a sufficiently coarse level with a small number of unknowns, a direct solver can be applied. The multigrid algorithm can be realized by a V-cycle as illustrated in Fig. 9 or some related cycle (W-cycle or F-cycle). In the matrix-free high-order finite element context, variants of the Chebyshev iteration around a simple additive scheme, such as point-Jacobi or approximate block-Jacobi with some rank-d approximation of the cell matrix, are state of the art. The results in this section are based on this selection. Overlapping Schwarz schemes are a new development detailed in Sect. 5 below.

In terms of finding the coarser representations for the multigrid hierarchy, high-order finite element and discontinuous Galerkin methods permit a range of options. The hierarchy can both be constructed by coarser meshes (h-multigrid), by lowering the polynomial degree (p-multigrid), by a discontinuous-continuous transfer as well as algebraically based on the matrix entries only (algebraic multigrid). The latter do not fit into a matrix-free context, since they explicitly rely on a sparse matrix and also often are not robust enough as the degree increases. As it has been shown by the work [70], scalability to the largest supercomputers is much more favorable if knowledge about coarsening by a mesh can be provided. In other words, geometric multigrid is to be preferred over algebraic multigrid in case there is such structure in the problem.

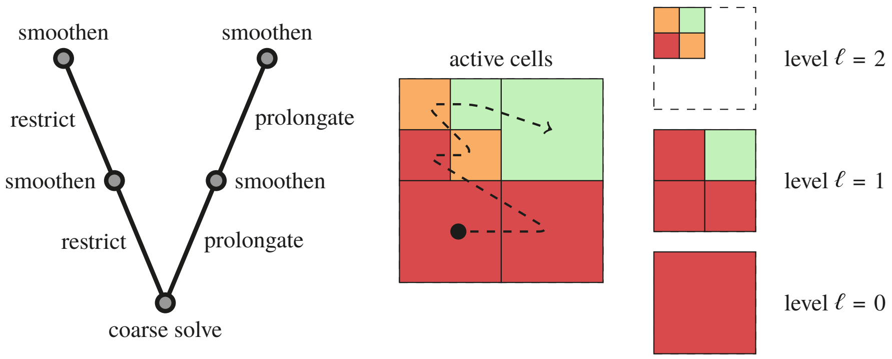


Fig. 9 Illustration of multigrid V-cycle with smoothing on each level and restriction/prolongation between the levels (left) and exemplary partitioning of a grid with adaptive refinement partitioned among 3 processors. The partitioning of the active cells is shown in the mid panel and on the various multigrid levels on the right panel

For these reasons, we have developed a comprehensive geometric multigrid framework with deal.II. In [27], a hybrid multigrid solver with all possibilities of h-, p-, and algebraic coarsening has been combined in a flexible framework, with the possibility to perform an additional c-transfer from discontinuous to continuous function spaces for the DG case. In terms of the h-MG method on adaptive meshes, the deal.II library implements the local smoothing algorithm [9, 36, 37] where smoothing is done level by level. Our work [15] developed a communication-efficient coarsening strategy for this setup, at the cost of a load imbalance for smoothing on the multigrid levels with adaptively refined meshes. The tradeoffs in this choice and the associated costs have been quantified by a performance model in [15].

Figure 10 shows the results of two strong scaling experiments of the multigrid V-cycle with the h-multigrid infrastructure of the deal.II library. The uniform grid and a typical adaptively refined case are compared for the same problem size of 137 million and 46 billion DoFs, respectively, see [15] for details on the experiment. Differences in run time are primarily due to the load imbalance for the level operations. The results demonstrate optimal parallel scaling of both the uniform and adaptively refined cases down to around 10^{-2} s, with a slightly better strong scaling of the adaptive case due to the slower baseline. This performance barrier—typical for strong scaling of multigrid schemes in general—can be explained by the specific type of global communication in this algorithm: from the fine mesh level with many unknowns distributed among a large number of cores, we transfer residuals to coarser meshes with restriction operators until the coarse grid solver is either run on a single core or with few cores in a tightly coupled manner. Then, the coarse-grid corrections are broadcast during prolongation, involving all processors again. The communication pattern of a multigrid V-cycle thus relates to a tree-based

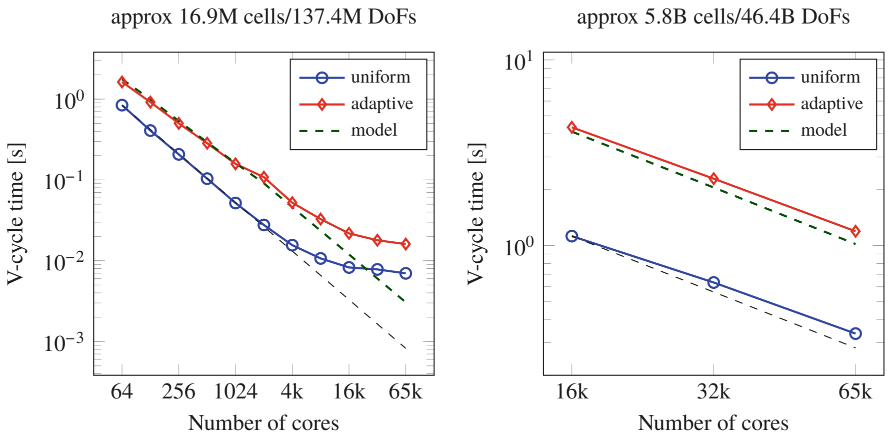


Fig. 10 Strong scaling of geometric multigrid V-cycle for 3D Laplacian on uniform and adaptively refined mesh using continuous Q_2 elements with matrix-free evaluation on up to 4096 nodes (64k cores) of 2 × 8 core Intel Sandy Bridge (SuperMUC phase 1). Adapted from [15]

implementation of `MPI_Allreduce`, with the difference that the communication tree is induced by the grid and substantial operations, namely smoothing and level transfer, are intermixed with the communication. In this particular case, nine matrix-vector products with nearest-neighbor communication are performed per level (eight in the smoother and one for the residual before restriction). In addition, two vertical nearest-neighbor exchange operations are done in restriction and prolongation. A typical matrix-vector product with up to 26 neighbors takes around 10^{-4} s on the chosen Intel Sandy Bridge system when run on a few thousands of nodes [52]. When done on seven levels plus the coarse mesh for the uniformly refined 137 million DoFs case, the expected saturated limit of around 8 ms is exactly seen in the figure. On the newer SuperMUC-NG machine, a latency barrier per V-cycle of around 2–4 ms per V-cycle has been measured, depending on the number of matrix-vector products for the level smoothers. This limit is attractive compared to alternative solvers for elliptic problems such as the fast multipole method or the fast Fourier transform [30, 35].

Multigrid schemes are at the heart of incompressible flow solvers through the pressure Poisson equation, as detailed in Sect. 6 below. Applications of matrix-free geometric multigrid to continuum mechanics were presented in [18] and to electronic calculations with sparse multivectors in [16, 17].

As an example of the large-scale suitability of the developed multigrid framework, Fig. 11 shows two scaling experiments on the SuperMUC-NG supercomputer with up to 304,128 cores of Intel Skylake. Black dashed lines denote ideal strong scaling along a line and weak scaling with a factor of 8 between the lines. The computational domain is a cube meshed by hexahedral elements, using the affine mesh code path for matrix-free algorithms discussed in Sect. 2. A consistent Gaussian quadrature with $n_q^{\mathrm{1D}} = k + 1$ points is chosen. We run a conjugate gradient solver to a relative tolerance of 10^{-3} compared to the initial unpreconditioned residual. This setup is motivated by applications where a very good initial guess is already

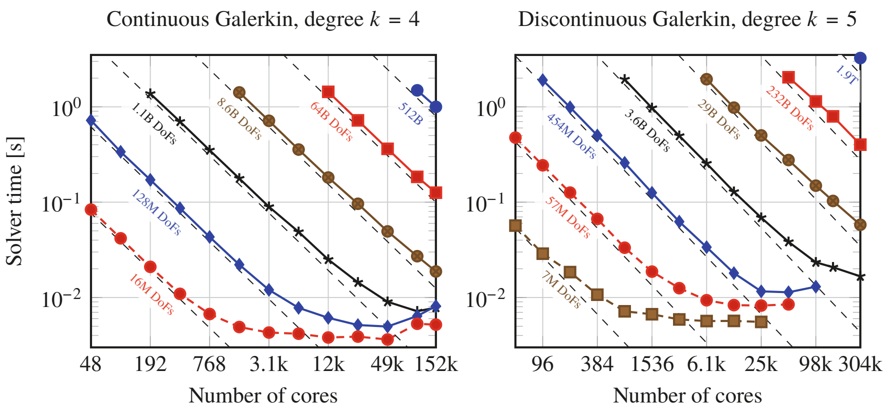


Fig. 11 Multigrid strong scaling analysis for tolerance 10^{-3} with 2 CG iterations

available, e.g. by extrapolation of solutions from the old time step [22, 45], and only a correction is needed. More accurate solves are obtained by tighter tolerances or by full multigrid setups [51]. The multigrid V-cycle is run in single precision to increase throughput, together with a double precision correction through the outer CG solver. This setup has been shown in [51] to increase throughput by around 1.8$\times$ without affecting the multigrid convergence.

In the left panel of Fig. 11, we present results for a continuous Galerkin discretization with a polynomial degree $k = 4$. A pure geometric coarsening down to a single mesh element is used. A Chebyshev iteration of degree five based on the matrix diagonal, i.e., point Jacobi, is used on all levels for pre- and post-smoothing. The maximal eigenvalue $\tilde{\lambda}_{\max}$ is estimated by 15 iterations of a conjugate gradient solver and the Chebyshev parameters are set to smoothen in a range $[0.06\tilde{\lambda}_{\max}, 1.2\tilde{\lambda}_{\max}]$. As a coarse solver, we use a Chebyshev iteration with the degree set to reduce the residual by 10^3 in terms of the Chebyshev a-priori error estimate [74]. We observe ideal weak scaling and strong scaling to around 10^{-2} s. More importantly, the absolute run time is excellent: For instance, the 8.6 billion DoF case on 1536 cores is solved in 1.4 s, i.e., 4.0 million DoFs are solved per core per second.

The right panel of Fig. 11 shows the result for multigrid applied to an IPDG discretization with $k = 5$. Here, we use a transfer from the discontinuous space to the associated continuous finite element space with $k = 5$ on the finest mesh level (see [3] for the theoretical background and [27] for the multigrid context) and then progress by h-coarsening to a single element. On the DG level, we use a Chebyshev smoother around a block-Jacobi method, with the block-Jacobi problems inverted by the fast diagonalization method [58]. On all continuous finite element levels, a Chebyshev iteration around the point-Jacobi method is used. The degree of the Chebyshev polynomial is six. This solver setup achieves a multigrid convergence rate of about 0.025, i.e., reduces the residual by 3 orders of magnitude with two V-cycles. If used in a full multigrid setting [51], a single V-cycle on the finest level would suffice to solve the problem to discretization accuracy. Merged vector operations with a Hermite-like basis for the Chebyshev iteration are used according to [56]. The final application performance of the largest computation on 1.9 trillion DoFs is 5.9 PFlop/s, with 5.6 PFlop/s done in single precision and 0.27 PFlop/s in double precision. The limiting factor is mostly memory transfer, however, with an application throughput of around 175 GB/s per node (the STREAM limit of one node is 205 GB/s).

5 Fast Tensor Product Schwarz Smoothers

In Sect. 4, we have discussed a scalable implementation of geometric multigrid methods, obtaining an efficient solver in the sense of cost per iteration. It employs the matrix-free operator implementation from Sect. 2 in order to reduce the computational cost for residuals and grid transfer. The missing building block for our cost model in Eq. (1) is an efficient implementation (in terms of computational cost per DoF) of an efficient smoother (in terms of number of multigrid iterations).

The main challenge consists of finding preconditioners whose cost is similar to operator evaluation. So far, we have discussed Chebyshev smoothers, which can be implemented matrix-free in a straight-forward fashion. Alas, their performance is not robust for higher order elements. Likewise, from an arithmetic cost point of view sparse matrices can be competitive at most for moderate polynomial degrees $k = 2, 3$ [55] or when done via auxiliary spaces of linear elements on a subdivided grid using some matrix-based preconditioner. However, Fig. 3 shows that even sparse matrices for linear elements are up to 10 times slower than the matrix-free operator evaluation. It seems that only the two SPPEXA projects ExaDUNE and ExaDG have addressed this question in [6, 76]. While [6] focuses on iterative solution of cell problems for multigrid smoothing, we consider domain decomposition based smoothers in the form of multilevel additive and multiplicative Schwarz methods based on low-rank tensor approximations. They consist of a subdivision of the mesh on each level into small subdomains consisting either of a single cell, or of the patch of cells sharing a common vertex. On each of these subdomains, local finite element problems are solved. Comparing with operator application, these smoothers share the structural property of evaluation of local operators on mesh cells or on a patch of cells. They differ by the fact that the smoothers involve local inverses instead of local forward operators, and that these local inverses in general are not amenable to a tensor decomposition like sum factorization. There is one exception though, namely separable differential operators. In d dimensions these can be written in the form

$$L = I_d \otimes \cdots \otimes I_2 \otimes L_1 + \cdots + L_d \otimes I_{d-1} \otimes \cdots \otimes I_1, \tag{6}$$

where L_k are one-dimensional differential operators and I_k are identity operators in directions $k = 1, \ldots, d$. This representation transfers to finite element operators with tensor product shape functions in a straight-forward way, reading I_k as one-dimensional mass matrices M_k.

Due to [58], the inverse of L can be represented as the product

$$L^{-1} = Q\Lambda^{-1}Q^{\mathsf{T}}, \tag{7}$$

with the diagonal matrix $\Lambda = I_d \otimes \cdots \otimes I_2 \otimes \Lambda_1 + \cdots + \Lambda_d \otimes I_{d-1} \otimes \cdots \otimes I_1$, where I_k denote identity matrices, and a rank-1 decomposition $Q = Q_d \otimes \cdots \otimes Q_1$. The tensor factors are obtained by solving d generalized eigenvalue problems

$$\begin{aligned} \Lambda_k &= Q_k^{\mathsf{T}} L_k Q_k, \\ I_k &= Q_k^{\mathsf{T}} M_k Q_k, \qquad k = 1, \ldots, d. \end{aligned} \tag{8}$$

Thus, the computational effort for computing the inverse has been reduced from $\mathcal{O}(k^{3d})$ to $\mathcal{O}(dk^3)$ and for the application of local solvers from $\mathcal{O}(k^{2d})$ to $\mathcal{O}(dk^{d+1})$

by exploiting sum factorization. Based on this technique, we have implemented a geometric multigrid method in [76] based on earlier work in [37–39].

5.1 The Laplacian on Cartesian Meshes

In order to test our concept and to obtain a performance baseline for more complicated cases, we first attend to the case where the decomposition described above can be applied in a straightforward way, namely the additive Schwarz method with subdomains equal to mesh cells. As Table 1 shows, it yields an efficient preconditioner with less than 25 conjugate gradient steps for a gain of accuracy of 10^8. While it is uniform in the mesh level, it is not uniform in the polynomial degree due to the increasing penalty parameter of the interior penalty method. The computational effort for a smoothing step based on local solvers in the form (7) is below the effort for a matrix-free operator application for polynomial degrees between 3 and 15 in three dimensions because it only involves operations on cells. The setup time for computing Q and Λ is even less. Thus, in the context of the performance analysis of the conjugate gradient method in Sect. 2, it barely adds to the cost per iteration step, but reduces the number of matrix-vector products when comparing to the accumulated numbers within a Chebyshev/point Jacobi method, and almost independently of polynomial degree.

In view of application to incompressible flow, we also study vertex patches as typical subdomains for smoothing. First, we observe that a regular vertex patch with 2^d cells attached to a vertex inherits the low-rank tensor product structure from its cells, possibly after renumbering due to changes in orientation. Thus, we can apply the same method as on a single cell, resulting effectively in a factor 2^d in the complexity estimates above. Patches around vertices with irregular topology like 3 or 5 cells in two dimensions do not possess a tensor product structure. Fortunately,

Table 1 Fractional CG iterations, preconditioned by h-MG with additive Schwarz smoother on cells

Levels	Convergence steps			
2D	$k = 3$	$k = 4$	$k = 7$	$k = 10$
7	14.5	14.3	18.8	20.9
8	14.5	14.3	18.8	20.9
9	14.5	14.3	18.8	20.9
10	14.5	14.3	18.8	20.9
3D	$k = 3$	$k = 4$	$k = 7$	$k = 10$
3	16.7	16.8	22.0	24.5
4	17.1	17.0	22.0	24.5
5	17.2	17.0	22.1	24.6
6	17.1	17.0	22.1	24.7

Relative solver tolerance of 10^{-8} and relaxation parameter $\widehat{\omega} = 0.7$

on meshes obtained by refinement of a coarse mesh, they are all determined by irregularities of the coarse mesh and thus small in number.

Vertex patches lead to overlapping decompositions with overlap of at least 4 and 8 in two and three dimensions, respectively. From the analysis of Schwarz methods, it becomes clear that a multiplicative method is required for highest multigrid convergence rates. In order to parallelize such a smoother and to avoid race conditions, mesh cells are colorized, that is, they are separated into "colors" such that patches of the same "color" do not share any common face or cell. As a consequence, the multiplicative method coincides with an additive method within each color, such that we can execute the local solvers in parallel within each color, and the colors sequentially. Typical convergence results for the Laplacian are reported in Table 2, suggesting that this scheme is almost a direct solver.

The vertex patch has 4 and 8 times as many unknowns as a single mesh cell in two and three dimensions, respectively. Thus, the effort for a smoothing step with 16 colors and the optimizations described above turns out to be about 20 to 24 times the effort of a matrix-free operator application, measured over polynomial degrees from 3 to 15. This seems excessive at first glance, but it must be kept in mind that the Chebyshev smoother of degree 6 used in Fig. 11 also involves 12 matrix-vector products for pre- and post-smooting. Futhermore, the current scheme comes with a reduction of the number of steps by a factor 10 compared to the additive cell smoother for the Laplacian, which makes it almost competitive [76]. Finally, the iteration counts are independent of the polynomial degree, making the scheme attractive for higher degrees. Moreover, we point out that this smoother also allows for the solution of a Stokes problem in four iteration steps [39].

Table 2 Fractional GMRES iterations, preconditioned by h-MG with multiplicative Schwarz smoothers on vertex patches

Levels	Convergence steps				Colors	Levels	Convergence steps				Colors
2D	$k=3$	$k=4$	$k=7$	$k=10$		2D	$k=3$	$k=4$	$k=7$	$k=10$	
7	2.5	2.5	2.1	2.1	8	7	2.9	2.9	2.6	2.5	17
8	2.5	2.5	2.1	2.0	8	8	2.9	2.9	2.6	2.5	17
9	2.5	2.4	2.1	2.0	8	9	2.9	2.9	2.6	2.5	17
10	2.5	2.4	2.0	2.0	8	10	2.9	2.9	2.6	2.4	17
3D	$k=3$	$k=4$	$k=7$	$k=10$		3D	$k=3$	$k=4$	$k=7$	$k=10$	
3	2.4	2.5	2.1	1.8	16	3	2.6	2.7	2.4	2.4	35
4	2.4	2.5	2.1	1.9	16	4	2.8	2.8	2.5	2.4	49
5	2.4	2.5	2.1	1.9	16	5	2.8	2.8	2.5	2.4	51
6	2.4	2.5	2.1	1.9	16	6	2.8	2.8	2.5	2.4	52

Based on minimal coloring (left) and graph coloring (right) with a relative solver tolerance of 10^{-8}

5.2 General Geometry

As soon as the mesh cells are not Cartesian anymore, the special structure of separable operators in (6) is lost and the inverse cannot be computed according to (7). In this case, we have two options: solving the local problems iteratively, as in [6], or approximately. A possible approximation which recovers the situation of the previous subsection consists of replacing a non-Cartesian mesh cell by an approximating (hyper-)rectangle, then inverting the separable differential operator on the rectangle (omitting the prefix hyper from here on).

Such a surrogate rectangle can be obtained from the following procedure: first, we compute the arc length of all edges. From these, we obtain the length of the rectangle in each of its natural directions by averaging over all parallel edges (in a topological sense). Thus, the geometry of the rectangle is determined up to its position and orientation in space. Given the fact that the Laplacian is invariant under translation and rotation, these do not matter and we can choose a rectangle centered at the origin with edges parallel to the coordinate directions. Different differential operators may require different approximations here.

The convergence theory of Schwarz methods allows for inexact local solvers as long as they are spectrally equivalent. Naturally, the deviation from exactness enters into the convergence speed of the method. Additionally, inexact local solvers can amplify the solution, such that a smaller relaxation parameter may be necessary. This is exhibited in Table 3, where we compare the efficiency of multigrid with exact local solvers and the method with surrogate rectangles as described above. We see that a reduction of the relaxation parameter $\widehat{\omega} = 0.7$ for exact local solvers to $\widehat{\omega} = 0.49$ is necessary for robust convergence. We point out though, that while the inexact methods need more iteration steps, they are much faster than exact inverses, since they use the Kronecker representation (7) of the approximate inverse. For instance, the setup cost is 3000 times higher, with a growing gap for higher polynomial degrees.

Table 3 Fractional CG iterations with addditive cell-based Schwarz smoothers, exact as well as inexact local solution with varying damping factors $\hat{\omega}$

Levels	Convergence steps to 10^{-8}					
2D	*exact* ($\hat{\omega} = 0.7$)	$\hat{\omega} = 0.35$	$\hat{\omega} = 0.42$	$\hat{\omega} = 0.49$	$\hat{\omega} = 0.56$	$\hat{\omega} = 0.63$
4	17.8	28.4	24.8	24.3	30.8	>100
5	17.3	27.1	23.9	23.8	40.7	>100
6	17.2	26.8	23.7	23.9	58.1	>100
3D	*exact* ($\hat{\omega} = 0.7$)	$\hat{\omega} = 0.35$	$\hat{\omega} = 0.42$	$\hat{\omega} = 0.49$	$\hat{\omega} = 0.56$	$\hat{\omega} = 0.63$
2	20.6	31.8	28.5	25.8	25.0	28.5
3	20.6	33.3	29.1	26.5	27.4	74.8
4	20.6	32.4	28.6	26.6	47.0	>100

Two pre- and post-smoothing steps are used, respectively, and the polynomial degree is $k = 4$

5.3 Linear Elasticitiy

In order to provide an outlook on how to apply this concept to more general problems, we consider linear elasticity, namely the Lamé-Navier equations, with the bilinear form

$$a(u, v) = 2\mu\big(\varepsilon(u), \varepsilon(v)\big) + \lambda\big(\nabla \cdot u, \nabla \cdot v\big). \tag{9}$$

Here, $\varepsilon(u) = \frac{1}{2}(\nabla u + \nabla u^{\mathsf{T}})$ is the strain tensor of the displacement field u and $\big(\cdot, \cdot\big)$ denote the appropriate DG discretization with interior penalty terms.

Consider a Cartesian vertex patch, that is, a patch with all faces aligned with the coordinate planes and with tensor product shape functions on each cell. As before, let M_k be the one-dimensional mass matrix in direction k and L_k the matrix representing the Laplacian including all face terms introduced by the interior penalty formulation. Furthermore, let G_k be the matrix associated to the first derivative, again including the DG interface terms which arise in products of the form $G_k^{\mathsf{T}} \otimes G_l$. With these notions and the three-dimensional Laplacian

$$L = M_3 \otimes M_2 \otimes L_1 + M_3 \otimes L_2 \otimes M_1 + L_3 \otimes M_2 \otimes M_1, \tag{10}$$

we can write the bilinear form $a(.,.)$ on the patch in matrix form

$$\begin{aligned} A_p = \mu &\begin{bmatrix} L + M_3 \otimes M_2 \otimes L_1 & M_3 \otimes G_2^{\mathsf{T}} \otimes G_1 & G_3^{\mathsf{T}} \otimes M_2 \otimes G_1 \\ M_3 \otimes G_2 \otimes G_1^{\mathsf{T}} & L + M_3 \otimes L_2 \otimes M_1 & G_3^{\mathsf{T}} \otimes G_2 \otimes M_1 \\ G_3 \otimes M_2 \otimes G_1^{\mathsf{T}} & G_3 \otimes G_2^{\mathsf{T}} \otimes M_1 & L + L_3 \otimes M_2 \otimes M_1 \end{bmatrix} \\ &+ \lambda \begin{bmatrix} M_3 \otimes M_2 \otimes L_1 & M_3 \otimes G_2 \otimes G_1^{\mathsf{T}} & G_3 \otimes M_2 \otimes G_1^{\mathsf{T}} \\ M_3 \otimes G_2^{\mathsf{T}} \otimes G_1 & M_3 \otimes L_2 \otimes M_1 & G_3 \otimes G_2^{\mathsf{T}} \otimes M_1 \\ G_3^{\mathsf{T}} \otimes M_2 \otimes G_1 & G_3^{\mathsf{T}} \otimes G_2 \otimes M_1 & L_3 \otimes M_2 \otimes M_1 \end{bmatrix} \end{aligned} \tag{11}$$

Clearly, this matrix lacks the simple structure of Kronecker products we employed in the previous subsections. Nevertheless, we have Korn's inequality [10], and thus the block diagonal of the left matrix is spectrally equivalent to the matrix itself. Consequently, we expect that

$$\tilde{A}_p = \mu \begin{bmatrix} L + M_3 \otimes M_2 \otimes L_1 & & \\ & L + M_3 \otimes L_2 \otimes M_1 & \\ & & L + L_3 \otimes M_2 \otimes M_1 \end{bmatrix}, \tag{12}$$

which has the desired Kronecker product structure, is a good local solver. Indeed, Table 4 confirms this expectation. Iteration counts remain almost constant over a

Table 4 Solver performance depending on level and polynomial degree k

Levels	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$	$k = 9$
3	–	–	–	–	–	–	–	4.0	4.1
4	–	–	–	3.7	3.9	3.9	4.0	4.1	4.1
5	–	3.7	3.7	3.7	3.8	3.9	3.9	3.9	3.9
6	5.1	3.7	3.8	3.6	3.8	3.9	3.9	3.9	3.9
7	5.2	3.8	3.9	3.7	3.7	3.8	3.7	3.8	3.8
8	5.5	3.9	3.9	3.8	3.8	3.7	3.8	–	–
9	5.4	3.9	4.0	–	–	–	–	–	–

CG iterations to reduce the residual by 10^8 preconditioned by h-MG with multiplicative vertex patch smoother and approximate local solvers $\tilde{A}_p^{-1}$. Only levels with 10^4 to 10^7 degrees of freedom are shown. $\mu = 1$, $\lambda = 1$ and the coarse grid consists of 2×2 cells

Table 5 Solver performance depending on Lamé parameters μ and λ

	(μ, λ)					
Levels	(100, 1)	(10, 1)	(1, 1)	(1, 5)	(1, 10)	(1, 25)
6	3.4	3.4	3.6	6.3	19.5	>200
7	3.6	3.6	3.7	6.2	19.9	>200
8	3.7	3.7	3.8	6.0	20.2	>200
9	3.8	3.8	3.9	5.9	20.2	>200
10	3.8	3.8	3.9	5.9	20.3	>200
11	3.8	3.8	3.9	5.8	19.9	>200

CG iterations to reduce the residual by 10^8 preconditioned by h-MG with block-diagonal smoother. Shape functions of degree $k = 4$ are used. The coarse grid consists of 2×2 cells

wide range of mesh levels and polynomial degrees. Comparing to Table 2, we lose less than a factor two, typically requiring 4 steps instead of 3.

While Korn's inequality helped us with the left matrix in (11), the matrix corresponding to the "grad-div" term in the Lamé–Navier equations has a nontrivial kernel and thus its inverse cannot be approximated by a block diagonal. We confirm this in Table 5. After augmenting $\tilde{A}_p$ by the diagonal terms of the grad-div matrix, we vary μ and λ. As expected, iteration counts increase when $\lambda \gg \mu$ to the point, where the method becomes infeasible.

The case $\lambda \gg \mu$ corresponds to an almost incompressible material. Thus, this behavior has to be addressed from two sides. First, the discretization must be suitable [34]. Then, the local solvers must be able to reduce the divergence sufficiently. Here, we have to find ways to implement a smoother like in [39] in an efficient way. Its structure prevents us from utilizing the tensor product techniques, namely the fast diagonalization method, used so far.

As an outlook, we describe a solution approach for two dimensions, which has been developed in a recent bachelor's thesis [64]. The diagonal blocks of the matrix A_p are

$$A_1 = (2\mu + \lambda)\, M_2 \otimes L_1 + \mu L_2 \otimes M_1, \quad A_2 = \mu M_2 \otimes L_1 + (2\mu + \lambda)\, L_2 \otimes M_1. \tag{13}$$

Both A_1 and A_2 admit a fast diagonalization, for instance

$$A_2^{-1} = \big(Q_2 \otimes Q_1\big)\big(I_2 \otimes \Lambda_1 + \Lambda_2 \otimes I_1\big)^{-1}\big(Q_2 \otimes Q_1\big)^{\mathsf{T}}. \tag{14}$$

Given the off-diagonal block $B = \mu G_2^{\mathsf{T}} \otimes G_1 + \lambda G_2 \otimes G_1^{\mathsf{T}}$, the Schur complement of A_p is

$$S = A_1 - B^{\mathsf{T}} A_2^{-1} B. \tag{15}$$

While this is not a sum of Kronecker products, Kronecker singular value decomposition (KSVD), see [72, 73], can be utilized to construct an approximation of the Schur complement which is fast diagonalizable. We proceed as follows:

A.1 compute the fast diagonalizations of A_1 and A_2
A.2 compute the rank-ρ_Λ KSVD of the inverse diagonal matrix in Eq. 14

$$\big(I \otimes \Lambda^{(1)} + \Lambda^{(2)} \otimes I\big)^{-1} \approx \sum_{i=1}^{\rho_\Lambda} C_i \otimes D_i \tag{16}$$

A.3 compute the rank-2 KSVD

$$\widehat{S} := E_1 \otimes F_1 + E_2 \otimes F_2 \approx \widetilde{S} \tag{17}$$

of the approximate Schur complement

$$\widetilde{S} := A_1 - B^{\mathsf{T}} \left[\sum_{i=1}^{\rho_\Lambda} Q_2 C_i^{-1} Q_2^{\mathsf{T}} \otimes Q_1 D_i^{-1} Q_1^{\mathsf{T}} \right] B \tag{18}$$

A.4 compute the fast diagonalization of $\widehat{S}$.

Then, Gaussian block elimination provides an approximate inverse

$$A_p^{-1} \approx \begin{bmatrix} I & -A_1^{-1}B \\ 0 & I \end{bmatrix} \begin{bmatrix} A_1^{-1} & 0 \\ 0 & \widehat{S}^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -B^{\mathsf{T}} A_1^{-1} & I \end{bmatrix}. \tag{19}$$

Implementation and evaluation of these smoothers are still work in progress, but the thesis [64] suggests fast and robust convergence at least in a finite difference context.

The take-home message from this section is that an efficient approximate solution of the local problems in Schwarz smoothers is possible using low-rank tensor representations and can be achieved with effort similar to a matrix-free operator application in the best case. Finding such low-rank representations is nevertheless highly dependent on the differential equation and geometry. Further investigation will be directed in particular at dealing with the grad-div operator.

6 High-performance Simulations of Incompressible Flows

Computational fluid dynamics (CFD) simulations of turbulent flows at large Reynolds number, e.g., $\mathrm{Re} > 10^6$, are among those problems that typically require a huge amount of computational resources in order to resolve the turbulent flow structures in space and time, and have been addressed as an application by the ExaDG project. The underlying model problem is given by the incompressible Navier–Stokes equations

$$\frac{\partial u}{\partial t} + \nabla \cdot (u \otimes u) - \nu \nabla^2 u + \nabla p = f \; , \tag{20}$$

$$\nabla \cdot u = 0 \; . \tag{21}$$

Scale-resolving simulations for engineering applications typically involve beyond $\mathcal{O}(10^{10} - 10^{11})$ unknowns (DoFs) and $\mathcal{O}(10^5 - 10^7)$ time steps. High-performance implementations for this type of problem are therefore of paramount importance for the CFD community. It is important to stress that implementing a given algorithm optimally for a given hardware, i.e., an implementation that performs close to the hardware limits, is only one step to achieve the goal of providing efficient flow solvers for engineering problems as emphasized in the introduction. While the previous sections discussed the second and third term in Eq. (1), namely the performance of matrix-free evaluation routines and fast multigrid solvers for high-order discretizations, we now also include discretization aspects into the discussion. The implementation makes use of the fast matrix-free evaluation routines and multigrid solvers discussed in previous sections.

We use a method of lines approach with high-order DG discretizations in space and splitting methods with BDF time integration. Splitting methods separate the solution of the incompressible Navier–Stokes equations into sub-problems such as a Poisson equation for the pressure and a (convection–)diffusion equation for the velocity and are among the most efficient solvers currently known. In a first contribution [45], we highlighted that previous discretization methods lack robustness, on the one hand in the limit of small time step sizes, and on the other hand in under-resolved scenarios where the spatial discretization only resolves the largest scales of the flow. The stability problem for small time step sizes has been addressed in detail in [21] where we found that a proper DG discretization of velocity-pressure coupling terms is essential to achieve robustness at small time steps. In [23], we presented the first high-order DG incompressible flow solver that is robust in the under-resolved regime and that relies completely on efficient matrix-free evaluation routines. The developed discretization approach is attractive as it provides a generic solver for turbulent flow simulations that is robust and accurate without the use of explicit turbulence models. Such a technique is known as implicit large-eddy simulation in the literature and has the advantage that it does not require turbulence model parameters. While this property of high-order DG discretizations is already known from discontinuous Galerkin discretizations of the compressible Navier–Stokes equations, the work [23] has been the first demonstrating this property for DG discretizations of the incompressible Navier–Stokes equations. The key ingredient for a robust high-order, L_2-conforming DG discretization for incompressible flows turns out to be the use of consistent stabilization terms that enforce the divergence-free constraint and inter-element mass conservation in a weak sense. These requirements can also be included into the finite element function spaces by using so-called $H(\text{div})$-conforming (normal-continuous) discretizations that are exactly (pointwise) divergence-free by using Raviart–Thomas elements. As investigated in detail in [26], such an approach has indeed very similar discretization properties when compared with the stabilized L_2-conforming approach in practically relevant, under-resolved application scenarios. The model has been extended to moving meshes in [20].

A detailed performance analysis has been undertaken in [22] where we discuss the incompressible flow solver w.r.t. its efficiency according to Eq. (1). Based on this efficiency model, we have then compared matrix-free solvers based on incompressible and compressible Navier–Stokes formulations in [24] for under-resolved turbulent incompressible flows. The compressible solver uses explicit time integration and therefore only requires one operator evaluation in every Runge–Kutta stage as opposed to the incompressible solver involving the solution of linear system of equations such as a pressure Poisson equation within every time step. Simple explicit solvers are often considered efficient due to better parallel scalability since implicit Krylov solvers with multigrid preconditioning involve global communication. However, our work shows a significant performance advantage of the incompressible formulation over the compressible one on the node-level for sufficient workload. Albeit speed-up factors are higher, it is difficult to achieve a performance advantage for the algorithmically simple, explicit-in-time compressible solver in the strong-scaling limit in terms of absolute run time. In our experience, the potential to outperform an implicit solver at some point in the strong-scaling limit has not materialized. We see it as a future challenge to devise optimal PDE solvers providing good performance over a wide range of problems and hardware platforms due to this high degree of interdisciplinarity.

We have applied this solver framework to conduct direct numerical simulations of turbulent channel flow in [45], the first direct numerical simulation of the turbulent flow over a periodic hill at Re $\approx 10^4$ in [46], and to large-eddy simulation of the FDA benchmark nozzle problem in [25]. Furthermore, we have developed multiscale wall modeling approaches that allow to use the proposed highly efficient schemes also for industrial cases with even higher Reynolds numbers than what is feasible for wall-resolved large eddy simulation [47].

Here, we show performance results obtained on SuperMUC-NG with Intel Skylake CPUs. We study the three-dimensional Taylor–Green vortex problem as a standard benchmark to assess the accuracy and computational efficiency of incompressible turbulent flow solvers. Regarding discretization accuracy and from a physical point of view, the quantity of interest is the kinetic energy dissipation rate shown in Fig. 12 as a function of time $0 \leq t \leq T = 20$ for increasing Reynolds numbers Re $= 100, 200, 400, 800, 1600, 3000, 10{,}000, \infty$. The first direct numerical simulation for the Re $= 1600$ case with a high-order DG scheme of the incompressible Navier–Stokes equations with a resolution of 1024^3 and polynomial degrees $k = 3, 7$ has been shown in [22]. Here, we show results for effective resolutions up to 3072^3 (corresponding to $0.99 \cdot 10^{11}$ DoFs) for the highest Reynolds number cases. Despite these fine resolutions, grid-converged results are achieved only up to Re $= 3000$. The inviscid problem (Re $= \infty$) is most challenging, and the results in Fig. 12 suggest that even finer resolutions are required for grid-convergence, a goal that might be achievable in the foreseeable future. The largest problem with $0.99 \cdot 10^{11}$ DoFs involved $6.6 \cdot 10^4$ time steps and required 11.4 h of wall time on 152,064 cores. In terms of degrees of freedom solved per time step per core, this results in a throughput of 1.05 MDoFs/s/core.

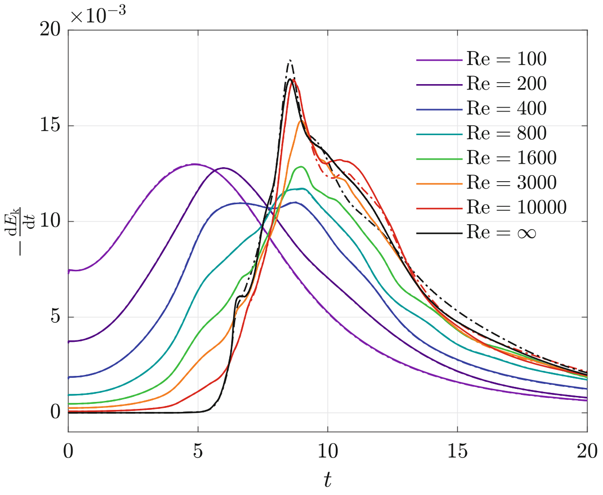


Fig. 12 Taylor–Green vortex: Kinetic energy dissipation rates for two different problem sizes (fine mesh as solid line and coarse mesh as dashed-dotted line) for each Re number: The polynomial degree is $k = 3$ and the effective resolutions $N_{\text{eff}} = (N_{\text{ele,1d}}(k+1))^3$ considered are $N_{\text{eff}} = 64^3, 128^3$ for Re $= 100$, $N_{\text{eff}} = 128^3, 256^3$ for Re $= 200, 400$, $N_{\text{eff}} = 256^3, 512^3$ for Re $= 800$, $N_{\text{eff}} = 1024^3, 2048^3$ for Re $= 1600$, and $N_{\text{eff}} = 2048^3, 3072^3$ for Re $= 3000, 10000, \infty$

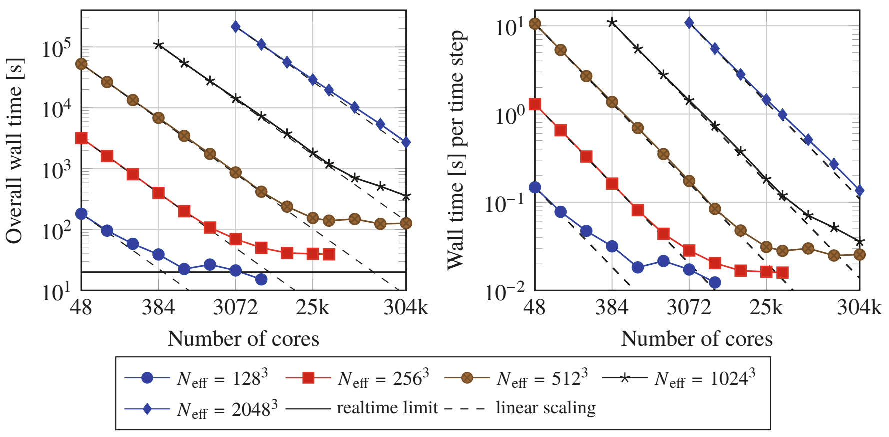


Fig. 13 Scaling analysis for incompressible flow solver on 3D Taylor–Green vortex with polynomial degree $k = 3$ at Re $= 1600$ and spatial resolutions of 128^3, 256^3, 512^3, 1024^3, 2048^3

Figure 13 shows strong scaling results for the TGV problem at Re $= 1600$ for effective resolutions of 128^3, 256^3, 1024^3, 2048^3 and polynomial degree $k = 3$. We assess strong scalability in terms of absolute run times for the whole application (including mesh-generation, setup of data structures, solvers, preconditioners, and postprocessing) rather than normalized speed-up factors as the aim of strong scalability is not only reducing but also minimizing time-to-solution, i.e., demonstrating strong-scalability of a code with poor serial performance is meaningless. The results in Fig. 13 reveal that we are able to perform the TGV simulations in realtime ($t_{\text{wall}} \leq T = 20$s) for spatial resolutions up to 128^3. These numbers can be considered outstanding and we are not aware of other high-order DG solvers achieving this performance, see also the discussions in [22, 24]. The minimum wall time in the strong-scaling limit increases on finer meshes due to more time steps (the time step size is restricted according to the CFL condition, $\Delta t \sim 1/h$, for the mixed explicit–implicit splitting solver used here). For this reason, we also show strong scalability in terms of the wall time per time step, to allow extrapolations of how many time steps can be solved within a given wall time limit which is the typical use case for large-eddy and direct numerical simulations of turbulent flows. In this metric, the curves level off around $0.02 - 0.03$ s of wall time per time step, independently of the spatial resolution. The SuperMUC-NG machine with $3 \cdot 10^5$ cores is too small to show the strong scaling limit for the largest problem size with 2048^3 resolution considered here. A parallel efficiency of 80.6% is achieved with a speed-up factor of 79.8 when scaling from 3072 cores to 304,128 cores.

7 hyper.deal: Extending the Matrix-Free Kernels to Higher Dimensions

The matrix-free kernels developed within the ExaDG project have been implemented in a recursive manner which enables compilation with arbitrary spatial dimension. In order to be compatible with the mesh infrastructure of deal.II which is restricted to dimensions up to 3, we have developed schemes working on a tensor product of two deal.II meshes. This allows extension to 2+2, 2+3, and 3+3 dimensions. The corresponding framework is currently under development as the deal.II-extension `hyper.deal` [59].

The major application that we have in mind are kinetic problems in phase space where we use the tensor product of a spatial and a velocity mesh. However, other applications might arise such as parameter-dependent flow problems. Table 6 gives an overview of computational times on a six-dimensional Vlasov–Poisson problem, which involves an advection in the 6D space of the particle density in x and v space and the solution of a 3D Poisson equation for finding the electric potential that in turn specifies the electric field that transports the density field (cf. [44] for the same application tackled with a semi-Lagrangian solver).

Figure 14 lists the throughput of the matrix-free evaluation of cell integrals for the multi-dimensional advection in three to six spatial dimensions for polynomial degrees $k = 2, 3, 4, 5$ for AVX2 and AVX-512 vectorization over elements, respectively, without any application-specific tuning at this stage. While throughput is very good in 3D and 4D as well as 5D up to $k = 4$, performance drops significantly in 6D because the local arrays in sum factorization exhaust caches,

Table 6 Contributions to run time on 6D Vlasov–Poisson system on 320 cores with 8.6 billion spatial DoFs over 42 time steps

Category	6D advect total (of which MPI exchange)	integrate v	3D Poisson + electric field
time [s]	560 (130)	13.0	35.9

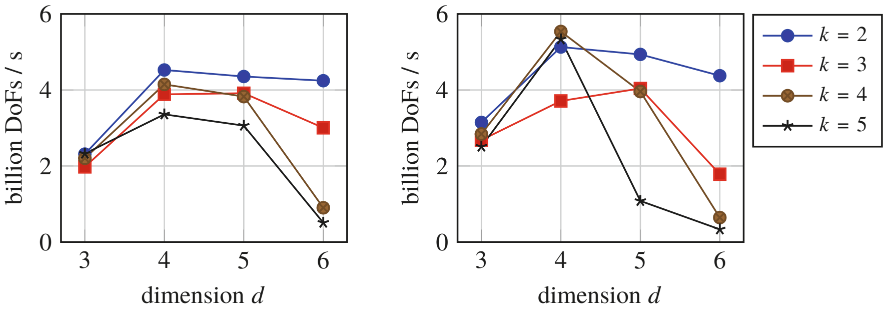


Fig. 14 Throughput of cell term for advection as a function of the spatial dimension on Intel Skylake with 4-wide vectorization (left) and 8-wide vectorization (right)

especially with AVX-512. Vectorization strategies within an element [50] are currently under development.

8 Outlook

Our work in the ExaDG project presented in this text has resulted in a highly competitive finite element framework. We have demonstrated excellent performance both for the pure operator evaluation, demonstrated e.g. by the CEED benchmark problems, as well as on an application level in computational fluid dynamics. We plan to engage in benchmarking also in the future to establish best-practices for the high-order finite element community. Furthermore, the evolving hardware landscape requires a continued effort, with increasing pressure to additional performance improvements on throughput architectures such as GPUs and FPGAs. In addition, we plan to extend our hybrid *hp*-multigrid framework to also handle *hp*-adaptive meshes. Finally, while the results from the Schwarz-based multigrid smoothers are very promising from a mathematical point of view, further steps are necessary to make them perform optimally on massively parallel hardware, and it is not yet clear how an optimal implementation compares in time-to-solution against the simpler Chebyshev-based ingredients we have considered on the large scale so far.

References

1. Alzetta, G., Arndt, D., Bangerth, W., Boddu, V., Brands, B., Davydov, D., Gassmoeller, R., Heister, T., Heltai, L., Kormann, K., Kronbichler, M., Maier, M., Pelteret, J.P., Turcksin, B., Wells, D.: The `deal.II` library, version 9.0. J. Numer. Math. **26**(4), 173–184 (2018). https://doi.org/10.1515/jnma-2018-0054
2. Anderson, R., Barker, A., Bramwell, J., Cerveny, J., Dahm, J., Dobrev, V., Dudouit, Y., Fisher, A., Kolev, T., Stowell, M., Tomov, V.: MFEM: modular finite element methods (2019). mfem.org
3. Antonietti, P.F., Sarti, M., Verani, M., Zikatanov, L.T.: A uniform additive Schwarz preconditioner for high-order discontinuous Galerkin approximations of elliptic problems. J. Sci. Comput. **70**(2), 608–630 (2017). https://doi.org/10.1007/s10915-016-0259-9
4. Arndt, D., Bangerth, W., Davydov, D., Heister, T., Heltai, L., Kronbichler, M., Maier, M., Pelteret, J.-P., Turcksin, B., Wells, D.: The deal.II finite element library: Design, features, and insights. Comput. Math. Appl. (2020). https://doi.org/10.1016/j.camwa.2020.02.022
5. Bastian, P., Engwer, C., Fahlke, J., Geveler, M., Göddeke, D., Iliev, O., Ippisch, O., Milk, R., Mohring, J., Müthing, S., Ohlberger, M., Ribbrock, D., Turek, S.: Hardware-based efficiency advances in the EXA-DUNE project. In: Bungartz, H.J., Neumann, P., Nagel, W.E. (eds.) Software for Exascale computing—SPPEXA 2013-2015, pp. 3–23. Springer, Cham (2016)
6. Bastian, P., Müller, E.H., Müthing, S., Piatkowski, M.: Matrix-free multigrid block-preconditioners for higher order discontinuous Galerkin discretisations. J. Comput. Phys. **394**, 417–439 (2019). https://doi.org/10.1016/j.jcp.2019.06.001
7. Bauer, S., Drzisga, D., Mohr, M., Rüde, U., Waluga, C., Wohlmuth, B.: A stencil scaling approach for accelerating matrix-free finite element implementations. SIAM J. Sci. Comput. **40**(6), C748–C778 (2018)

8. Bergen, B., Hülsemann, F., Rüde, U.: Is 1.7×10^{10} unknowns the largest finite element system that can be solved today? In: Proceeding of ACM/IEEE Conference Supercomputing (SC'05), pp. 5:1–5:14 (2005). https://doi.org/10.1109/SC.2005.38
9. Brandt, A.: Multi-level adaptive solutions to boundary-value problems. Math. Comput. **31**, 333–390 (1977). https://doi.org/10.1090/S0025-5718-1977-0431719-X
10. Brenner, S.C.: Korn's inequalities for piecewise H^1 vector fields. Math. Comput. **73**(247), 1067–1087 (2004)
11. Brown, J.: Efficient nonlinear solvers for nodal high-order finite elements in 3D. J. Sci. Comput. **45**(1–3), 48–63 (2010)
12. Cantwell, C.D., Sherwin, S.J., Kirby, R.M., Kelly, P.H.J.: Form h to p efficiently: Selecting the optimal spectral/hp discretisation in three dimensions. Math. Model. Nat. Phenom. **6**, 84–96 (2011)
13. Cantwell, C.D., Moxey, D., Comerford, A., Bolis, A., Rocco, G., Mengaldo, G., De Grazia, D., Yakovlev, S., Lombard, J.E., Ekelschot, D., Jordi, B., Xu, H., Mohamied, Y., Eskilsson, C., Nelson, B., Vos, P., Biotto, C., Kirby, R.M., Sherwin, S.J.: Nektar++: An open-source spectral/hp element framework. Comput. Phys. Comm. **192**, 205–219 (2015). https://doi.org/10.1016/j.cpc.2015.02.008
14. Charrier, D.E., Hazelwood, B., Tutlyaeva, E., Bader, M., Dumbser, M., Kudryavtsev, A., Moskovsky, A., Weinzierl, T.: Studies on the energy and deep memory behaviour of a cache-oblivious, task-based hyperbolic PDE solver. Int. J. High Perf. Comput. Appl. **33**(5), 973–986 (2019). https://doi.org/10.1177/1094342019842645
15. Clevenger, T.C., Heister, T., Kanschat, G., Kronbichler, M.: A flexible, parallel, adaptive geometric multigrid method for FEM. Technical report, arXiv:1904.03317 (2019)
16. Davydov, D., Kronbichler, M.: Algorithms and data structures for matrix-free finite element operators with MPI-parallel sparse multi-vectors. ACM Trans. Parallel Comput. (2020). https://doi.org/10.1145/3399736
17. Davydov, D., Heister, T., Kronbichler, M., Steinmann, P.: Matrix-free locally adaptive finite element solution of density-functional theory with nonorthogonal orbitals and multigrid preconditioning. Phys. Status Solidi B: Basic Solid State Phys. **255**(9), 1800069 (2018). https://doi.org/10.1002/pssb.201800069
18. Davydov, D., Pelteret, J.P., Arndt, D., Kronbichler, M., Steinmann, P.: A matrix-free approach for finite-strain hyperelastic problems using geometric multigrid. Int. J. Numer. Meth. Eng. (2020). https://doi.org/10.1002/nme.6336
19. Deville, M.O., Fischer, P.F., Mund, E.H.: High-order Methods for Incompressible Fluid Flow, vol. 9. Cambridge University, Cambridge (2002)
20. Fehn, N., Heinz, J., Wall, W.A., Kronbichler, M.: High-order arbitrary Lagrangian-Eulerian discontinuous Galerkin methods for the incompressible Navier-Stokes equations. Technical report, arXiv:2003.07166 (2020).
21. Fehn, N., Wall, W.A., Kronbichler, M.: On the stability of projection methods for the incompressible Navier–Stokes equations based on high-order discontinuous Galerkin discretizations. J. Comput. Phys. **351**, 392–421 (2017). https://doi.org/10.1016/j.jcp.2017.09.031
22. Fehn, N., Wall, W.A., Kronbichler, M.: Efficiency of high-performance discontinuous Galerkin spectral element methods for under-resolved turbulent incompressible flows. Int. J. Numer. Meth. Fluids **88**(1), 32–54 (2018). https://doi.org/10.1002/fld.4511
23. Fehn, N., Wall, W.A., Kronbichler, M.: Robust and efficient discontinuous Galerkin methods for under-resolved turbulent incompressible flows. J. Comput. Phys. **372**, 667–693 (2018). https://doi.org/10.1016/j.jcp.2018.06.037
24. Fehn, N., Wall, W.A., Kronbichler, M.: A matrix-free high-order discontinuous Galerkin compressible Navier–Stokes solver: a performance comparison of compressible and incompressible formulations for turbulent incompressible flows. Int. J. Numer. Meth. Fluids **89**(3), 71–102 (2019). https://doi.org/10.1002/fld.4683

25. Fehn, N., Wall, W.A., Kronbichler, M.: Modern discontinuous Galerkin methods for the simulation of transitional and turbulent flows in biomedical engineering: a comprehensive LES study of the FDA benchmark nozzle model. Int. J. Numer. Meth. Biomed. Eng. **35**(12), e3228 (2019). https://doi.org/10.1002/cnm.3228
26. Fehn, N., Kronbichler, M., Lehrenfeld, C., Lube, G., Schroeder, P.W.: High-order DG solvers for under-resolved turbulent incompressible flows: a comparison of L^2 and $H(\text{div})$ methods. Int. J. Numer. Meth. Fluids **91**(11), 533–556 (2019). https://doi.org/10.1002/fld.4763
27. Fehn, N., Munch, P., Wall, W.A., Kronbichler, M.: Hybrid multigrid methods for high-order discontinuous Galerkin discretizations. J. Comput. Phys. (2020). https://doi.org/10.1016/j.jcp.2020.109538
28. Fischer, P., Kerkemeier, S., Peplinski, A., Shaver, D., Tomboulides, A., Min, M., Obabko, A., Merzari, E.: Nek5000 Web page (2019). https://nek5000.mcs.anl.gov
29. Fischer, P., Min, M., Rathnayake, T., Dutta, S., Kolev, T., Dobrev, V., Camier, J.S., Kronbichler, M., Warburton, T., Świrydowicz, K., Brown, J.: Scalability of high-performance PDE solvers. Int. J. High Perf. Comput. Appl. (2020). https://doi.org/10.1177/1094342020915762
30. Gholami, A., Malhotra, D., Sundar, H., Biros, G.: FFT, FMM, or multigrid? A comparative study of state-of-the-art Poisson solvers for uniform and nonuniform grids in the unit cube. SIAM J. Sci. Comput. **38**(3), C280–C306 (2016). https://doi.org/10.1137/15M1010798
31. Gmeiner, B., Rüde, U., Stengel, H., Waluga, C., Wohlmuth, B.: Towards textbook efficiency for parallel multigrid. Numer. Math.-Theory Me. Appl. **8**(1), 22–46 (2015)
32. Gmeiner, B., Huber, M., John, L., Rüde, U., Wohlmuth, B.: A quantitative performance study for Stokes solvers at the extreme scale. J. Comput. Sci. **17**, 509–521 (2016). https://doi.org/10.1016/j.jocs.2016.06.006. http://www.sciencedirect.com/science/article/pii/S1877750316301077. Recent Advances in Parallel Techniques for Scientific Computing
33. Hager, G., Wellein, G.: Introduction to High Performance Computing for Scientists and Engineers. CRC Press, Boca Raton (2011)
34. Hansbo, P., Larson, M.G.: Discontinuous Galerkin methods for incompressible and nearly incompressible elasticity by Nitsche's method. Comput. Methods Appl. Mech. Eng. **191**, 1895–1908 (2002)
35. Ibeid, H., Olson, L., Gropp, W.: FFT, FMM, and multigrid on the road to exascale: performance challenges and opportunities. J. Parallel Distrib. Comput. **136**, 63–74 (2020). https://doi.org/10.1016/j.jpdc.2019.09.014
36. Janssen, B., Kanschat, G.: Adaptive multilevel methods with local smoothing for H^1- and H^{curl}-conforming high order finite element methods. SIAM J. Sci. Comput. **33**(4), 2095–2114 (2011). https://doi.org/10.1137/090778523
37. Kanschat, G.: Multi-level methods for discontinuous Galerkin FEM on locally refined meshes. Comput. Struct. **82**(28), 2437–2445 (2004). https://doi.org/10.1016/j.compstruc.2004.04.015
38. Kanschat, G.: Robust smoothers for high order discontinuous Galerkin discretizations of advection-diffusion problems. J. Comput. Appl. Math. **218**, 53–60 (2008). https://doi.org/10.1016/j.cam.2007.04.032
39. Kanschat, G., Mao, Y.: Multigrid methods for $\mathbf{H}^{\text{div}}$-conforming discontinuous Galerkin methods for the Stokes equations. J. Numer. Math. **23**(1), 51–66 (2015). https://doi.org/10.1515/jnma-2015-0005
40. Kempf, D., Hess, R., Müthing, S., Bastian, P.: Automatic code generation for high-performance discontinuous Galerkin methods on modern architectures. Technical report, arXiv:1812.08075 (2018)
41. Knepley, M.G., Brown, J., Rupp, K., Smith, B.F.: Achieving high performance with unified residual evaluation. Technical report, arXiv:1309.1204 (2013)
42. Kormann, K.: A time-space adaptive method for the Schrödinger equation. Commun. Comput. Phys. **20**(1), 60–85 (2016). https://doi.org/10.4208/cicp.101214.021015a

43. Kormann, K., Kronbichler, M.: Parallel finite element operator application: graph partitioning and coloring. In: Proceeding of 7th IEEE International Conference eScience, pp. 332–339 (2011). https://10.1109/eScience.2011.53
44. Kormann, K., Reuter, K., Rampp, M.: A massively parallel semi-Lagrangian solver for the six-dimensional Vlasov–Poisson equation. Int. J. High Perform. Comput. Appl. **33**(5), 924–947 (2019). https://doi.org/10.1177/1094342019834644
45. Krank, B., Fehn, N., Wall, W.A., Kronbichler, M.: A high-order semi-explicit discontinuous Galerkin solver for 3D incompressible flow with application to DNS and LES of turbulent channel flow. J. Comput. Phys. **348**, 634–659 (2017). https://doi.org/10.1016/j.jcp.2017.07.039
46. Krank, B., Kronbichler, M., Wall, W.A.: Direct numerical simulation of flow over periodic hills up to $\mathrm{Re}_h = 10{,}595$. Flow Turbulence Combust. **101**, 521–551 (2018). https://doi.org/10.1007/s10494-018-9941-3
47. Krank, B., Kronbichler, M., Wall, W.A.: A multiscale approach to hybrid RANS/LES wall modeling within a high-order discontinuous Galerkin scheme using function enrichment. Int. J. Numer. Meth. Fluids **90**, 81–113 (2019). https://doi.org/10.1002/fld.4712
48. Kronbichler, M., Allalen, M.: Efficient high-order discontinuous Galerkin finite elements with matrix-free implementations. In: Bungartz, H.J., Kranzlmüller, D., Weinberg, V., Weismüller, J., Wohlgemuth, V. (eds.) Advances and New Trends in Environmental Informatics, pp. 89–110. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-99654-7_7
49. Kronbichler, M., Kormann, K.: A generic interface for parallel cell-based finite element operator application. Comput. Fluids **63**, 135–147 (2012). https://doi.org/10.1016/j.compfluid.2012.04.012
50. Kronbichler, M., Kormann, K.: Fast matrix-free evaluation of discontinuous Galerkin finite element operators. ACM Trans. Math. Softw. **45**(3), 29:1–29:40 (2019). https://doi.org/10.1145/3325864
51. Kronbichler, M., Ljungkvist, K.: Multigrid for matrix-free high-order finite element computations on graphics processors. ACM Trans. Parallel Comput. **6**(1), 2:1–2:32 (2019). https://doi.org/10.1145/3322813
52. Kronbichler, M., Wall, W.A.: A performance comparison of continuous and discontinuous Galerkin methods with fast multigrid solvers. SIAM J. Sci. Comput. **40**(5), A3423–A3448 (2018). https://doi.org/10.1137/16M110455X
53. Kronbichler, M., Schoeder, S., Müller, C., Wall, W.A.: Comparison of implicit and explicit hybridizable discontinuous Galerkin methods for the acoustic wave equation. Int. J. Numer. Meth. Eng. **106**(9), 712–739 (2016). https://doi.org/10.1002/nme.5137
54. Kronbichler, M., Kormann, K., Pasichnyk, I., Allalen, M.: Fast matrix-free discontinuous Galerkin kernels on modern computer architectures. In: Kunkel, J.M., Yokota, R., Balaji, P., Keyes, D.E. (eds.) ISC High Performance 2017, LNCS 10266, pp. 237–255 (2017). https://doi.org/10.1007/978-3-319-58667-013
55. Kronbichler, M., Diagne, A., Holmgren, H.: A fast massively parallel two-phase flow solver for microfluidic chip simulation. Int. J. High Perf. Comput. Appl. **32**(2), 266–287 (2018). https://doi.org/10.1177/1094342016671790
56. Kronbichler, M., Kormann, K., Fehn, N., Munch, P., Witte, J.: A Hermite-like basis for faster matrix-free evaluation of interior penalty discontinuous Galerkin operators. Technical report, arXiv:1907.08492 (2019)
57. Ljungkvist, K.: Matrix-free finite-element computations on graphics processors with adaptively refined unstructured meshes. In: Proceedings of the 25th High Performance Computing Symposium, HPC '17, pp. 1:1–1:12. Society for Computer Simulation International, San Diego (2017). http://dl.acm.org/citation.cfm?id=3108096.3108097
58. Lynch, R.E., Rice, J.R., Thomas, D.H.: Direct solution of partial difference equations by tensor product methods. Numer. Math. **6**, 185–199 (1964). https://doi.org/10.1007/BF01386067
59. Munch, P., Kormann, K., Kronbichler, M.: hyper.deal: An efficient, matrix-free finite-element library for high-dimensional partial differential equations. Technical report, arXiv:2002.08110 (2020)

60. Müthing, S., Piatkowski, M., Bastian, P.: High-performance implementation of matrix-free high-order discontinuous Galerkin methods. Technical report, arXiv:1711.10885 (2017)
61. Orszag, S.A.: Spectral methods for problems in complex geometries. J. Comput. Phys. **37**, 70–92 (1980)
62. Raffenetti, K., Amer, A., Oden, L., Archer, C., Bland, W., Fujita, H., Guo, Y., Janjusic, T., Durnov, D., Blocksome, M., Si, M., Seo, S., Langer, A., Zheng, G., Takagi, M., Coffman, P., Jose, J., Sur, S., Sannikov, A., Oblomov, S., Chuvelev, M., Hatanaka, M., Zhao, X., Fischer, P., Rathnayake, T., Otten, M., Min, M., Balaji, P.: Why is MPI so slow?: Analyzing the fundamental limits in implementing MPI-3.1. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '17, pp. 62:1–62:12. ACM, New York (2017). https://doi.org/10.1145/3126908.3126963
63. Rathgeber, F., Ham, D.A., Mitchell, L., Lange, M., Luporini, F., McRae, A.T.T., Bercea, G.T., Markall, G.R., Kelly, P.H.J.: Firedrake: automating the finite element method by composing abstractions. ACM Trans. Math. Soft. **43**(3), 24:1–24:27 (2017). https://doi.org/10.1145/2998441
64. Schmidt, S.: Fast, tensor-based solution of problems involving incompressibility, Bachelor thesis. Heidelberg University, Heidelberg (2019)
65. Schoeder, S., Kormann, K., Wall, W.A., Kronbichler, M.: Efficient explicit time stepping of high order discontinuous Galerkin schemes for waves. SIAM J. Sci. Comput. **40**(6), C803–C826 (2018). https://doi.org/10.1137/18M1185399
66. Schoeder, S., Kronbichler, M., Wall, W.: Arbitrary high-order explicit hybridizable discontinuous Galerkin methods for the acoustic wave equation. J. Sci. Comput. **76**, 969–1006 (2018). https://doi.org/10.1007/s10915-018-0649-2
67. Schoeder, S., Sticko, S., Kreiss, G., Kronbichler, M.: High-order cut discontinuous Galerkin methods with local time stepping for acoustics. Int. J. Numer. Meth. Eng. (2020). https://doi.org/10.1002/nme.6343
68. Schoeder, S., Wall, W.A., Kronbichler, M.: ExWave: A high performance discontinuous Galerkin solver for the acoustic wave equation. Soft. X **9**, 49–54 (2019). https://doi.org/10.1016/j.softx.2019.01.001
69. Solomonoff, A.: A fast algorithm for spectral differentiation. J. Comput. Phys. **98**(1), 174–177 (1992). https://doi.org/10.1016/0021-9991(92)90182-X
70. Sundar, H., Biros, G., Burstedde, C., Rudi, J., Ghattas, O., Stadler, G.: Parallel geometric-algebraic multigrid on unstructured forests of octrees. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, p. 43. IEEE Computer Society, Silver Spring (2012)
71. Świrydowicz, K., Chalmers, N., Karakus, A., Warburton, T.: Acceleration of tensor-product operations for high-order finite element methods. Int. J. High Perf. Comput. Appl. **33**(4), 735–757 (2019). https://doi.org/10.1177/1094342018816368
72. Van Loan, C.F.: The ubiquitous Kronecker product. J. Comput. Appl. Math. **123**(1–2), 85–100 (2000)
73. Van Loan, C.F., Pitsianis, N.: Approximation with Kronecker products. In: Linear Algebra for Large Scale and Real-time Applications, pp. 293–314. Springer, Berlin (1993)
74. Varga, R.S.: Matrix Iterative Analysis, 2nd edn. Springer, Berlin (2009)
75. Wichmann, K.R., Kronbichler, M., Löhner, R., Wall, W.A.: Practical applicability of optimizations and performance models to complex stencil-based loop kernels in CFD. Int. J. High Perf. Comput. Appl. **33**(4), 602–618 (2019). https://doi.org/10.1177/1094342018774126
76. Witte, J., Arndt, D., Kanschat, G.: Fast tensor product Schwarz smoothers for high-order discontinuous Galerkin methods. Technical report, arXiv:1910.11239 (2019)

Exa-Dune—Flexible PDE Solvers, Numerical Methods and Applications

Peter Bastian, Mirco Altenbernd, Nils-Arne Dreier, Christian Engwer, Jorrit Fahlke, René Fritze, Markus Geveler, Dominik Göddeke, Oleg Iliev, Olaf Ippisch, Jan Mohring, Steffen Müthing, Mario Ohlberger, Dirk Ribbrock, Nikolay Shegunov, and Stefan Turek

Abstract In the EXA-DUNE project we have developed, implemented and optimised numerical algorithms and software for the scalable solution of partial differential equations (PDEs) on future exascale systems exhibiting a heterogeneous massively parallel architecture. In order to cope with the increased probability of hardware failures, one aim of the project was to add flexible, application-oriented resilience capabilities into the framework. Continuous improvement of the underlying hardware-oriented numerical methods have included GPU-based sparse approximate inverses, matrix-free sum-factorisation for high-order discontinuous Galerkin discretisations as well as partially matrix-free preconditioners. On top of that, additional scalability is facilitated by exploiting massive coarse grained parallelism offered by multiscale and uncertainty quantification methods where we have focused on the adaptive choice of the coarse/fine scale and the overlap region as well as the combination of local reduced basis multiscale methods and the multilevel Monte-Carlo algorithm. Finally, some of the concepts are applied in a land-surface model including subsurface flow and surface runoff.

P. Bastian (✉) · S. Müthing
Heidelberg University, Heidelberg, Germany
e-mail: peter.bastian@iwr.uni-heidelberg.de

N.-A. Dreier · C. Engwer · J. Fahlke · R. Fritze · M. Ohlberger
University of Münster, Münster, Germany

M. Altenbernd · D. Göddeke
University of Stuttgart, Stuttgart, Germany

M. Geveler · D. Ribbrock · S. Turek
TU Dortmund, Dortmund, Germany

O. Iliev · J. Mohring · N. Shegunov
Fraunhofer ITWM, Kaiserslautern, Germany

O. Ippisch
TU Clausthal-Zellerfeld, Clausthal-Zellerfeld, Germany

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_9

1 Introduction

In the EXA-DUNE project we extend the Distributed and Unified Numerics Environment (DUNE)[1] [6, 7] by hardware-oriented numerical methods and hardware-aware implementation techniques developed in the (now) FEAT3[2] [55] project to provide an exascale-ready software framework for the numerical solution of a large variety of partial differential equation (PDE) systems with state-of-the-art numerical methods including higher-order discretisation schemes, multi-level iterative solvers, unstructured and locally-refined meshes, multiscale methods and uncertainty quantification, while achieving close-to-peak performance and exploiting the underlying hardware.

In the first funding period we concentrated on the node-level performance as the framework and in particular its algebraic multigrid solver already show very good scalability in MPI-only mode as documented by the inclusion of DUNE's solver library in the High-Q-Club, the codes scaling to the full machine in Jülich at the time, with close to half a million cores. Improving the node-level performance in light of future exascale hardware involved multithreading ("MPI+X") and in particular exploiting SIMD parallelism (vector extensions of modern CPUs and accelerator architectures). These aspects were addressed within the finite element assembly and iterative solution phases. Matrix-free methods evaluate the discrete operator without storing a matrix, as the name implies, and promise to be able to achieve a substantial fraction of peak performance. Matrix-based approaches on the other hand are limited by memory bandwidth (at least) in the solution phase and thus typically exhibit only a small fraction of the peak (GFLOP/s) performance of a node, but decades of research have led to robust and efficient (in terms of number of iterations) iterative linear solvers for practically relevant systems. Importantly, a consideration of matrix-free and matrix-based methods needs to take the order of the method into account. For low-order methods it is imperative that a matrix entry can be recomputed in less time than it takes to read it from memory, to counteract the memory wall problem. This requires to exploit the problem structure as much as possible, i.e., to rely on constant coefficients, (locally) regular mesh structure and linear element transformations [28, 37]. In these cases it is even possible to apply stencil type techniques, like developed in the EXA-STENCIL project [40]. On the other hand, for high-order methods with tensor-product structure the complexity of matrix-free operator evaluation can be much less than that of matrix-vector multiplication, meaning that less floating-point operations have to be performed which at the same time can be executed at a higher rate due to reduced memory pressure and better suitability for vectorization [12, 39, 50]. This makes high-order methods extremely attractive for exascale machines [48, 51].

[1]http://www.dune-project.org/.

[2]http://feast.tu-dortmund.de/.

In the second funding phase we have mostly concentrated on the following aspects:

1. **Asynchronicity and fault tolerance:** High-level C++ abstractions form the basis of transparent error handling using exceptions in a parallel environment, fault-tolerant multigrid solvers as well as communication hiding Krylov methods.
2. **Hardware-aware solvers for PDEs:** We investigated matrix-based sparse-approximate inverse preconditioners including novel machine-learning approaches, vectorization through multiple right-hand sides as well as matrix-free high-order Discontinous Galerkin (DG) methods and partially matrix-free robust preconditioners based on algebraic multigrid (AMG).
3. **Multiscale (MS) and uncertainty quantification (UQ) methods:** These methods provide an additional layer of embarrassingly parallel tasks on top of the efficiently parallelized forward solvers. A challenge here is load balancing of the asynchronous tasks which has been investigated in the context of the localized reduced basis multiscale method and multilevel Monte Carlo methods.
4. **Applications:** We have considered large-scale water transport in the subsurface coupled to surface flow as an application where the discretization and solver components can be applied.

In the community, there is broad consensus on the assumptions about exascale systems that did not change much during the course of this 6 year project. A report by the Exascale Mathematics Working Group to the U.S. Department of Energy's Advanced Scientific Computing Research Program [16] summarises these challenges as follows, in line with [35] and more recently the Exascale Computing Project:[3] (1) The anticipated power envelope of 20 MW implies strong limitations on the amount and organisation of the hardware components, an even stronger necessity to fully exploit them, and eventually even power-awareness in algorithms and software. (2) The main performance difference from peta- to exascale will be through a 100–1000 fold increase in parallelism at the node level, leading to extreme levels of concurrency and increasing heterogeneity through specialised accelerator cores and wide vector instructions. (3) The amount of memory per 'core' and the memory and interconnect bandwidth/latency will only increase at a much smaller rate, hence increasing the demand for lower memory footprints and higher data locality. (4) Finally, hardware failures, and thus the mean-time-between-failure (MTBF), were expected to increase proportionally (or worse) corresponding to the increasing number of components. Recent studies have indeed confirmed this expectation [30], although not at the projected rate. First exascale systems are scheduled for 2020 in China [42], 2021 in the US and 2023 [25] in Europe. Although the details are not yet fully disclosed, it seems that the number of nodes will not be larger than 10^5 and will thus remain in the range of previous machines such as the

[3]https://www.exascaleproject.org/.

BlueGene. The major challenge will thus be to exploit the node level performance of more than 10 TFLOP/s.

The rest of this paper is organized as follows. In Sect. 2 we lay the foundations of asynchronicity and resilience, while Sect. 3 discusses several aspects of hardware-aware and scalable iterative linear solvers. These building blocks will then be used in Sects. 4 and 5 to drive localized reduced basis and multilevel Monte-Carlo methods. Finally, Sect. 6 covers our surface-subsurface flow application.

2 Asynchronicity and Fault Tolerance

As predicted in the first funding period, latency has indeed become a major issue, both within a single node as well as between different MPI ranks. The core concept underlying all latency- and communication-hiding techniques is asynchronicity. This is also crucial to efficiently implement certain local-failure local-recovery methods. Following the DUNE philosophy, we have designed a generic layer that abstracts the use of asynchronicity in MPI from the user. In the following, we first describe this layer and its implementation, followed by representative examples on how to build middleware infrastructure on it, and on its use for s-step Krylov methods and fault tolerance beyond global checkpoint-restart techniques.

2.1 Abstract Layer for Asynchronicity

We first introduce a general abstraction for asynchronicity in parallel MPI applications, which we developed for DUNE. While we integrated these abstractions with the DUNE framework, most of the code can easily be imported into other applications, and is available as a standalone library.

The C++ API for MPI was dropped from MPI-3 since it offered no real advantage over the C bindings, beyond being a simple wrapper layer. Most MPI users coding in C++ are still using the C bindings, writing their own C++ interface/layer, in particular in more generic software frameworks. At the same time the C++11 standard introduced high-level concurrency concepts, in particular the future/promise construct to enable an asynchronous program flow while maintaining value semantics. We adopt this approach as a first principle in our MPI layer to handle asynchronous MPI operations and propose a high-level C++ MPI interface, which we provide in DUNE under the generic interface of `Dune::Communication` and a specific implementation `Dune::MPICommunication`.

An additional issue of the concrete MPI library in conjunction with C++ is the error handling concept. In C++, exceptions are the advocated approach to handle error propagation. As exceptions change the local code path on the, e.g., failing process in a hard fault scenario, exceptions can easily lead to a deadlock. As we discuss later, the introduction of our asynchronous abstraction layer enables global error handling in an exception friendly manner.

In concurrent environments a C++ future decouples values from the actual computation (promise). The program flow can continue while a thread is computing the actual result and promotes this via promise to the future. The MPI C and Fortran interfaces offer asynchronous operations, but in contrast to thread parallel, the user does not specify the operation within the concurrent operation. Actually, MPI on its own does not offer any real concurrency at all, and provides instead a handle-based programming interface to avoid certain cases of deadlocks: the control flow is allowed to continue without finishing the communication, while the communication usually only proceeds when calls into the MPI library are executed.

We developed a C++ layer on top of the asynchronous MPI operations, which follows the design of the C++11 future. Note that the actual `std::future` class cannot be used for this purpose.

```
template<typename T>
class Future{
  void wait();
  bool ready() const;
  bool valid() const;
  T get();
};
```

As different implementations like thread-based `std::future`, task-based `TBB ::future`, and our new `MPIFuture` are available, usability greatly benefits from a dynamically typed interface. This is a reasonable approach, as `std::future` is using a dynamical interface already and also the MPI operations are coarse grained, so that the additional overhead of virtual function calls is negligible. At the same time the user expects a future to offer value semantics, which contradicts the usual pointer semantics used for dynamic polymorphism. In EXA-DUNE we decided to implement type-erasure to offer a clean and still flexible user interface. An `MPIFuture` is responsible for handling all states associated with an MPI operation.

```
class MPIFuture{
private:
  mutable MPI_Request req_;
  mutable MPI_Status status_;
  impl::Buffer<R> data_;
  impl::Buffer<S> send_data_;
public:
  ...
};
```

The future holds a mutable `MPI_Request` and `MPI_Status` to access information on the current operation and it holds buffer objects, which manage the actual data. These buffers offer a great additional value, as we do not access the raw data directly, but can include data transformation and varying ownership. For example it is now possible to directly send an `std::vector<double>`, where the receiver automatically resizes the `std::vector` according to the incoming data stream.

This abstraction layer enables different use cases, highlighted below:

1. **Parallel C++ exception handling**: Exceptions are the recommended way to handle faults in C++ programs. As exceptions alter the execution path of a single node, they are not suitable for parallel programs. As asynchronicity allows for moderately diverging execution paths, we can use it to implement parallel error propagation using exceptions.
2. **Solvers and preconditioners tolerant to hard and soft faults**: This functionality is used for failure propagation, restoration of MPI in case of a hard fault, and asynchronous in-memory checkpointing.
3. **Asynchronous Krylov solvers**: Scalar products in Krylov methods require global communication. Asynchronicity can be used to hide the latency and improve strong scalability.
4. **Asynchronous parallel IO**: The layer allows to transform any non-blocking MPI operation into a really asynchronous operation. This allows also to support asynchronous IO, to hide the latency of write operations and overlap with the computation of the next iteration or time step.
5. **Parallel localized reduced basis methods**: Asynchronicity will be used to mitigate the load-imbalance inherent in the error estimator guided adaptive online enrichment of local reduced bases.

2.2 Parallel C++ Exception Handling

In parallel numerical algorithms, unexpected behaviour can occur quite frequently: a solver could diverge, the input of a component (e.g., the mesher) could be inappropriate for another component (e.g., the discretiser), etc. A well-written code should detect unexpected behaviour and provide users with a possibility to react appropriately in their own programs, instead of simply terminating with some error code. For C++, exceptions are the recommended method to handle this. With well placed exceptions and corresponding try-catch blocks, it is possible to accomplish a more robust program behaviour. However, the current MPI specification [44] does not define any way to propagate exceptions from one rank (process) to another. In the case of unexpected behaviour within the MPI layer itself, MPI programs simply terminate, maybe after a time-out. This is a design decision that unfortunately implies a severe disadvantage in C++, when combined with the ideally asynchronous progress of computation and communication: an exception that is thrown locally by some rank can currently lead to a communication deadlock, or ultimately even to undesired program termination. Even though exceptions are technically an illegal use of the MPI standard (a peer no longer participates in a communication), it undesirably conflicts with the C++ concept of error handling.

Building on top of the asynchronicity layer, we have developed an approach to enable parallel C++ exceptions. We follow C++11 techniques, e.g., use future-like abstractions to handle asynchronous communication. Our currently implemented interface requires ULFM [11], an MPI extension to restore communicators after rank losses, which is scheduled for inclusion into MPI-4. We also provide a fallback solution for non-ULFM MPI installations, that employs an additional communicator for propagation and can, by construction, not handle hard faults, i.e., the loss of a node resulting in the loss of rank(s) in some communicator.

To detect exceptions in the code we have extended the `Dune::MPIGuard`, that previously only implemented the scope guard concept to detect and react on local exceptions. Our extension revokes the MPI communicator using the ULFM functionality if an exception is detected, so that it is now possible to use communication inside a block with scope guard. This makes it superfluous to call the `finalize` and `reactivate` methods of the `MPIGuard` before and after each communication.

```
try{
  MPIGuard guard(comm);
  do_something();
  communicate(comm);
}catch(...){
  comm.shrink();
  recover(comm);
}
```

Listing 1 MPIGuard

Listing 1 shows an example how to use the `MPIGuard` and recover the communicator in a node loss scenario. In this example, an exception that is thrown only on a few ranks in `do_something()` will not lead to a deadlock, since the `MPIGuard` would revoke the communicator. Details of the implementation and further descriptions are available in a previous publication [18]. We provide the "black-channel" fallback implementation as a standalone version.[4] This library uses the P-interface of the MPI standard, which makes it possible to redefine MPI functions. At the initialization of the MPI setting the library creates an opaque communicator, called blackchannel, on which a pending `MPI_Irecv` request is waiting. Once a communicator is revoked, the revoking rank sends messages to the pending blackchannel request. To avoid deadlocks, we use `MPI_Waitany` to wait for a request, which listens also for the blackchannel request. All blocking communication is redirected to non-blocking calls using the P-interface. The library is linked via `LD_PRELOAD` which makes it usable without recompilation and could be removed easily once a proper ULFM implementation is available in MPI.

Figure 1 shows a benchmark comparing the time which is used for duplicating a communicator, revoking it and restore a valid state. The benchmark was performed on PALMA2, the HPC cluster of the University of Muenster. Three implementations are compared; *OpenMPI_BC* and *IntelMPI_BC* are using the blackchannel library based on OpenMPI and IntelMPI, respectively. *OpenMPI_ULFM* uses the ULFM

[4]https://gitlab.dune-project.org/exadune/blackchannel-ulfm.

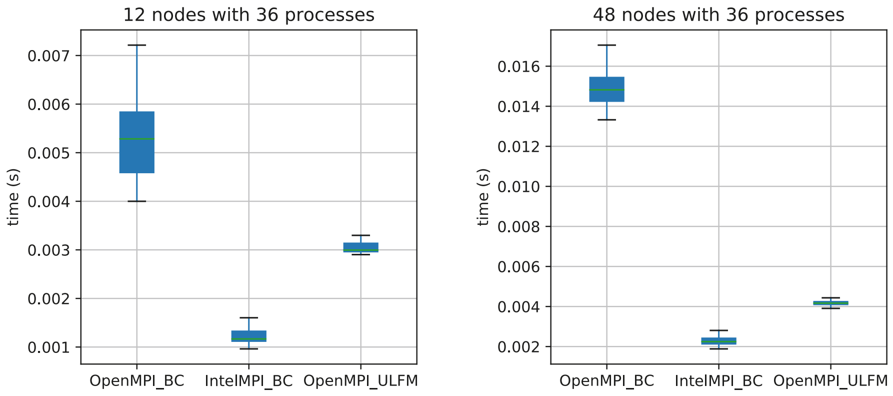


Fig. 1 Benchmark of different MPI implementations: 12 nodes with 36 processes (left), 48 nodes with 36 processes (right), cf. [18]

implementation provided by fault-tolerance.org, which is based on OpenMPI. We performed 100 measurements for each implementation. The blackchannel implementation is competitive to the ULFM implementation. As OpenMPI is in this configuration not optimized and does not use the RDMA capabilities of the interconnect, it is slower than the IntelMPI implementation. The speed up of the *OpenMPI_ULFM* version compared to the *OpenMPI_BC* version is due to the better communication strategy.

2.3 Compressed in-Memory Checkpointing for Linear Solvers

The previously described parallel exception propagation, rank loss detection and communicator restoration by using the ULFM extension, allow us to implement a flexible in-memory checkpointing technique which has the potential to recover from hard faults on-the-fly without any user interaction. Our implementation establishes a backup and recovery strategy which in part is based on a local-failure local-recovery (LFLR) [54] approach, and involves lossy compression techniques to reduce the memory footprint as well as bandwidth pressure. The contents of this subsection have not been published previously.

Modified Solver Interface To enable the use of exception propagation as illustrated in the previous section and to implement different backup recovery approaches we kept all necessary modifications to DUNE-ISTL, the linear solver library. We embed the solver initialisation and the iterative loop in a try-catch block, and provide additional entry and execution points for recovery and backup, see Listing 2 for details. Default settings are provided on the user level, i.e., DUNE-PDELAB.

```
 1  init_variables();
 2  done = false;
 3  while (!done) try {
 4    MPIGuard guard(comm);
 5    if (this->processRecovery(...))
 6      reinit_execution();
 7    } else {
 8      init_execution();
 9    }
10    for (i=0 ; i<=maxit; i++ ) {
11      do_iteration();
12      if (converged) {
13        done = true;
14        break;
15      }
16      this->processBackup(...);
17    }
18  } catch(Exception & e) {
19    done = false;
20    comm.reconstitute();
21    if (!this->processOnException(...))
22      throw;
23  }
```

Listing 2 Solver modifications

This implementation ensures that the iterative solving process is active until the convergence criterion is reached. An exception inside the try-block on any rank is detected by the `MPIGuard` and propagated to all other ranks, so that all ranks will jump to the catch-block.

This catch-block can be specialised for different kind of exceptions, e.g., if a solver has diverged and a corresponding exception is thrown it could define some specific routine to define a modified restart with a possibly more robust setting and/or initial guess. The catch-block in Listing 2 exemplarily shows a possible solution in the scenario of a communicator failure, e.g., a node loss which is detected by using the ULFM extension to MPI, encapsulated by our wrapper for MPI exceptions. Following the detection and propagation, all still valid ranks end up in the catch-block and the communicator must be re-established in some way (Listing 2, line 20). This can be done by shrinking the communicator or replacing lost nodes by some previously allocated spare ones. After the communicator reconstitution a user-provided stack of functions can be executed (Listing 2, line 21) to react on the exception. If there is no on-exception-function or neither of them returns true the exception is re-thrown to the next higher level, e.g., from the linear solver to the application level, or in case of nested solvers, e.g. in optimisation or uncertainty quantification.

Furthermore, there are two additional entry points for user provided function stacks: In line 5 of Listing 2 a stack of recovery functions is executed and if it returns true, the solver expects that some modification, i.e., recovery, has been done.

In this case it could be necessary that the other participating ranks have to update some data, like resetting their local right hand side to the initial values. The backup function stack in line 16 allows the user to provide functions for backup creation etc., after an iteration finished successfully.

Recovery Approaches First, regardless of these solver modifications, we describe the recovery concepts which are implemented into an exemplary recovery interface class providing functions that can be passed to the entry points within the modified solver. The interoperability of these components and the available backup techniques are described later. Our recovery class supports three different methods to recover from a data loss. The first approach is a global rollback to a backup, potentially involving lossy compression: progress on non-faulty ranks may be lost but the restored data originate from the same state, i.e., iteration. This means there is no asynchronous progression in the recovered iterative process but possibly just an error introduced through the used backup technique, e.g., through lossy compression. This compression error can reduce the quality of the recovery and lead to additional iterations of the solver, but is still superior to a restart, as seen later. For the second and third approaches, we follow the local-failure local-recovery strategy and re-initialize the data which are lost on the faulty rank by using a backup. The second, slightly simpler strategy uses these data to continue with solver iterations. The third method additionally smoothes out the probably deteriorated (because of compression) data by solving a local auxiliary problem [29, 31]. This problem is set up by restricting the global operator to its purely local degrees of freedom with indices $\mathcal{F} \subset \mathbb{N}$ and a Dirichlet boundary layer. The boundary layer can be obtained by extending $\mathcal{F}$ to some set $\mathcal{J}$ using the ghost layer, or possibly the connectivity pattern of the operator $\mathbf{A}$. The Dirichlet values on the boundary layer are set to their corresponding values x_N on the neighbouring ranks and thus additional communication is necessary:

$$\begin{aligned} \mathbf{A}(\mathcal{F},\mathcal{F})\tilde{x}(\mathcal{F}) &= b(\mathcal{F}) && \text{in } \mathcal{F} \\ \tilde{x} &= x_N && \text{on } \mathcal{J}\backslash\mathcal{F} \end{aligned}$$

If this problem is solved iteratively and backup data are available, the computation speed can be improved by initializing $\tilde{x}$ with the data from the backup.

Backup Techniques Our current implementation provides two different techniques for compressed backups as well as a basic class which allows ‘zero’-recovery (zeroeing of lost data) if the user wants to use the auxiliary solver in case of data loss without storing any additional data during the iterative procedure.

The next backup class uses a multigrid hierarchy for lossy data compression. Thus it should only be used if a multigrid operator is already in use within the solving process because otherwise the hierarchy has to be built beforehand and introduces additional overhead. Compressing the iterative vector with the multigrid hierarchy currently involves a global communication. In addition there is no adaptive control of the compression depth (i.e., hierarchy level where the backup is stored), but it has to be specified by the user, see a previous publication for details [29].

We also implemented a compressed backup technique based on SZ compression [41]. SZ allows compression to a specified accuracy target and can yield better compression rates than multigrid compression. The compression itself is purely local and does not involve any additional communication. We provide an SZ backup with a fixed user-specified compression target as well as a fully adaptive one which couples the compression target to the residual norm within the iterative solver. For the first we achieve an increased rate while we approach the approximate solution, as seen in Fig. 2 (top, pink lines), at the price of an increased overhead in case of a data loss (cf. Fig. 3). The backup with adaptive compression target (blue lines) gives more constant compression rates, and a better recovery in case of faults in particular in the second half of the iterative procedure of the solver.

The increased compression rate for the fixed SZ backup is obtained because, during the iterative process, the solution gets more smooth and thus can be compressed better by the algorithm. For the adaptive method this gain is counteracted by the demand of a higher compression accuracy.

All backup techniques require to communicate a data volume smaller than the volume of four full checkpoints, see Fig. 2 (bottom). Furthermore this bandwidth requirement is distributed over all 68 iterations (in the fault-free scenario) and could be decreased further by a lower checkpoint frequency.

The chosen backup technique is initiated before the recovery class and passed to it. Further backup techniques can be implemented by using the provided base class and overloading the virtual functions.

Bringing the Approaches Together The recovery class provides three functions which are added to the function stacks within the modified solver interface. The

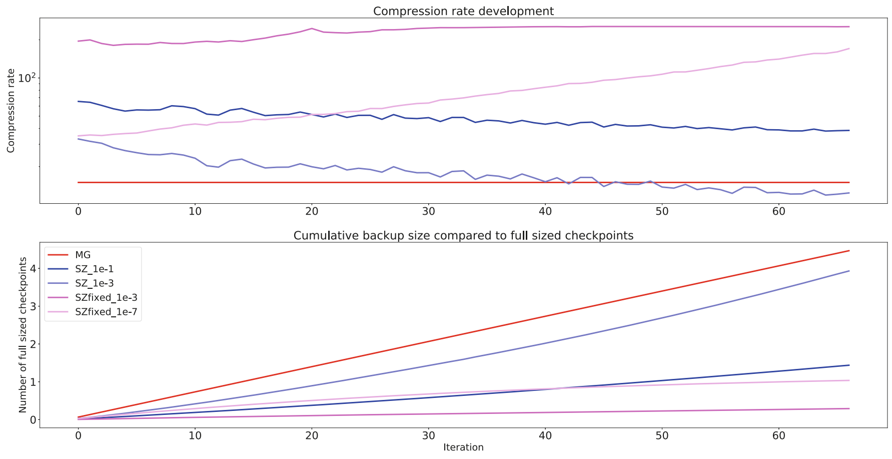


Fig. 2 Compression rate in the iterative solution for an anisotropic Poisson problem on 52 cores with approximately 480 K DOF per core

backup routine is added to the stack of backup functions of the specified iterative solver and generates backups of the current iterative solution by using the provided backup class.

```
     SomeSolver solver;
     SomeBackup backup;
     Recovery recovery(backup);
     solver.addBackupFunction(&Recovery::backup, &recovery);
```

To adapt numerical as well as communication overhead for different fault scenarios and machine characteristics, the backup creation frequency can be varied. After the creation of the backup it is sent to a remote rank where it is kept in memory but never written to disk. In the following this is called ‘remote backup’. Currently the backup propagation happens circular by rank. It is also possible to trigger writing a backup to disk.

In the near future we will implement an on-the-fly recovery if an exception is thrown. These will be provided to the other two function stacks and will differ depending on the availability of the ULFM extensions: if the extension is not available we can only detect and propagate exceptions but not recover a communicator in case of hard faults, i.e., node losses (cf. Sect. 2.2). In this scenario the function provided to the on-exception stack will only write out the global state. Fault-free nodes will write the data of the current iterative vector, whereas for faulty nodes the corresponding remote backup is written. In the following the user will be able to provide a flag to the executable which modifies the backup object initiation to read in the stored checkpoint data. Afterwards the recovery function of our interface will overwrite the initial values of the solver with the checkpointed and possibly smoothed data like described above. If the ULFM extensions are available, the recovery can be realised without any user interaction: during the backup class initiation a global communication ensures that it is the first and therefore fault-free start of the parallel execution. If the process is a respawned one which replaces a lost rank, this communication is matched by a send communication created from the rank which holds the corresponding remote backup. This communication will be initiated by the on-exception function. In addition to this message the remote backup rank sends the stored compressed backup so that the respawned rank can use this backup to recover the lost data.

So far, we have not fully implemented rebuilding the solver and preconditioner hierarchy, and the re-assembly of the local systems, in case of a node loss. This can be done with, e.g., message logging [13], or similar techniques which allow recomputing the individual data on the respawned rank without additional communication.

Figure 3 shows the effect of various combinations of different backup and recovery techniques in case of a data loss on one rank after iteration 60. The problem is an anisotropic Poisson problem with zero Dirichlet boundary conditions which reaches the convergence criterion after 68 iterations in a fault-free scenario (black

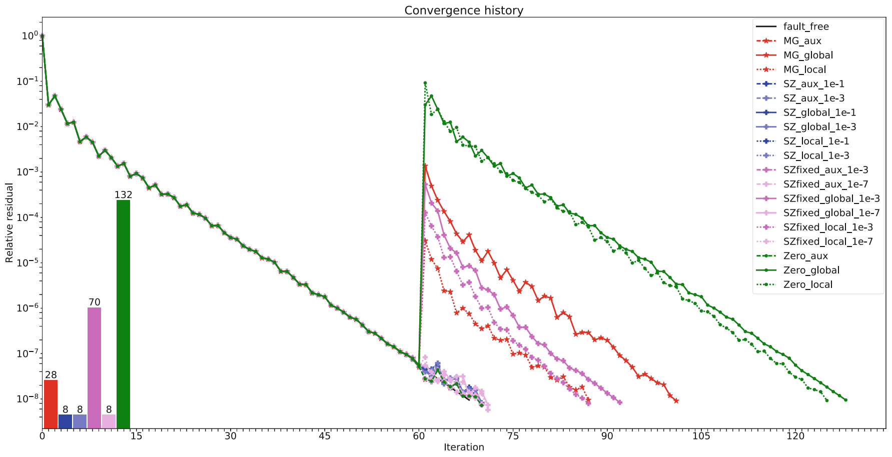


Fig. 3 Convergence history in case of data loss and recovery on one rank, same setting as in Fig. 2. Bottom left: number of iterations to solve the auxiliary problem when using the backups as initial guess. Note that the groups of the same colour are important, not the individual graphs

line). It is executed in parallel on 52 ranks with approximately 480,000 degrees of freedom per rank. Thus one rank loss corresponds to a loss of around 2% of data. For solving a conjugate gradient solver with an algebraic multigrid preconditioner is applied. In addition to the residual norm we show the number of iterations which are needed to solve the auxiliary problem when using different backups as initial guess at the bottom left.

The different backup techniques are colour-coded (multigrid: red; adaptive SZ compression: blue; fixed SZ compression: pink; no backup: green). For the SZ techniques we consider two cases, each with a different compression accuracy (fixed compression), respectively a different additional scaling coefficient (SZ). Recovery techniques are coded with different line styles: global roll-back recovery is indicated by straight lines; simple local recovery is shown with dotted lines and if an auxiliary problem is solved to improve the quality of the recovery it is drawn with a dashed line style. We observe that a zero recovery, multigrid compression and a fixed SZ backup with a low accuracy target are not competitive if no auxiliary problem is solved. The number of iterations needed until convergence then increases significantly. By applying an auxiliary solver the convergence can be almost fully restored (one additional global iteration) but the auxiliary solver needs a high amount of iterations (multigrid: 28; sz: 70; no backup: 132). Other backup techniques only need 8 auxiliary solver iterations. When using adaptive or very accurate fixed SZ compression the convergence behaviour can be nearly preserved even when only a local recovery or a global roll-back is applied. The adaptive compression technique has similar data overhead as the fixed SZ compression (cf. Fig. 2, bottom) but gives slightly better results: both adaptive SZ compression approaches introduce only one additional iteration for all recovery approaches. For the accurate fixed SZ compression (SZfixed_*_1e-7) we have two additional iterations when using local or global recovery but if we apply the auxiliary solver we also have only one additional iteration until convergence.

2.4 Communication Aware Krylov Solvers

In Krylov methods multiple scalar products per iteration must be computed. This involves global sums in a parallel setting. As a first improvement we merged the evaluation of the convergence criterion to the computation of a scalar product. Obviously this does not effect the computed values, but the iteration terminates one iteration later. However this reduces the number of global reductions per iteration from 3 to 2 and thus already saves communication overhead.

As a second step we modify the algorithm, such that only one global communication is performed per iteration. This algorithm can also be found in the paper of Chronopoulos and Gear [15]. Another optimization is to overlap the two scalar products with the application of the operator and preconditioner, respectively. This algorithm was first proposed by Gropp [27]. A fully elaborate version was then presented by Ghysels and Vanroose [27]. This version only needs one global reduction per iteration, which is overlapped with both the application of the preconditioner and operator. This algorithm is shown in Algorithm 2.

Algorithm 1 PCG

```
$r_0 = b - Ax_0$
$p_1 = Mr_0$

$\rho_1 = \langle p_1, r_0 \rangle$
**for** $i = 1, \dots$ **do**
    $q_i = Ap_i$
    $\alpha_i = \langle p_i, q_i \rangle$
    $x_i = x_{i-1} + \frac{\rho_i}{\alpha_i} p_i$
    $r_i = r_{i-1} - \frac{\rho_i}{\alpha_i} q_i$
    $z_{i+1} = Mr_i$
    break if $\|r_i\| < \varepsilon$
    $\rho_{i+1} = \langle z_{i+1}, r_i \rangle$
    $p_{i+1} = \frac{\rho_{i+1}}{\rho_i} p_i + z_{i+1}$
```

Algorithm 2 Pipelined CG

```
$r_0 = b - Ax_0$
$p_1 = Mr_0$
$q_1 = Ap_1$
$\rho_1 = \langle p_1, r_0 \rangle$
$\alpha_1 = \langle p_1, q_1 \rangle$
$s_1 = Mq_1$
$t_1 = As_1$
**for** $i = 1, \dots$ **do**
    $x_i = x_{i-1} + \frac{\rho_i}{\alpha_i} p_i$
    $r_i = r_{i-1} - \frac{\rho_i}{\alpha_i} q_i$
    break if $\|r_i\| < \varepsilon$
    $z_{i+1} = z_i - \frac{\rho_i}{\alpha_i} s_i$
    $w_{i+1} = w_i - \frac{\rho_i}{\alpha_i} t_i$
    $\rho_{i+1} = \langle z_{i+1}, r_i \rangle$
    $\tilde{\alpha}_{i+1} = \langle z_{i+1}, w_{i+1} \rangle$
    $\alpha_{i+1} = \frac{\alpha_i \rho_{i+1}^2}{\rho_i^2} + \tilde{\alpha}_{i+1}$
    $v_{i+1} = M w_{i+1}$
    $u_{i+1} = A v_{i+1}$
    $s_{i+1} = \frac{\rho_{i+1}}{\rho_i} s_i + v_{i+1}$
    $t_{i+1} = \frac{\rho_{i+1}}{\rho_i} t_i + u_{i+1}$
    $p_{i+1} = \frac{\rho_{i+1}}{\rho_i} p_i + z_{i+1}$
    $q_{i+1} = \frac{\rho_{i+1}}{\rho_i} q_i + w_{i+1}$
```

With the new communication interface, described above, we are able to compute multiple sums in one reduction pattern and overlap the communication with computation. To apply these improvements in Krylov solvers the algorithm must be adapted, such that the communication is independent of the overlapping computation. For this adaption we extend the `ScalarProduct` interface by a function which can be passed multiple pairs of vectors for which the scalar product should be computed. The function returns a `Future` which contains a `std::vector<field_type>`, once it has finished.

```
Future<vector<field_type>>
dots(initializer_list<tuple<X&, X&>> pairs);
```

The function can be used in the Krylov methods like this:

```
scalarproduct_future = sp.dot_norm({{p,q}, {z, b}, {b,b}});
// compute while communicate
auto result = scalarproduct_future.get();
field_type p_dot_q = result[0];
field_type z_dot_b = result[1];
field_type norm_b = std::sqrt(result[2]);
```

The runtime improvement of the algorithm strongly depends on the problem size and on the hardware. On large systems the communication overhead makes up a

Table 1 Memory requirement, computational effort and global reductions per iteration for different versions of the preconditioned conjugate gradients method

	Required memory	Additional computational effort	Global reductions
PCG	$4N$	–	2
Chronopoulos and Gear	$6N$	$1N$	1
Gropp	$6N$	$2N$	2 overlapped
Ghysels and Vanroose	$10N$	$5N$	1 overlapped

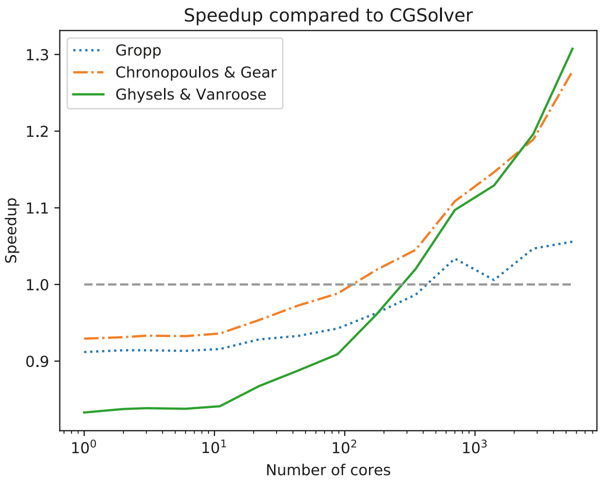


Fig. 4 Strong scaling for (pipelined) Krylov subspace methods

large part of the runtime. However, the maximum speedup is 3 for reducing the number of global reductions and 2 for overlapping communication and computation, compared to the standard version, so that a maximum speedup of 6 is possible. The optimization also increases the memory requirements and vector operations per iteration. An overview of runtime and memory requirements of the methods can be found in Table 1.

Figure 4 shows strong scaling for different methods. The shown speedup is per iteration and with respect to the `Dune::CGSolver`, which is the current CG implementation in DUNE. We use an SSOR preconditoner in an additive overlapping Schwarz setup. The problem matrix is generated from a 5-star Finite Difference model problem. With less cores the current implementation is faster than our optimized one. But with higher core count our optimized version outperforms it. The test was executed on the helics3 cluster of the University on Heidelberg, with 5600 cores on 350 nodes. We expect that on larger systems the speedup will further increase, since the communication is more expensive. The overlap of communication and computation does not really come into play, since the currently used MPI version does not support it completely.

3 Hardware-Aware, Robust and Scalable Linear Solvers

In this section we highlight improved concepts for high-performance iterative solvers. We provide matrix-based robust solvers on GPUs using sparse approximate inverses and optimize algorithm parameters using machine learning. On CPUs we significantly improve the node-level performance by using optimal matrix-free operators for Discontinous Galerkin methods, specialized partially matrix-free preconditioners as well as vectorized linear solvers.

3.1 Strong Smoothers on the GPU: Fast Approximate Inverses with Conventional and Machine Learning Approaches

In continuation of the first project phase, we enhanced the assembly of sparse approximate inverses (SPAI), a kind of preconditioner that we had shown to be very effective within the DUNE solver before [9, 26]. Concerning the assembly of such matrices we have investigated three strategies regarding their numerical efficacy (that is their quality in approximating A^{-1}), the computational complexity of the actual assembly and ultimately, the total efficiency of the amortised assembly combined with all applications during a system solution. For both strategies, this includes a decisive performance engineering for different hardware architectures with focus on the exploitation of GPUs.

SPAI-1 As a starting point we have developed, implemented and tuned a fast SPAI-1 assembly routine based on MKL/LAPACK routines (CPU) and on the cuBlas/cuSparse libraries, performing up to four times faster on the GPU. This implementation is based on the batched solution of QR decompositions that arise in Householder transformations during the SPAI minimisation process. In many cases, we observe that the resulting preconditioner features a high quality comparable to Gauss–Seidel methods. Most importantly, this result still holds true when taking into account the total time-to-solution, which includes the assembly time of the SPAI, even on a single core where the advantages of SPAI preconditioning over forward/backward substitution during the iterative solution process are not yet exploited. More systematic experiments with respect to these statements as well as their extension to larger test architectures are currently being conducted.

SAINV This preconditioner creates an approximation of the factorised inverse $A^{-1} = ZDR$ of a matrix $A \in \mathbb{R}^{N\times N}$ with D being a diagonal, Z an upper triangular and R a lower triangular Matrix.

To describe our new GPU implementation, we write the row-wise updates in the right-looking, outer product form of the A-biconjugation-process of the SAINV factorisation as follows: The assembly of the preconditioner is based on a loop over the existing rows $i \in \{1, \dots, N\}$ of Z (initialised as unit matrix I_N), where in every iteration the loop generally calls three operations, namely a sparse-matrix vector

Algorithm 3 Algorithm of the row-wise updates

```
for ($j = i + 1, \dots, N$) do
    if $D_{jj} \neq 0$ then                         ▷ check if the fraction is unequal to zero
        $\alpha \leftarrow -\frac{D_{jj}}{D_{ii}} z_i^{(i-1)}$
        for $n = 1, \dots$ ,nnz($\alpha$) do
            if $\alpha_n > \varepsilon * max_{i,j}(A_{ij})$ then     ▷ here $\alpha_n$ is the n-th entry of the vector $\alpha$
                if check($z_i^n$,$z_j^n$) then      ▷ Has $z_j$ already an entry at the columnindex of the n-th entry of $\alpha$ ?
                    add($z_j^n$,$\alpha_n$)
                    update_minimum($z_j$)           ▷ get new min. value of j-th row
                else if nnz($z_j$) < $\omega \times \frac{nnz(A)}{dim(A)}$ then     ▷ maximum number of rowentries already reached?
                    insert($z_j$,$\alpha_n$)        ▷ insert the value $\alpha_n$ at the fitting position
                    update_minimum($z_j$)
                else if $\alpha_n > min(z_j)$ then      ▷ check if the value of $\alpha_n$ is bigger than the minimum of $z_j$
                    replace($min(z_j)$,$\alpha_n$)      ▷ replace the old minimum with the value of $\alpha_n$
                    update_minimum($z_j$)
```

multiplication, a dot product and an update of the remaining rows $i + 1, \dots, N$ based on a drop-parameter ε.

In our implementation we use the ELLPACK and CSR formats, pre-allocating a fixed amount of nonzeros of the matrix Z using ω times the average number of nonzeros per row of A. Having a fixed row size, no reallocation of the arrays of the matrix format is needed and the row-wise update can be computed in parallel. This idea is based on the observation that while the density ω for typical drop tolerances is not strictly limited, it generally falls into the interval $]0, 3[$. As the SpMV and the dot kernels are well established, we take a closer look at the row-wise update, which is described more detailed in Algorithm 3. We first compute the values to be added and store them in a variable α. Then we iterate over all nonzero entries of α (which of course has the same sparsity pattern as z_i) and check if the computed value exceeds a certain drop-tolerance. If this condition is met, we have three conditions for an insertion into the matrix Z:

1. Check if there is already an existing nonzero value in the j-th row at the column index of the value α_n and search for the new minimal entry of this row.
2. Else check if there is still place in the j-th row, so we can simply insert the value α_n into that row and search for the new minimal entry of this row.
3. Else check if the value α_n is greater than the current minimum. If this condition is satisfied, then switch the old minimal value with α_n and search for the new minimal entry of this row.

If none of these conditions is met, we drop the computed value without updating the current column and repeat these steps for the next values unequal to zero of the current row. This cap of values for each row also has the following disadvantages: by having a too small maximum of nonzeros per row, a qualitative A-orthogonalization cannot be performed. To avoid this case we only take values of ω greater than one, which seems to be sufficient. Also, if a row has already reached the maximum number of nonzeros, additional but relatively small values may be dropped. This

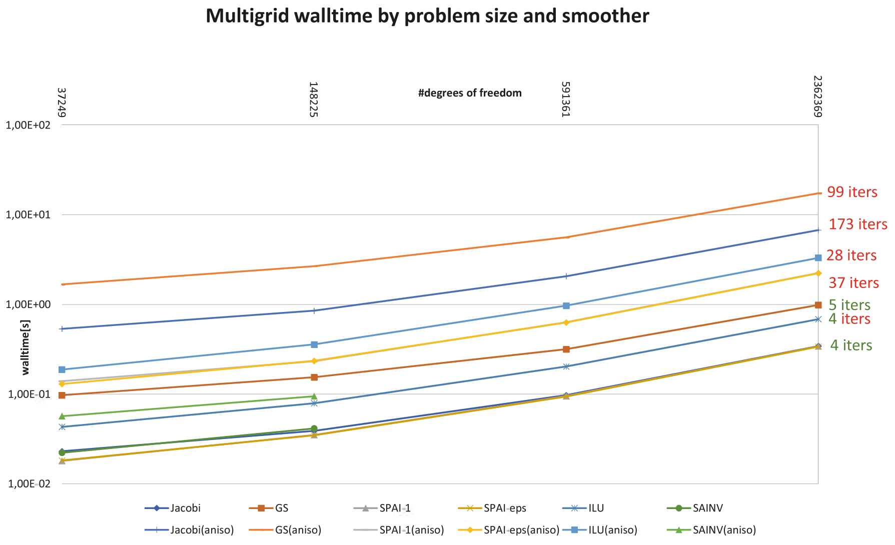


Fig. 5 GPU smoother comparison, isotropic and anisotropic Poisson benchmarks

can become an issue if the sum of these small numbers leads to a relevant entry in a later iteration. For a comparison, Fig. 5 depicts the time-to-solution for V-cycle multigrid using different strong smoothers on a P100 GPU. All smoothers are constructed using 8 Richardson iterations with (reasonably damped if necessary) preconditioners such as Jacobi, Gauss–Seidel, ILU-0, SPAI-1, SPAI-ϵ and SAINV. We set up the benchmark case from a 2D Poisson problem in the isotropic case and with two-sided anisotropies in the grid to harden the problem even for well-ordered ILU approaches. The SPAI approaches are the best choice for the smoother on the GPU.

Machine Learning Finally we started investigating how to construct approximate inverses using methods from Machine Learning [53]. The basic idea here is to treat A^{-1} as a discrete function in the course of a function regression process. The neural network therefore learns how to deduct (extrapolate) an approximation of the inverse. Once trained with many data pairs of matrices and their inverse (a sparse representation of it) a neural network like a multilayer perceptron can be able to approximate inverses rapidly. As a starting point we have employed the finite element method for the Poisson equation on different domains with linear basis functions and have used it to generate expedient systems of equations to solve. Problems of this kind are usually based on sparse M-matrices with characteristics that can be used to reduce the calculation time and effort of the neural network training and evaluation. Our results show that given the pre-defined quality of the preconditioner (equivalent to the ϵ in a SPAI-ϵ method), we can by far numerically outperform even Gauss–Seidel. Using Tensorflow [1] and numpy [4], the learning algorithm can even be performed on the GPU. Here we have used a three-layered fully-connected perceptron with fifty neurons in each layer plus input and output layers, and employed the resulting preconditioners in a Richardson method to solve the mentioned problem on a three times refined L-domain with a fixed number of degrees of freedom. The numerical effort of each evaluation of the neural network is basically the effort of a matrix-vector-multiplication for each layer in which the matrix size depends on the number of neurons per layer (M) and the non zero entries (N) of the input matrix, like $O(NM)$ for the first layer. The inner layers' effort, without input and output layer, just depends on the number of neurons. The crucial task now is to balance the quality of the resulting approximation and the effort to evaluate the network. We use fully connected feed-forward multilayer perceptrons as a starting point. Fully connected means that every neuron in the network is connected to each neuron of the next layer. Moreover there are no backward connections between the different layers (feed-forward). The evaluation of such neural networks is a sequence of chained matrix-vector products.

The entries of the system matrix are represented vector-wise in the input layer (cf. Fig. 6). In the same way, our output layer contains the entries of the approximate inverse. Between these layers we can add a number of hidden layers consisting of hidden neurons. How many hidden neurons we need to create strong approximate inverses is a key design decision and we discuss this below. In general our supervised training algorithm is a backward propagation with random initialisation. Alongside a linear propagation function $i_{\text{total}} = \mathbf{W} \cdot o_{\text{total}} + b$ with the total (layer) net input i_{total}, the weight matrix $\mathbf{W}$, the vector for the bias weights b and the total output of the previous layer o_{total}, we use the rectified linear unit (ReLu) function as activation function $\alpha(x)$ and thus we can calculate the output y of each neuron as $y := \alpha(\sum_{\text{j}} o_{\text{j}} \cdot w_{\text{ij}})$. Here o_{j} is the output of the preceding sending units and w_{ij} are the corresponding weights between the neurons.

For the optimization we use the L2 error function and update the weights with $w_{\text{ij}}^{(t+1)} = w_{\text{ij}}^{(t)} + \gamma \cdot o_i \cdot \delta_{\text{j}}$, with the output o_i of the sending unit and learning rate γ. δ_{j} symbolises the gradient decent method:

$$\delta_{\text{j}} = \begin{cases} f'(i_{\text{j}}) \cdot (\hat{o}_{\text{j}} - o_{\text{j}}) & \text{if neuron j is an output neuron} \\ f'(i_{\text{j}}) \cdot \sum_{k \in S}(\delta_k \cdot w_{kj}) & \text{if neuron j is a hidden neuron.} \end{cases}$$

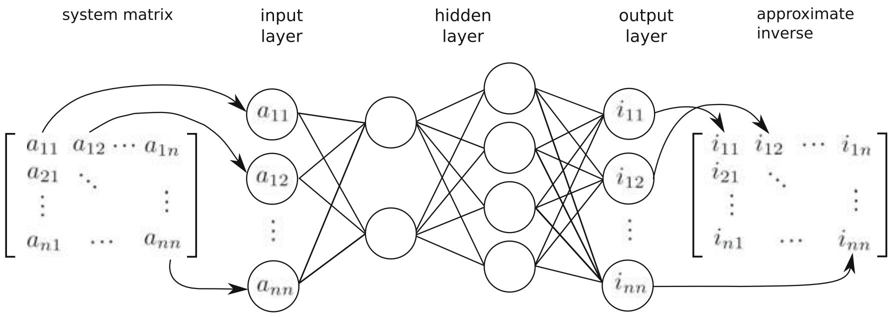


Fig. 6 Model of a neural network for matrix inversion, cf. [53]

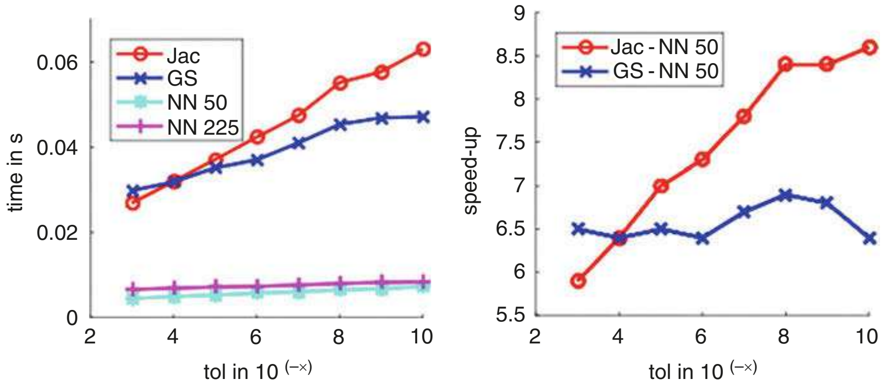


Fig. 7 Results for the defect correction with the neural network, cf. [53]

For details concerning the test/training algorithm we refer to a previous publication [53]. For the defect correction prototype, we find a significant speedup for a moderately anisotropic Poisson problem, see Fig. 7.

3.2 Autotuning with Artificial Neural Networks

Inspired by our usage of Approximate Inverses generated by artificial neural networks (ANNs), we exploit (Feed Forward-) neural networks (FNN) for the automatic tuning of solver parameters. We were able to show that it is possible to use such an approach to provide much better a-priori choices for the parametrisation of iterative linear solvers. In detailed studies for 2D Poisson problems we conducted benchmarks for many test matrices and autotuning systems using FNNs as well as convolutionary neural networks (CNNs) to predict the ω parameter in a SOR solver. In Fig. 8 we depict 100 randomly choosen samples of this study. It can be seen that even for good a-priori choices of ω the NN-driven system can compete whilst 'bad' choices (labeled constant) might lead to a stalling solver.

3.3 Further Development of Sum-Factorized Matrix-Free DG Methods

While we were able to achieve good node-level performance with our matrix-free DG methods in the first funding period, our initial implementations still did not utilize more than about 10% of the theoretical peak FLOP throughput. In the second funding period, we systematically improved on those results by focusing on several aspects of our implementation:

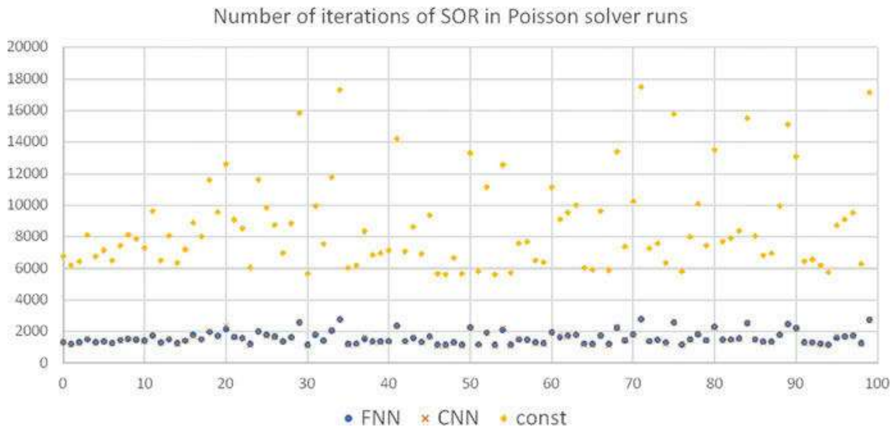


Fig. 8 Result for 100 samples of the FNN-based autotuning system for the ω parameter in SOR

Introduction of Block-Based DOF Processing Our implementation is based on DUNE-PDELAB, a very flexible discretization framework for both continuous and discontinuous discretizations of PDEs. In order to support a wide range of discretizations, PDELab has a powerful system for mapping DOFs to vector and matrix entries. Due to this flexibility, the mapping process is rather expensive. On the other hand, Discontinuous Galerkin values will always be blocked in a cell-wise manner. This can be exploited by only ever mapping the first degree of freedom associated with each cell and then assuming that all subsequent values for this cell are directly adjacent to the first entry. We have added a special 'DG codepath' to DUNE-PDELAB which implements this optimization.

Avoiding Unnecessary Memory Transfers As all of the values for each cell are stored in consecutive locations in memory, we can further optimize the framework behavior by skipping the customary gather/scatter steps before and after the assembly of each cell and facet integral. This is implemented by replacing the data buffer normally passed to the integration kernels with a dummy buffer that stores a pointer to the first entry in the global vector/matrix and directly operates on the global values. This is completely transparent to the integration kernels, as they only ever access the global data through a well-defined interface on these buffer objects. Together with the previous optimization, these two changes have allowed us to reduce the overhead of the framework infrastructure on assembly times from more than 100% to less than 5%.

Explicit Vectorization The DG implementation used in the first phase of the project was written as scalar code and relied on the compiler's auto vectorization support to utilize the SIMD instruction set of the processor, which we tried to facilitate by providing compile time loop bounds and aligned data structures. In the second phase, we have switched to explicit vectorization with a focus on AVX2, which is a common foundation instruction set across all current ×86-based HPC processors. We exploit the possibilities of our C++ code base and use a well-abstracted library which wraps the underlying compiler intrinsic calls [23]. In a separate project [34], we are extending this functionality to other SIMD instruction sets like AVX512.

Loop Reordering and Fusion While vectorization is required to fully utilize modern CPU architectures, it is not sufficient. We also have to feed the execution units with a substantial number of mutually independent chains of computation ($\approx$40–50 on current CPUs). This amount of parallelism can only be extracted from typical DG integration kernels by fusing and reordering computational loops. In contrast to other implementations of matrix-free DG assembly [22, 43], we do not group computations across multiple cells or facets, but instead across quadrature points and multiple input/output variables. In 3D, this works very well for scalar PDEs that contain both the solution itself and its gradient, which adds up to four quantities that exactly fit into an AVX2 register.

Results Table 2 compares the throughput and the hardware efficiency of our matrix-free code for two diffusion-reaction problems A (axis-parallel grid, constant coefficients per cell) and B (affine geometries, variable coefficients per cell) with a matrix-based implementation. Figure 9 compares throughput and floating point performance of our implementation for these problems as well as an additional problem C with multi-linear geometries, demonstrating that we are able to achieve more than 50% of theoretical peak FLOP rate on this machine as well as a good computational processing rate as measured in DOFs/s.

While our work in this project was mostly focused on scalar diffusion-advection-reaction problems, we have also applied the techniques shown here to projection-based Navier–Stokes solvers [51]. One important lesson learned was the unsustainable amount of work required to extend our approach to different problems and/or hardware architectures. This led us to develop a Python-based code generator in a new project [34], which provides powerful abstractions for the building blocks listed above. This toolbox can be extended and combined in new ways to achieve performance comparable to hand-optimized code. Especially for more complex problems involving systems of equations, there are a large number of possible ways to group variables and their derivatives into sum factorization kernels due to our approach of vectorizing over multiple quantities within a single cell. The resulting search space is too large for manual exploration, which the above project solved by the addition of benchmark-driven automatic comparison of those variants. Finally, initial results show good scalability of our code as shown by the strong scaling results in Fig. 10. Our implementation shows good scalability until we reach a local problem size of just 18 cells, where we still need to improve the asynchronicity of ghost data communication and assembly.

Table 2 Full operator application, 2× Intel Xeon E5-2698v3 2.3 GHz (32 cores), for two problems of different complexity, and for comparison matrix assembly for the simpler problem and matrix-based operator application

p	Matrix-free A		Matrix-free B		Matrix-based		Matrix assembly	
	$\frac{\text{DOF}}{\text{s}}$	$\frac{\text{GFLOP}}{\text{s}}$	$\frac{\text{DOF}}{\text{s}}$	$\frac{\text{GFLOP}}{\text{s}}$	$\frac{\text{DOF}}{\text{s}}$	$\frac{\text{GFLOP}}{\text{s}}$	$\frac{\text{DOF}}{\text{s}}$	$\frac{\text{GFLOP}}{\text{s}}$
1	1.70×10^8	104	1.19×10^8	321	2.06×10^8	24.3	1.60×10^7	345
2	3.93×10^8	238	2.52×10^8	450	6.42×10^7	27.3	8.71×10^6	371
3	5.38×10^8	328	3.30×10^8	524	2.69×10^7	29.8	4.66×10^6	368
4	5.95×10^8	387	3.88×10^8	560	9.54×10^6	23.5	2.57×10^6	301
5	6.17×10^8	424	4.03×10^8	568	4.58×10^6	23.3	1.93×10^6	307
6	5.99×10^8	439	4.06×10^8	563	2.31×10^6	23.5	1.21×10^6	231
7	5.70×10^8	442	3.98×10^8	556	1.46×10^6	30.3	6.65×10^5	143
8	5.41×10^8	445	3.85×10^8	541	6.14×10^5	24.1	8.74×10^5	198
9	5.05×10^8	439	3.70×10^8	530	2.98×10^5	26.2	7.01×10^5	133
10	4.71×10^8	432	3.59×10^8	524	–	–	–	–

Note that the matrix-based computations use significantly smaller problem sizes due to memory constraints. p denotes the polynomial degree of the DG space

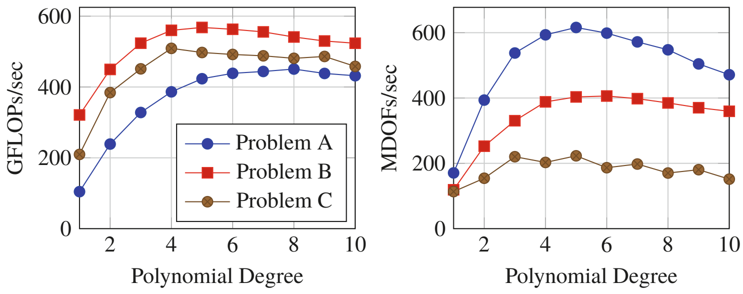


Fig. 9 Floating point performance in GFLOPs/s and throughput in MDOFs/s for full operator application, 2× Intel Xeon E5-2698v3 2.3 GHz for all model problems

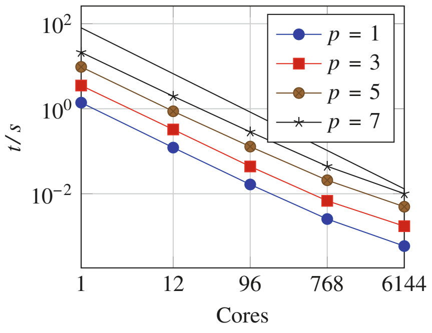


Fig. 10 Runtimes for strong scalability on IWR compute cluster (416 nodes with 2× E5-2630 v3 each, 64 GiB/node, QDR Infiniband)

3.4 Hybrid Solvers for Discontinuous Galerkin Schemes

In Sect. 3.3 we concentrated on the performance of matrix-free *operator application*. This is sufficient for instationary problems with explicit time integration, but in case of stationary problems or implicit time integration, (linear) algebraic systems need to be solved. This requires operator application and robust, scalable preconditioners.

For this we extended hybrid AMG-DG preconditioners [8] in a joint work with Eike Müller from Bath University, UK, [10]. In a solver for matrices arising from higher order DG discretizations the basic idea is to perform all computations on the DG system in a matrix-free fashion and to explicitly assemble only a matrix in a low-order subspace which is significantly smaller. In the sense of subspace correction methods [58] we employ a splitting

$$V_{DG}^{p} = \sum_{T \in \mathcal{T}_h} V_T^p + V_c$$

where V_T^p is the finite element space of polynomial degree p on element T and the coarse space V_c is either the lowest-order conforming finite element space V_h^1 on the mesh $\mathcal{T}_h$, or the space of piecewise constants V_h^0. Note that the symmetric weighted interior penalty DG method from [21] reduces to the cell-centered finite volume method with two-point flux approximation on V_h^0. Note also, that the system on V_c can be assembled without assembling the large DG system.

For solving the blocks related to V_T^p, two approaches have been implemented. In the first (named *partially matrix-free*), these diagonal blocks are factorized using LAPACK and each iteration uses a backsolve. In the second approach the diagonal blocks are solved iteratively to low accuracy using matrix-free sum factorization. Both variants can be used in additive and multiplicative fashion. Figure 11 shows that the partially matrix-free variant is optimal for polynomial degree $p \leq 5$, but starting from $p = 6$, the fully matrix-free version starts to outperform all other options.

In order to demonstrate the robustness of our hybrid AMG-DG method we use the permeability field of the SPE10 benchmark problem [14] within a heterogeneous elliptic problem. This is considered to be a hard test problem in the porous media community. The DG method from [21] is employed. Figure 12 depicts results for different variants and polynomial degrees run in parallel on 20 cores. A moderate increase with the polynomial degree can be observed. With respect to time-to-solution (not reported) the additive (block Jacobi) partially matrix-free variant is to be preferred for polynomial degree larger than one.

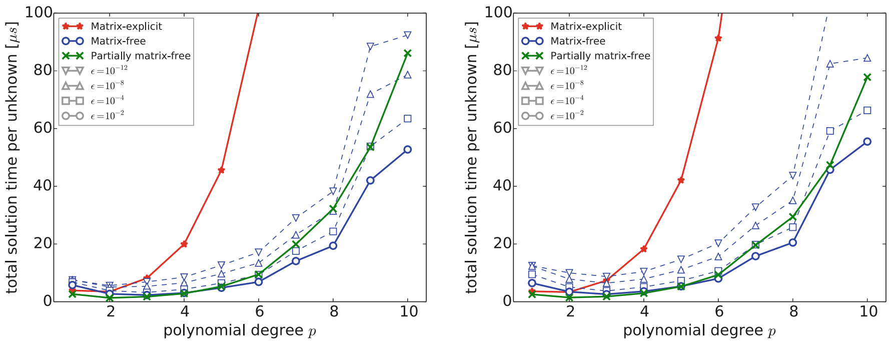


Fig. 11 Total solution time for different implementations and a range of block-solver tolerances ϵ for the Poisson problem (left) and the diffusion problem with spatially varying coefficients (right), cf. [10]

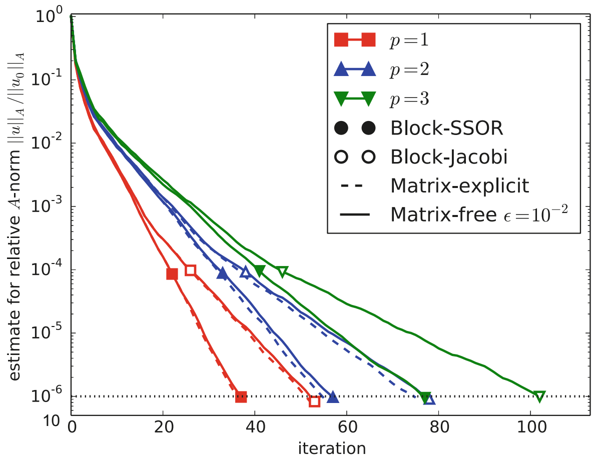


Fig. 12 Convergence history for SPE10 benchmark. The relative energy norm is shown for polynomial degrees 1 (red squares), 2 (blue upward triangles) and 3 (green downward triangles). Results for the block-SSOR smoother are marked by filled symbols and results for the block-Jacobi smoother by empty symbols. cf. [10]

3.5 Horizontal Vectorization of Block Krylov Methods

Methods like Multiscale FEM (see Sect. 4), optimization and inverse problems need to invert the same operator for many right-hand-side vectors. This leads to a block problem, by the following conceptual reformulation:

$$\text{foreach} \quad i \in [0, N] : \text{solve } Ax_i = b_i \qquad \rightarrow \qquad \text{solve } AX = B,$$

with matrices $X = (x_0, \ldots x_N)$, $B = (b_0, \ldots b_N)$. Such problems can be solved using Block Krylov solvers. The benefit is that the approximation space can grow faster, as the solver orthogonalizes the updates for all right-hand-sides. Even for a single right-hand-side Block Krylov based enriched Krylov methods can be used to accelerate the solution process.

Preconditioners and the actual Krylov solver can be sped up using horizontal vectorization. Assuming k right-hand-sides we observe that the scalar product yields a $k \times k$ dense matrix and has $O(k^2)$ complexity. While the mentioned larger approximation space should improve the convergence rate, this is only true for weaker preconditioners, therefore we pursued a different strategy and approximate the scalar product matrix by a sparse matrix, so that we again retain $O(k)$ complexity. In particular we consider the case of a diagonal or block-diagonal matrix. The diagonal matrix basically results in k independent solvers running in parallel, so that the performance gain is solely based on SIMD vectorization and the associated favorable memory layout.

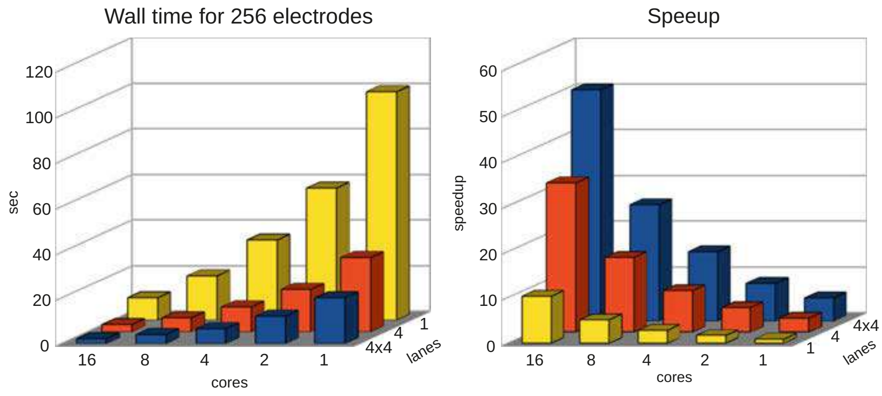


Fig. 13 Horizontal vectorization of a linear solver for 256 right-hand-side vectors. Timings on a Haswell-EP (E5-2698v3, 16 cores, AVX2, 4 lanes). Comparison with 1–16 cores and no SIMD, AVX (4 lanes), AVX (4 $\times$ 4 lanes)

For the implementation in DUNE-ISTL we use platform portable C++ abstractions of SIMD intrinsics, building on the VC library[38] and some DUNE specific extensions. We use this to exchange the underlying data type of the right-hand-side and the solution vector, so that we no longer store scalars, but SIMD vectors. This is possible when using generic programming techniques, like C++ templates, and yields a row-wise storage of the dense matrices X and B. This row-wise storage is optimal and ensures a high arithmetic intensity. The implementations of the Krylov solvers have to be adapted to the SIMD data types, since some operations, like casts and branches, are not available generically for SIMD data types. As a side effect, all preconditioners, including the AMG, are now fully vectorized.

Performance tests using 256 right-hand-side vectors for a 3D Poisson problem show nearly optimal speedup on a 64 core system (see Fig. 13). The tests are carried out on a Haswell-EP (E5-2698v3, 16 cores, AVX2, 4 lanes). We observe a speedup of 50, while the theoretical speedup is 64.

4 Adaptive Multiscale Methods

The main goal in the second funding phase was a distributed adaptive multilevel implementation of the localized reduced basis multi-scale method (LRBMS [49]). Like Multiscale FEM (MsFEM), LRBMS is designed to work on heterogenous multiscale or large scale problems. It performs particularly well for problems that exhibit scale separation with effects on both a fine and a coarse scale contributing to the global behavior. Unlike MsFEM, LRBMS is best applied in multi-query settings in which a parameterized PDE needs to be solved many times for different parameters. As an amalgam of domain decomposition and model order reduction techniques, the computational domain is partitioned into a coarse grid with each macroscopic grid cell representing a subdomain for which, in an offline pre-compute stage, local reduced bases are constructed. Appropriate coupling is then applied to produce a global solution approximation from localized data. For increased approximation fidelity we can integrate localized global solution snapshots into the bases, or the local bases can adaptively be enriched in the online stage, controlled by a localized a-posteriori error estimator.

4.1 Continuous Problem and Discretization

We consider elliptic parametric multi-scale problems on a domain $\Omega \subset \mathbb{R}^d$ where we look for $p(\boldsymbol{\mu}) \in Q$ that satisfy

$$b(p(\boldsymbol{\mu}), q; \boldsymbol{\mu}) = l(q) \qquad \text{for all}\, q \in H^1_0(\Omega), \tag{1}$$

$\boldsymbol{\mu}$ are parameters with $\boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^p$, $p \in \mathbb{N}$. We let $\epsilon > 0$ be the multi-scale parameter associated with the fine scale. For demonstration purposes we consider a particular linear elliptic problem setup in $\Omega \subset \mathbb{R}^d$ $(d = 2, 3)$ that exhibits a multiplicative splitting in the quantities affected by the multi-scale parameter ϵ. It is a model for the so called global pressure $p(\boldsymbol{\mu}) \in H^1_0(\Omega)$ in two phase flow in porous media, where the total scalar mobility $\lambda(\boldsymbol{\mu})$ is parameterized. κ_ϵ denotes the heterogenous permeability tensor and f the external forces. Hence, we seek p that satisfies weakly in $H^1_0(\Omega)$,

$$-\nabla \cdot (\lambda(\boldsymbol{\mu})\kappa_\epsilon \nabla p(\boldsymbol{\mu})) = f \qquad \text{in}\, \Omega. \tag{2}$$

With $A(x; \boldsymbol{\mu}) := \lambda(\boldsymbol{\mu})\kappa_\epsilon(x)$ this gives rise to the following definition of the forms in (1)

$$b(p(\boldsymbol{\mu}), q; \boldsymbol{\mu}) := \int_\Omega A(\boldsymbol{\mu})\nabla p \cdot \nabla q, \quad l(q) := \int_\Omega f q.$$

For the discretization we first require a triangulation $\mathcal{T}_H$ of Ω for the macro level. We call the elements $T \in \mathcal{T}_H$ subdomains. We then require each subdomain be covered with a fine partition $\tau_h(T)$ in a way that $\mathcal{T}_H$ and $\tau_h := \Sigma_{T\in\mathcal{T}_H}\tau_h(T)$ are nested. We denote by $\mathcal{F}_H$ the faces of the coarse triangulation and by $\mathcal{F}_h$ the faces of the fine triangulation.

Let $V(\tau_h) \subset H^2(\tau_h)$ denote any approximate subset of the broken Sobolev space $H^2(\tau_h) := \{q \in L^2(\Omega) \mid q|_t \in H^2(t)\, \forall t \in \tau_h\}$. We call $p_h(\boldsymbol{\mu}) \in V(\tau_h)$ an approximate solution of (1), if

$$b_h\big(p_h(\boldsymbol{\mu}), v; \boldsymbol{\mu}\big) = l_h(v; \boldsymbol{\mu}) \qquad \text{for all}\, v \in V(\tau_h). \tag{3}$$

Here, the DG bilinear form b_h and the right hand side l_h are chosen according to the SWIPDG method [20], i.e.

$$b_h(v, w; \boldsymbol{\mu}) := \sum_{t \in \tau_h} \int_t A(\boldsymbol{\mu}) \nabla v \cdot \nabla w + \sum_{e \in \mathcal{F}(\tau_h)} b_h^e(v, w; \boldsymbol{\mu})$$

$$l_h(v; \boldsymbol{\mu}) := \sum_{t \in \tau_h} \int_t f v,$$

where the DG coupling bilinear forms b_h^e for a face e is given by

$$b_h^e(v, w; \boldsymbol{\mu}) := \int_e \langle A(\boldsymbol{\mu}) \nabla v \cdot \mathbf{n_e} \rangle [w] + \langle A(\boldsymbol{\mu}) \nabla w \cdot \mathbf{n_e} \rangle [v] + \frac{\sigma_e(\boldsymbol{\mu})}{|e|^\beta} [v][w].$$

The LRBMS method allows for a variety of discretizations, i.e. approximation spaces $V(\tau_h)$. As a particular choice of an underlying high dimensional approximation space we choose $V(\tau_h) = Q_h^k := \bigoplus_{T \in \mathcal{T}_H} Q_h^{k,T}$, where the discontinuous local spaces are defined as

$$Q_h^{k,T} := Q_h^{k,T}(\tau_h(T)) := \{q \in L^2(T) \mid q|_t \in \mathbb{P}_k(t) \, \forall t \in \tau_h(T)\}.$$

4.2 Model Reduction

For model order reduction in the LRBMS method we choose the reduced space $Q_{\text{red}} := \bigoplus_{T \in \mathcal{T}_H} Q_{\text{red}}^T \subset Q_h^k$ with local reduced approximation spaces $Q_{\text{red}}^T \subset Q_h^{k,T}$. We denote $p_{\text{red}}(\boldsymbol{\mu})$ to be the reduced solution of (3) in Q_{red}. This formulation naturally leads to solving a sparse blocked linear system similar to a DG approximation with high polynomial degree on the coarse subdomain grid.

The construction of subdomain reduced spaces Q_{red}^T is again very flexible. Initialization with shape functions on T up to order k ensures a minimum fidelity. Basis extensions can be driven by a discrete weak greedy approach which incorporates localized solutions of the global system. Depending on available computational resources, and given a suitable localizable a-posteriori error estimator $\eta(p_{\text{red}}(\boldsymbol{\mu}), \boldsymbol{\mu})$, we can forego computing global high-dimensional solutions altogether and only rely on online enrichment to extend Q_{red}^T 'on the fly'. With online enrichment, given a reduced solution $p_{\text{red}}(\boldsymbol{\mu})$ for some arbitrary $\boldsymbol{\mu} \in \mathcal{P}$, we first compute local error indicators $\eta^T(p_{\text{red}}(\boldsymbol{\mu}), \boldsymbol{\mu})$ for all $T \in \mathcal{T}_H$. If $\eta^T(p_{\text{red}}(\boldsymbol{\mu}), \boldsymbol{\mu})$ is greater than some prescribed bound $\delta_{\text{tol}} > 0$, we solve on a overlay region $\mathcal{N}(T) \supset T$ and extend Q_{red}^T with $p_{\mathcal{N}(T)}(\boldsymbol{\mu})|_T$. Inspired by results in [17] we set the overlay region's diameter $\text{diam}(\mathcal{N}(T))$ of the order $\mathcal{O}(\text{diam}(T)|log(\text{diam}(T))|)$. In practice we use the completely on-line/off-line decomposable error estimator developed in [49, Sec. 4] which in turn is based on the idea of producing a conforming reconstruction of the diffusive flux $\lambda(\boldsymbol{\mu})\kappa_\epsilon \nabla_h p_h(\boldsymbol{\mu})$ in some Raviart-Thomas-Nédélec space $V_h^l(\tau_h) \subset H_{div}(\Omega)$ presented in [21, 56].

This process is then repeated until either a maximum number of enrichment steps occur or $\eta^T(p_{\text{red}}(\boldsymbol{\mu}), \boldsymbol{\mu}) \leq \delta_{\text{tol}}$.

Algorithm 4 Schematic representation of the LRBMS pipeline

```
Require: $P_{\text{train}} \subset \mathcal{P}$
Require: Reconstruction operator $R_h(p_{\text{red}}(\boldsymbol{\mu})) : Q_{\text{red}}(\mathcal{T}_H) \to Q_h^k(\tau_h)$
 1: function GreedyBasisGeneration($\delta_{grdy}$, $\eta(p_{\text{red}}(\boldsymbol{\mu}), \boldsymbol{\mu})$=None )
 2:     if $\eta(p_{\text{red}}(\boldsymbol{\mu}), \boldsymbol{\mu})$ is not None then
 3:         $E \leftarrow \{\eta(p_{\text{red}}(\mu_i), \mu_i) \mid \mu_i \in \mathcal{P}_{\text{train}}\}$
 4:     else
 5:         $E \leftarrow \{||R_h(p_{\text{red}}(\mu_i)) - p_h(\mu_i)|| \mid \mu_i \in \mathcal{P}_{\text{train}}\}$
 6:     while $E \neq \emptyset$ AND $max(E) \geq \delta_{grdy}$ do
 7:         $i \leftarrow \text{argmax}(E)$
 8:         compute $p_h(\boldsymbol{\mu_i})$
 9:         for all $T \in \mathcal{T}_H$ do
10:             extend $Q_{\text{red}}^T$ with $p_h(\boldsymbol{\mu})|_T$
11:         $E \leftarrow E \setminus E_i$

12: Generate $\mathcal{T}_H$                                        ▷ Offline Phase
13: for all $T \in \mathcal{T}_H$ do
14:     create $\tau_h(T)$
15:     init $Q_{\text{red}}^T$ with DG shape functions of order $k$
16: GreedyBasisGeneration($\cdots$)                                ▷ Optional
17: compute $p_{\text{red}}(\boldsymbol{\mu})$ for arbitrary$\boldsymbol{\mu}$                    ▷ Online phase
18: for all $T \in \mathcal{T}_H$ do                         ▷ Optional Adaptive Enrichment
19:     $\eta \leftarrow \eta^T(p_{\text{red}}(\boldsymbol{\mu}), \boldsymbol{\mu})$
20:     while $\eta \geq \delta_{\text{tol}}$ do
21:         compute $p_{\mathcal{N}(T)}(\boldsymbol{\mu})$
22:         $Q_{\text{red}}^T \leftarrow p_{\mathcal{N}(T)}(\boldsymbol{\mu})|_T$
```

4.3 Implementation

We base our MPI-parallel implementation of LRBMS on the serial version developed previously. In this setup the high-dimensional quantities and all grid structures are implemented in Dune. The model order reduction as such is implemented in Python using pyMOR [45]. The model reduction algorithms in pyMOR follow a solver agnostic design principle. Abstract interfaces allow for example projections, greedy algorithms or reduced data reconstruction to be written without knowing details of the PDE solver backend. The global macro grid $\mathcal{T}_H$ can be any MPI-enabled Dune grid manager with adjustable overlap size for the domain decomposition, we currently use Dune-YaspGrid. The fine grids $\tau_h(T)$ are constructed using the same grid manager as on the macro scale, with MPI subcommunicators.These are currently limited to a size of one (rank-local), however the overall scalability could benefit from dynamically sizing these subcommunicators to balance communication overhead and computational intensity as demonstrated in [36, Sec. 2.2]. The assembly of the local (coupling) bilinear forms is done in DUNE-GDT [24], with pyMOR/Python bindings facilitated through DUNE-XT [46], where DUNE-GRID-GLUE [19] generates necessary grid views for the SWIPDG coupling between otherwise unrelated grids. Switching to DUNE-GRID-GLUE constitutes a major step forward in robustness of the overall algorithm, compared to our previous manually implemented approach to matching independent local grids for coupling matrices assembly.

We have identified three major challenges in parallelizing all the steps in LRBMS:

1. **Global solutions $p_h(\mu)$ of the blocked system in Eq. (3) with an appropriate MPI-parallel iterative solver.** With the serial implementation already using DUNE-ISTL as the backend for matrix and vector data, we only had to generate an appropriate communication configuration for the blocked SWIPDG matrix structure to make the BiCGStab solver usable in our context. We tested this setup on the SuperMUC Petascale System in Garching. The results in Fig. 14 show very near ideal speedup from 64 nodes with 1792 MPI ranks up to a full island with 512 nodes and 14336 ranks.

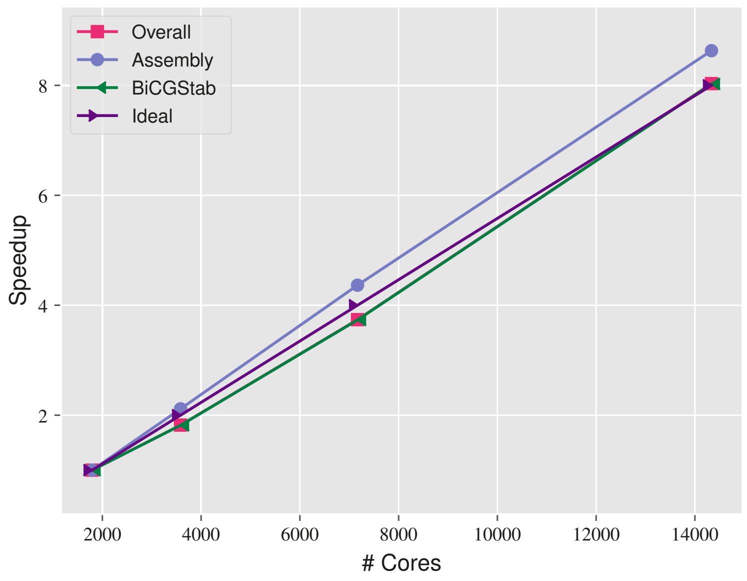


Fig. 14 Localized Reduced Basis Method: Block-SWIPDG speedup results; linear system solve (green), discretization and system assembly (blue), theoretic ideal speedup (violet) and actual achieved speedup for the overall run time (red). Simulation on $\sim 7.9 \cdot 10^6$ cubical cells shows minimum 94% parallel efficiency, scaling from 64 to 512 nodes (SuperMUC Phase 2)

2. **(Global) Reduced systems also need a distributed solver.** By design all reduced quantities in pyMOR are, at the basic, unabstracted level, NumPy arrays [57]. Therefore we cannot easily re-use the DUNE-ISTL based solvers for the high-dimensional systems. Our current implementation gathers these reduced system matrices from all MPI-ranks to rank 0, recombines them, solves the system with a direct solver and scatters the solution. There is great potential in making this step more scalable by either using a distributed sparse direct solver like Mumps [3] or translating the data into the DUNE-ISTL backend.
3. **Adaptive online enrichment is inherently load imbalanced due to its localized error estimator guidance.** The load imbalance results from one rank idling while waiting to receive updates to a basis on a subdomain in its overlap region from another rank. This idle time can be minimized by encapsulating the update in a `MPIFuture` described in Sect. 2.1. This will allow the rank to continue in its own enrichment process until the updated basis is actually needed in a subsequent step.

5 Uncertainty Quantification

The solution of stochastic partial differential equations (SPDEs) is characterized by extremely high dimensions and poses great (computational) challenges. Multilevel Monte Carlo (MLMC) algorithms attract great interest due to their superiority over the standard Monte Carlo approach. Based on Monte Carlo (MC), MLMC retains the properties of independent sampling. To overcome the slow convergence of MC, where many computationally expensive PDEs have to be solved, MLMC combines in a proper way cheap MC estimators and expensive MC estimators, achieving (much) faster convergence. One of the critical components of the MLMC algorithms is the way in which the coarser levels are selected. The exact definition of the levels is an open question and different approaches exist. In the first funding phase, Multiscale FEM was used as a coarser level in MLMC. During the second phase, the developed parallel MLMC algorithms for uncertainty quantification were further enhanced. The main focus was on exploring the capabilities of the renormalization approach for defining the coarser levels in the MLMC algorithm, and on using MLMC as a coarse grained parallelization approach.

Here, we employ MLMC to exemplarily compute the mean flux through saturated porous media with prescribed pressure drop and known distribution of the random coefficients.

Mathematical Problem As a model problem in $\mathbb{R}^2$ or $\mathbb{R}^3$, we consider steady state single phase flow in random porous media:

$$-\nabla \cdot [k(x,\omega)\nabla p(x,\omega)] = 0 \text{ for } x \in D = (0,1)^d, \omega \in \Omega$$

subject to the boundary conditions $p_{x=0} = 1$ and $p_{x=1} = 0$ and zero flux on other boundaries. Here p is pressure, k is scalar permeability, and ω is a random vector. The quantity of interest is the mean (expected value) $E[Q]$ of the total flux Q through the inlet of the unit cube i.e., $Q(x,\omega) := \int_{x=0} k(x,\omega)\partial_n p(x,\omega)dx$. Both the coefficient $k(x,\omega)$ and the solution $p(x,\omega)$ are subject to uncertainty, characterized by the random vector ω in a properly defined random space Ω. For generating permeability fields we consider the practical covariance $C(x,y) = \sigma^2 \exp(-||x-y||_2/\lambda)$. An algorithm based on forward and inverse Fourier transform over the circulant covariance matrix is used to generate the permeability field. For solving the deterministic PDEs a Finite Volume method on a cell centered grid is used [32]. More details and further references can be found in a previous paper [47].

Monte Carlo Simulations To quantify the uncertainty, and compute the mean of the flux we use a MLMC algorithm. Let ω_M be a random vector over a properly defined probability space, and Q_M be the corresponding flux. It is known that $E[Q_M]$ can be made arbitrarily close to $E[Q]$ by choosing M sufficiently large. The standard MC algorithm convergences very slowly, proportionally to the variance over the square root of the number of samples, which makes it often unfeasible. MLMC introduces L levels with the L-th level coinciding with the considered problem, and exploits the telescopic sum identity:

$$E[Q_M^L(\omega)] = E[Q_M^0(\omega)] + E[Q_M^1(\omega) - Q_M^0(\omega)] + \dots E[Q_M^L(\omega) - Q_M^{L-1}(\omega)]$$

The notation $Y^l = Q^1 - Q^{l-1}$ is also used. The main idea of MLMC is to properly define levels, and combine a large number of cheap simulations, that are able to approximate the variance well, with a small number of expensive correction simulations providing the needed accuracy. For details on Monte Carlo and MLMC we refer to previous publications [32, 47] and the references therein. Here, the target is to estimate the mean flux on a fine grid, and we define the levels as discretizations on coarser grids. In order to define the permeability at the coarser levels we use the renormalization approach.

MLMC has previously run the computations at each level with the same tolerance. However, in order to evaluate the number of samples needed per level, one has to know the variance at each level. Because the exact variance is not known in advance, MLMC starts by performing simulations with a prescribed, moderate number of samples per level. The results are used to evaluate the variance at each level, and thus to evaluate the number of samples needed per level. This procedure can be repeated several times in an Estimate–Solve cycle. At each estimation step, information from all levels is needed, which leads to a synchronization point in the parallel realization of the algorithm. This may require dynamic redistribution of the resources after each new evaluation.

MLMC can provide a coarse graining in the parallelization. A well balanced algorithm has to account for several factors: (1) How many processes should be allocated per level; (2) how many processes should be allocated per deterministic problem including permeability generation; (3) how to parallelize the permeability generation; (4) which of the parallelization algorithms for deterministic problems available in EXA-DUNE should be used; (5) should each level be parallelized separately and if not, how to group the levels for efficient parallelization. The last factor is the one giving coarse grain parallelization opportunities. For the generation of the permeability, we use the parallel MPI implementation of the FFTW library. As deterministic solver, we use a parallel implementation of the conjugate gradient scheme preconditioned with AMG, provided by DUNE-ISTL. Both of them have their own internal domain decomposition.

We shortly discuss one Estimate-Solve cycle of the MLMC algorithm. Without loss of generality we assume 3-level MLMC. Suppose that we have already computed the required number of samples per level (i.e., we are after Estimate and before Solve). Let us denote by $N_i, i = \{0, 1, 2\}$ the number of required realizations per level for $\widehat{Y}_l$, by p_i the number of processes allocated per $\widehat{Y}_i$, by $p^g_{l_i}$ the respective group size of processes working on a single realization, by n the number of realizations for each group of levels, with t_i the respective time for solving a single problem once, and finally with p^{total} the total number of available processes. Then we can compute the total CPU time for the current Estimate-Solve cycle as

$$T^{\text{total}}_{\text{CPU}} = N_0 t_0 + N_1 t_1 + N_2 t_2.$$

Ideally each process should take $T^p_{\text{CPU}} = T^{\text{total}}_{\text{CPU}}/p^{\text{total}}$. Dividing the CPU time needed for one $\widehat{Y}_i$ by T^p_{CPU}, we get a continuous value for the number of processes on a given level $p^{\text{ideal}}_i = N_i t_i / T^p_{\text{CPU}}$ for $i = \{0, 1, 2\}$. Then we can take $p_i = \lfloor p^{\text{ideal}}_i \rfloor$. To obtain an integer value for the number of processes allocated per level, first we construct a set of all possible splits of the original problem as a combination of subproblems (e.g., parallelize level 2 separately and the combination of levels 0 and 1, or parallelize all levels simultaneously, etc.). Each element of this set is evaluated independently, and all combinations of upper and lower bounds are calculated, such that p^{ideal}_i is divisible by $p^g_{l_i}$, $\sum_{l=0}^{2} p_i < p^{\text{total}}$ and $p_i \le N_i p^g_{l_i}$. Traversing, computing and summing the computational time needed for each element gives us a time estimation. Then we select the element (grouping of levels) with minimal computational time. To tackle the distribution of the work on a single level, a similar approach can be employed. Due to the large dimension of the search tree a heuristic technique can be employed. Here we consider a simple predefined group size for each deterministic problem, having in mind that when using AMG the work for a single realization at the different levels is proportional to the unknowns at this level.

Numerical Experiments Results for a typical example are shown in Fig. 15. The parameters are $\sigma = 2.75, \lambda = 0.3$. The tests are done on SuperMUC-NG, LRZ Munich on a dual Intel Skylake Xeon Platinum 8174 node. Due to the stochasticity of the problem, we plot the speedup multiplied with the proportion of the tolerance. The renormalization has shown to be a very good approach for defining the coarser levels in MLMC. The proposed parallelization algorithm gives promising scalability results. It is weakly coupled to the number of samples that MLMC estimates.

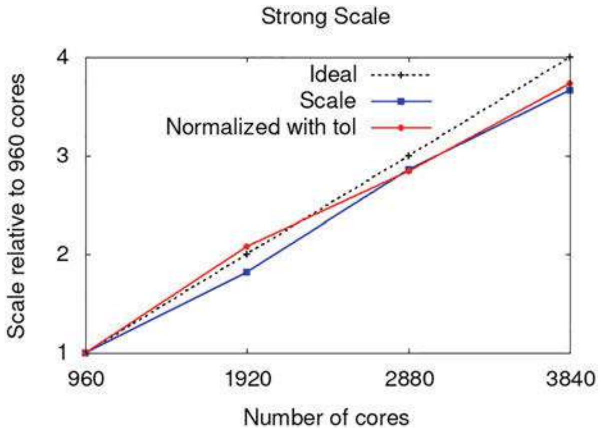


Fig. 15 Scalability of the MLMC approach

Although the search for an optimal solution is an NP-hard problem, the small number of levels enables a full traversing of the search tree. It can be further improved by automatically selecting the number of processes per group that solves a single problem. One also may consider introducing interrupts between the MPI communicators on a level to further improve the performance.

6 Land-Surface Flow Application

To test some of the approaches developed in the EXA-DUNE project, especially the usage of sum-factorized operator evaluation with more complex problems, we developed an application to simulate coupled surface-subsurface flow for larger geographical areas. This is a topic with high relevance for a number of environmental questions from soil protection and groundwater quality up to weather and climate prediction.

One of the central aims of the new approach developed in the project is to be able to relate a physical meaning to the parameter functions used in each grid cell. This is not possible with the traditional structured grid approaches as the necessary resolution would be prohibitive. To avoid the excessive memory requirements of completely unstructured grids we build on previous results for block-structured meshes and use a two-dimensional unstructured grid on the surface which is extruded in a structured way in vertical direction. However, more flexible discretization schemes are needed for such grids, compared to the usual cell-centered Finite-Volume approaches.

6.1 Modelling and Numerical Approach

To describe subsurface flow we use the Richards equation [52]

$$\frac{\partial \theta(\psi)}{\partial t} - \nabla \cdot \left[k(\psi)\left(\nabla \psi + \mathbf{e}_g\right)\right] + q_w = 0$$

where θ is the volumetric water content, ψ the soil water potential, k the hydraulic conductivity, $\mathbf{e}_g$ the unit vector pointing in the direction of gravity and q_w a volumetric source or sink term.

In nearly all existing models for coupled surface-subsurface flow, the kinematic-wave approximation is used for surface flow, which only considers surface slope as driving force and does not even provide a correct approximation of the steady-state solution. The physically more realistic shallow-water-equations are used rarely, as they are computationally expensive. We use the diffusive-wave approximation, which still retains the effects of water height on run-off, yields a realistic steady-state solution and is a realistic approximation for flow on vegetation covered ground [2]:

$$\frac{\partial h}{\partial t} - \nabla \cdot [D(h, \nabla h)\nabla(h+z)] = f_c, \tag{4}$$

where h is the height of water over the surface level z, f_c is a source-sink term (which is used for the coupling) and the diffusion coefficient D is given by

$$D(h, \nabla h) = \frac{h^\alpha}{C \cdot \|\nabla(h+z)\|^{1-\gamma}}$$

with $\|\cdot\|$ refering to the Euclidean norm and α, γ and C are empirical constants. In the following we use $\alpha = \frac{5}{3}$ and $\gamma = \frac{1}{2}$ to obtain Manning's formula and a friction coefficient of $C = 1$.

Both equations are discretised with a cell-centered Finite-Volume scheme and alternatively with a SWIPDG scheme in space (see Sect. 3) and an appropriate diagonally implicit Runge–Kutta scheme in time for the subsurface and an explicit Runge–Kutta scheme for the surface flow. Upwinding is used for the calculation of conductivity in subsurface flow [5] and for the water height in the diffusion term in surface flow.

First tests have shown that the formulation of the diffusive-wave approximation from the literature as given by Eq. (4) does not result in a numerically stable solution if the gradient becomes very small, as then a gradient approaching zero is multiplied by a diffusion coefficient going to infinity. A much better behaviour is achieved by rewriting the equation as

$$\frac{\partial h}{\partial t} - \nabla \cdot \left[\frac{h^\alpha}{C} \cdot \frac{\nabla(h+z)}{\|\nabla(h+z)\|^{1-\gamma}}\right] = f_c,$$

where the rescaled gradient $\frac{\nabla(h+z)}{\|\nabla(h+z)\|^{1-\gamma}}$ is always going to zero when $\nabla(h+z)$ is going to zero as long as $\gamma < 1$ and the new diffusion coefficient h^α/C is bounded.

Due to the very different time-scales for surface and subsurface flow, an operator-splitting approach is used for the coupled system. A new coupling condition has been implemented, which is a kind of Dirichlet-Neumann coupling, but guarantees a mass-conservative solution. With a given height of water on the surface (from the initial condition or the last time step modified by precipitation and evaporation), subsurface flow is calculated with a kind of Signorini boundary condition, where all surface water is infiltrated in one time step as long as the necessary gradient is not larger than the pressure resulting from the water ponding on the surface (in infiltration) and potential evaporation rates are maintained as long as the pressure at the surface is not below a given minimal pressure (during evaporation). The advantage of the new approach is that it does not require a tracking of the sometimes complicated boundary between wet and dry surface elements, that it yields no unphysical results and that the solution is mass-conservative even if not iterated until convergence.

Parallelisation is obtained by an overlapping or non-overlapping domain-decomposition (depending on the grid). However, only the two-dimensional surface grid is partitioned whereas the vertical direction is kept on one process due to the strong coupling. Thus there is also no need for communication of surface water height for the coupling, as the relevant data is always stored in the same process. The linear equation systems are solved with the BiCGstab-solver from DUNE-ISTL with Block-ILU0 as preconditioner. The much larger mesh size in horizontal direction compared to the vertical direction results in strong coupling of the unknowns in the vertical direction. The Block-ILU0 scheme provides an almost exact solver of the strongly coupled blocks in the vertical direction and is thus a very effective scheme. Furthermore, one generally has a limited number of cells in the vertical direction and extends the mesh in horizontal direction to simulate larger regions. Thus the good properties of the solver are retained when scaling up the size of the system.

6.2 Performance Optimisations

As the time steps in the explicit scheme for surface flow can get very small due to the stability limit, a significant speedup can be achieved by using a semi-implicit scheme, where the non-linear coefficients are calculated with the solution from the previous time step or iteration. However, if the surface is nearly completely dry, this could lead to numerical problems, thus an explicit scheme is still used under nearly dry conditions with an automatic switching between both.

While matrix-free DG solvers with sum-factorization can yield excellent per node performance (Sect. 3.3), it is a rather tedious task to implement them for new partial differential equations. Therefore, a code-generation framework is currently being developed in a project related to EXA-DUNE [33]. The framework is used to implement an alternative optimized version of the solver for the Richards equation as this is the computationally most expensive part of the computations. Expensive material functions like the van Genuchten model including several power functions are replaced by cubic spline approximations, which allow a fast vectorized incorporation of flexible material functions to simulate strongly heterogeneous systems. Sum-factorisation is used in the operator evaluations for the DG-discretization with a selectable polynomial degree.

A special pillar grid has been developed as a first realisation of a 2.5D grid [33]. It adds a vertical dimension to a two-dimensional grid (which is either structured or unstructured). However, as the current version still produces a full three-dimensional mesh at the moment, future developments are necessary to exploit the full possibilities of the approach.

6.3 Scalability and Performance Tests

Extensive tests covering infiltration as well as exfiltration have been performed (e.g. Fig. 16) to test the implementation and the new coupling condition. Good scalability is achieved in strong as well as in weak scaling experiments on up to 256 nodes and 4096 cores of the bwForCluster in Heidelberg (Fig. 17). Simulations for a large region with topographical data taken from a digital elevation model (Fig. 18) have been conducted as well.

With the generated code-based solver for the Richards equation a substantial fraction of the system's peak performance (up to 60% on a Haswell-CPU) can be

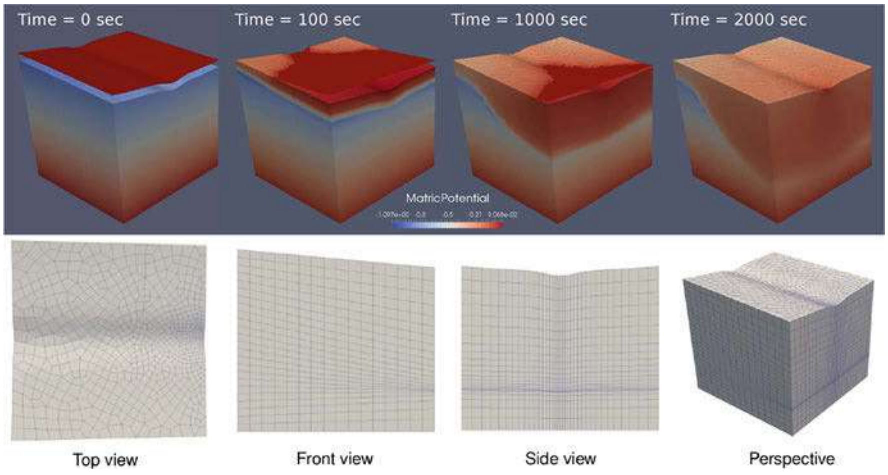


Fig. 16 Surface runoff and infiltration of 5 cm water into a dry coarse sand (top) and the unstructured 2.5D mesh used for the simulations (bottom)

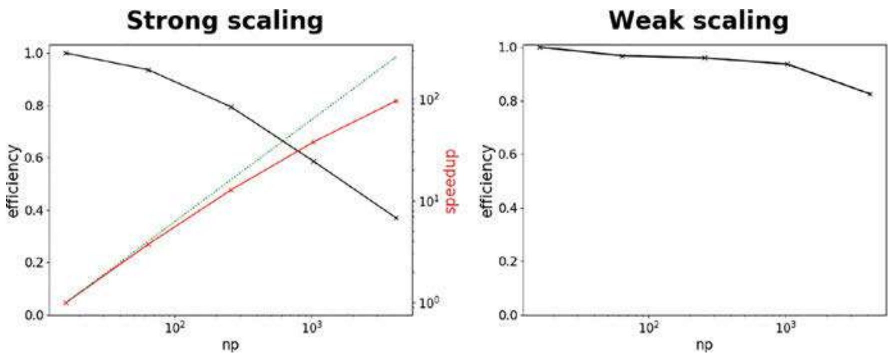


Fig. 17 Results of a strong (left) and weak (right) scalability test with a coupled run-off and infiltration experiment on 1 to 256 nodes (16 to 4096 Intel Xeon E5-2630 v3 2.4 GHz CPU cores) of the bwForCluster at IWR in Heidelberg

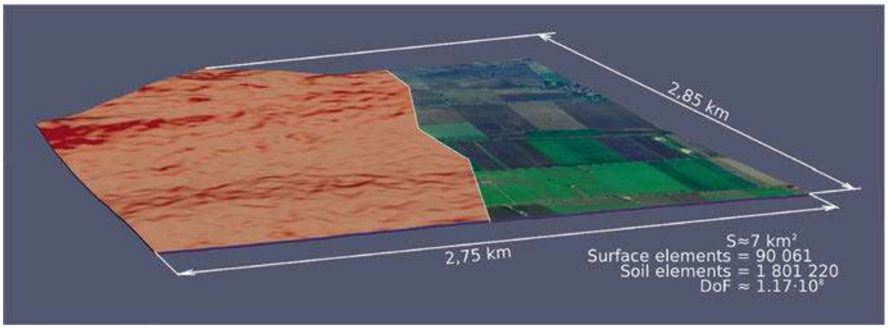


Fig. 18 Pressure distribution with an overlay of the landscape taken from Google Earth calculated in a simulation of water transport in a real landscape south of Brunswick simulated on 30 nodes with 1200 cores of HLRN-IV in Göttingen (2× Intel Skylake Gold 6148 2.4 GHz CPUs)

utilized due to the combination of sum factorization and vectorisation (Fig. 19). For the Richards equation (as for other PDEs before) the number of millions of degrees of freedom per second is independent of the polynomial degree with this approach. We measure a total speedup of 3 compared to the naive implementation in test simulations on a Intel Haswell Core i7-5600U 2.6 GHz CPU with first order DG base functions on a structured $32 \times 32 \times 32$ mesh for subsurface and 32×32 grid for surface flow. Even higher speedups are expected with higher-order base functions and matrix-free iterative solvers. The fully-coupled combination of the Richards solver obtained with the code generation framework and surface-runoff is tested with DG up to fourth order on structured as well as on unstructured grids. Parallel simulations are possible as well.

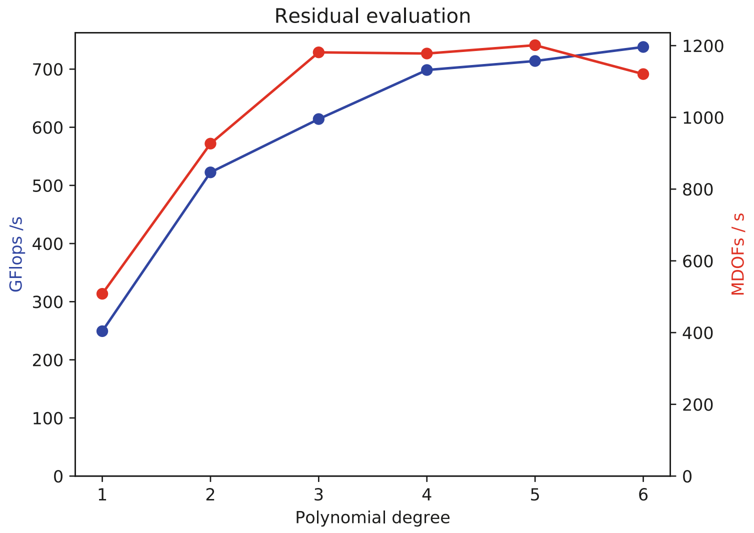


Fig. 19 Performance of the Richards solver implemented with the code generation framework for EXA-DUNE

7 Conclusion

In EXA-DUNE we extended the DUNE software framework in several directions to make it ready for the exascale architectures of the future which will exhibit a significant increase in node level performance through massive parallelism in form of cores and vector instructions. Software abstractions are now available that enable asynchronicity as well as parallel exception handling and several use cases for these abstractions have been demonstrated in this paper: resilience in multigrid solvers and several variants of asynchronous Krylov methods. Significant progress has been achieved in hardware-aware iterative linear solvers: we developed preconditioners for the GPU based on approximate sparse inverses, developed matrix-free operator application and preconditioners for higher-order DG methods and our solvers are now able to vectorize over multiple right hand sides. These building blocks have then been used to implement adaptive localized reduced basis methods, multilevel Monte-Carlo methods and a coupled surface-subsurface flow solver on up to 14k cores. The EXA-DUNE project has spawned a multitude of research projects, running and planned, as well as further collaborations in each of the participating groups. We conclude that the DUNE framework has made a major leap forward due to the EXA-DUNE project and work on the methods investigated here will continue in future research projects.

References

1. Abadi, M.: TensorFlow: large-scale machine learning on heterogeneous systems (2015). http://tensorflow.org/
2. Alonso, R., Santillana, M., Dawson, C.: On the diffusive wave approximation of the shallow water equations. Eur. J. Appl. Math. **19**(5), 575–606 (2008)
3. Amestoy, P.R., Duff, I.S., Koster, J., J.-Y., L.: A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM J. Matrix Anal. Appl. **23**(1), 15–41 (2001)
4. Ascher, D., Dubois, P., Hinsen, K., Hugunin, J., Oliphant, T.: Numerical Python. Lawrence Livermore National Laboratory, Livermore (1999)
5. Bastian, P.: A fully-coupled discontinuous Galerkin method for two-phase flow in porous media with discontinuous capillary pressure. Computat. Geosci. **18**(5), 779–796 (2014)
6. Bastian, P., Blatt, M., Dedner, A., Engwer, C., Klöfkorn, R., Kornhuber, R., Ohlberger, M., Sander, O.: A generic grid interface for parallel and adaptive scientific computing. Part II: Implementation and tests in DUNE. Computing **82**(2–3), 121–138 (2008). https://doi.org/10.1007/s00607-008-0004-9
7. Bastian, P., Blatt, M., Dedner, A., Engwer, C., Klöfkorn, R., Ohlberger, M., Sander, O.: A generic grid interface for parallel and adaptive scientific computing. Part I: abstract framework. Computing **82**(2–3), 103–119 (2008). https://doi.org/10.1007/s00607-008-0003-x
8. Bastian, P., Blatt, M., Scheichl, R.: Algebraic multigrid for discontinuous Galerkin discretizations of heterogeneous elliptic problems. Numer. Linear Algebra Appl. **19**(2), 367–388 (2012). https://doi.org/10.1002/nla.1816
9. Bastian, P., Engwer, C., Fahlke, J., Geveler, M., Göddeke, D., Iliev, O., Ippisch, O., Milk, R., Mohring, J., Müthing, S., Ohlberger, M., Ribbrock, D., Turek, S.: Advances concerning multiscale methods and uncertainty quantification in EXA-DUNE. In: Software for Exascale Computing—SPPEXA 2013–2015. Lecture Notes in Computational Science and Engineering. Springer Berlin (2016)
10. Bastian, P., Müller, E., Müthing, S., Piatkowski, M.: Matrix-free multigrid block-preconditioners for higher order discontinuous Galerkin discretisations. J. Comput. Phys. **394**, 417–439 (2019). https://doi.org/10.1016/j.jcp.2019.06.001
11. Bland, W., Bosilca, G., Bouteiller, A., Herault, T., Dongarra, J.: A proposal for user-level failure mitigation in the MPI-3 standard. Department of Electrical Engineering and Computer Science, University of Tennessee (2012)
12. Buis, P., Dyksen, W.: Efficient vector and parallel manipulation of tensor products. ACM Trans. Math. Softw. **22**(1), 18–23 (1996). https://doi.org/10.1145/225545.225548
13. Cantwell, C., Nielsen, A.: A minimally intrusive low-memory approach to resilience for existing transient solvers. J. Sci. Comput. **78**(1), 565–581 (2019). https://doi.org/10.1007/s10915-018-0778-7
14. Christie, M., Blunt, M.: Tenth SPE comparative solution project: a comparison of upscaling techniques. In: SPE Reservoir Simulation Symposium (2001). https://doi.org/10.2118/72469-PA
15. Chronopoulos, A., Gear, C.W.: s-step iterative methods for symmetric linear systems. J. Comput. Appl. Math. **25**(2), 153–168 (1989)
16. Dongarra, J., Hittinger, J., Bell, J., Chacón, L., Falgout, R., Heroux, M., Hovland, P., Ng, E., Webster, C., Wild, S., Pao, K.: Applied mathematics research for exascale computing. Tech. rep., U. S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program (2014). http://science.energy.gov/~/media/ascr/pdf/research/am/docs/EMWGreport.pdf
17. Elfverson, D., Ginting, V., Henning, P.: On multiscale methods in Petrov–Galerkin formulation. Numer. Math. **131**(4), 643–682 (2015). https://doi.org/10.1007/s00211-015-0703-z
18. Engwer, C., Dreier, N.A., Altenbernd, M., Göddeke, D.: A high-level C++ approach to manage local errors, asynchrony and faults in an MPI application. In: Proceedings of 26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2018) (2018)

19. Engwer, C., Müthing, S.: Concepts for flexible parallel multi-domain simulations. In: Dickopf, T., Gander, M.J., Halpern, L., Krause, R., Pavarino, L.F. (eds.) Domain Decomposition Methods in Science and Engineering XXII, pp. 187–195. Springer, Berlin (2016)
20. Ern, A., Stephansen, A., Vohralík, M.: Guaranteed and robust discontinuous Galerkin a posteriori error estimates for convection-diffusion-reaction problems. J. Comput. Appl. Math. **234**(1), 114–130 (2010). https://doi.org/10.1016/j.cam.2009.12.009
21. Ern, A., Stephansen, A., Zunino, P.: A discontinuous Galerkin method with weighted averages for advection-diffusion equations with locally small and anisotropic diffusivity. IMA J. Numer. Anal. **29**(2), 235–256 (2009). https://doi.org/10.1093/imanum/drm050
22. Fehn, N., Wall, W., Kronbichler, M.: A matrix-free high-order discontinuous Galerkin compressible Navier–Stokes solver: a performance comparison of compressible and incompressible formulations for turbulent incompressible flows. Numer. Methods Fluids **89**(3), 71–102 (2019)
23. Fog, A.: VCL C++ vector class library (2017). http://www.agner.org/optimize/vectorclass.pdf
24. Fritze, R., Leibner, T., Schindler, F.: dune-gdt (2016). https://doi.org/10.5281/zenodo.593009
25. Gagliardi, F., Moreto, M., Olivieri, M., Valero, M.: The international race towards exascale in Europe. CCF Trans. High Perform. Comput. **1**(1), 3–13 (2019). https://doi.org/10.1007/s42514-019-00002-y
26. Geveler, M., Ribbrock, D., Göddeke, D., Zajac, P., Turek, S.: Towards a complete FEM-based simulation toolkit on GPUs: unstructured grid finite element geometric multigrid solvers with strong smoothers based on sparse approximate inverses. Comput. Fluids **80**, 327–332 (2013). https://doi.org/10.1016/j.compfluid.2012.01.025
27. Ghysels, P., Vanroose, W.: Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. Parallel Comput. **40**(7), 224–238 (2014)
28. Gmeiner, B., Huber, M., John, L., Rüde, U., Wohlmuth, B.: A quantitative performance study for Stokes solvers at the extreme scale. J. Comput. Sci. **17**, 509–521 (2016). https://doi.org/10.1016/j.jocs.2016.06.006
29. Göddeke, D., Altenbernd, M., Ribbrock, D.: Fault-tolerant finite-element multigrid algorithms with hierarchically compressed asynchronous checkpointing. Parallel Comput. **49**, 117–135 (2015). https://doi.org/10.1016/j.parco.2015.07.003
30. Gupta, S., Patel, T., Tiwari, D., Engelmann, C.: Failures in large scale systems: long-term measurement, analysis, and implications. In: Proceedings of the 30th IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) (2017)
31. Huber, M., Gmeiner, B., Rüde, U., Wohlmuth, B.: Resilience for massively parallel multigrid solvers. SIAM J. Sci. Comput. **38**(5), S217–S239
32. Iliev, O., Mohring, J., Shegunov, N.: Renormalization based MLMC method for scalar elliptic SPDE. In: International Conference on Large-Scale Scientific Computing, pp. 295–303 (2017)
33. Kempf, D.: Code generation for high performance PDE solvers on modern architectures. Ph.D. thesis, Heidelberg University (2019)
34. Kempf, D., Heß, R., Müthing, S., Bastian, P.: Automatic code generation for high-performance discontinuous Galerkin methods on modern architectures (2018). arXiv:1812.08075
35. Keyes, D.: Exaflop/s: the why and the how. Comptes Rendus Mécanique **339**(2–3), 70–77 (2011). https://doi.org/10.1016/j.crme.2010.11.002
36. Klawonn, A., Köhler, S., Lanser, M., Rheinbach, O.: Computational homogenization with million-way parallelism using domain decomposition methods. Comput. Mech. **65**(1), 1–22 (2020). https://doi.org/10.1007/s00466-019-01749-5
37. Kohl, N., Thönnes, D., Drzisga, D., Bartuschat, D., Rüde, U.: The HyTeG finite-element software framework for scalable multigrid solvers. Int. J. Parallel Emergent Distrib. Syst. **34**(5), 477–496 (2019). https://doi.org/10.1080/17445760.2018.1506453
38. Kretz, M., Lindenstruth, V.: Vc: A C++ library for explicit vectorization. Softw. Practice Exp. **42**(11), 1409–1430 (2012). https://doi.org/10.1002/spe.1149
39. Kronbichler, M., Kormann, K.: A generic interface for parallel cell-based finite element operator application. Comput. Fluids **63**, 135–147 (2012). https://doi.org/10.1016/j.compfluid.2012.04.012

40. Lengauer, C., Apel, S., Bolten, M., Chiba, S., Rüde, U., Teich, J., Größlinger, A., Hannig, F., Köstler, H., Claus, L., Grebhahn, A., Groth, S., Kronawitter, S., Kuckuk, S., Rittich, H., Schmitt, C., Schmitt, J.: ExaStencils: Advanced multigrid solver generation. In: Software for Exascale Computing—SPPEXA 2nd phase. Springer, Berlin (2020)
41. Liang, X., Di, S., Tao, D., Li, S., Li, S., Guo, H., Chen, Z., Cappello, F.: Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In: IEEE Big Data, pp. 438–447 (2018). https://doi.org/10.1109/BigData.2018.8622520
42. Lu, Y.: Paving the way for China exascale computing. CCF Trans. High Perform. Comput. **1**(2), 63–72. (2019). https://doi.org/10.1007/s42514-019-00010-y
43. Luporini, F., Ham, D.A., Kelly, P.H.J.: An algorithm for the optimization of finite element integration loops. ACM Trans. Math. Softw. **44**(1), 3:1–3:26 (2017). https://doi.org/10.1145/3054944
44. Message Passing Interface Forum: MPI: a message-passing interface standard version 3.1 (2015). https://www.mpi-forum.org/docs/
45. Milk, R., Rave, S., Schindler, F.: pyMOR—generic algorithms and interfaces for model order reduction. SIAM J. Sci. Comput. **38**(5), S194–S216 (2016). https://doi.org/10.1137/15M1026614
46. Milk, R., Schindler, F., Leibner, T.: Extending DUNE: the dune-xt modules. Arch. Numer. Softw. **5**(1), 193–216 (2017). https://doi.org/10.11588/ans.2017.1.27720
47. Mohring, J., Milk, R., Ngo, A., Klein, O., Iliev, O., Ohlberger, M., Bastian: uncertainty quantification for porous media flow using multilevel Monte Carlo. In: International Conference on Large-Scale Scientific Computing, pp. 145–152 (2015)
48. Müthing, S., Piatkowski, M., Bastian, P.: High-performance implementation of matrix-free high-order discontinuous Galerkin methods (2017). arXiv:1711.10885
49. Ohlberger, M., Schindler, F.: Error control for the localized reduced basis multiscale method with adaptive on-line enrichment. SIAM J. Sci. Comput. **37**(6), A2865–A2895 (2015). https://doi.org/10.1137/151003660
50. Orszag, S.: Spectral methods for problems in complex geometries. J. Comput. Phys. **37**(1), 70–92 (1980). https://doi.org/10.1016/0021-9991(80)90005-4
51. Piatkowski, M., Müthing, S., Bastian, P.: A stable and high-order accurate discontinuous Galerkin based splitting method for the incompressible Navier–Stokes equations. J. Comput. Phys. **356**, 220–239 (2018). https://doi.org/10.1016/j.jcp.2017.11.035
52. Richards, L.A.: Capillary conduction of liquids through porous mediums. Physics **1**, 318–333 (1931)
53. Ruelmann, H., Geveler, M., Turek, S.: On the prospects of using machine learning for the numerical simulation of PDEs: training neural networks to assemble approximate inverses. Eccomas Newsletter 27–32 (2018)
54. Teranishi, K., Heroux, M.: Toward local failure local recovery resilience model using MPI-ULFM. In: Proceedings of the 21st European MPI Users' Group Meeting, EuroMPI/ASIA '14, pp. 51:51–51:56 (2014). https://doi.org/10.1145/2642769.2642774
55. Turek, S., Göddeke, D., Becker, C., Buijssen, S., Wobker, S.: FEAST—realisation of hardware-oriented numerics for HPC simulations with finite elements. Concurrency Comput. Pract. Exp. **22**(6), 2247–2265 (2010)
56. Vohralík, M.: A posteriori error estimates for lowest-order mixed finite element discretizations of convection-diffusion-reaction equations. SIAM J. Numer. Anal. **45**(4), 1570–1599 (2007). https://doi.org/10.1137/060653184
57. van der Walt, S., Colbert, S.C., Varoquaux, G.: The NumPy array: a structure for efficient numerical computation. Comput. Sci. Eng. **13**(2), 22–30 (2011). https://doi.org/10.1109/MCSE.2011.37
58. Xu, J.: Iterative methods by space decomposition and subspace correction. SIAM Rev. **34**(4), 581–613 (1992). https://doi.org/10.1137/1034116

ExaFSA: Parallel Fluid-Structure-Acoustic Simulation

Florian Lindner, Amin Totounferoush, Miriam Mehl, Benjamin Uekermann, Neda Ebrahimi Pour, Verena Krupp, Sabine Roller, Thorsten Reimann, Dörte C. Sternel, Ryusuke Egawa, Hiroyuki Takizawa, and Frédéric Simonis

Abstract In this paper, we present results of the second phase of the project ExaFSA within the priority program SPP1648—Software for Exascale Computing. Our task was to establish a simulation environment consisting of specialized highly efficient and scalable solvers for the involved physical aspects with a particular focus on the computationally challenging simulation of turbulent flow and propagation of the induced acoustic perturbations. These solvers are then coupled in a modular, robust, numerically efficient and fully parallel way, via the open source coupling library preCICE. Whereas we made a first proof of concept for a three-field simulation (elastic structure, surrounding turbulent acoustic flow in the near-field, and pure acoustic wave propagation in the far-field) in the first phase, we removed several scalability limits in the second phase. In particular, we present new contributions to (a) the initialization of communication between processes of

F. Lindner · A. Totounferoush · M. Mehl (✉)
University of Stuttgart, Stuttgart, Germany
e-mail: florian.lindner@ipvs.uni-stuttgart.de; amin.totounferoush@ipvs.uni-stuttgart.de; miriam.mehl@ipvs.uni-stuttgart.de

B. Uekermann
Eindhoven University of Technology, Eindhoven, Netherlands
e-mail: b.w.uekermann@tue.nl

N. Ebrahami Pour · V. Krupp · S. Roller
University of Siegen, Siegen, Germany
e-mail: neda.epour@uni-siegen.de; sabine.roller@uni-siegen.de

T. Reimann · D. C. Sternel
Technische Universität Darmstadt, Darmstadt, Germany
e-mail: thorsten.reimann@sc.tu-darmstadt.de; doerte.sternel@hpc-hessen.de

R. Egawa · H. Takizawa
Tohoku University, Sendai, Japan
e-mail: takizawa@tohoku.ac.jp

F. Simonis
Technical University of Munich, München, Germany
e-mail: simonis@in.tum.de

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_10

the involved independent solvers, (b) optimization of the parallelization of data mapping, (c) solver-specific white-box data mapping providing higher efficiency but less flexibility, (d) portability and scalability of the flow and acoustic solvers FASTEST and Ateles on vector architectures by means of code transformation, (e) physically correct information transfer between near-field acoustic flow and far-field acoustic propagation.

1 Introduction

The simulation of fluid-structure-acoustic interactions is a typical example for multi-physics simulations. Two fundamentally different physical sound sources can be distinguished: structural noise and flow-induced noise. As we are interested in accurate results for the resulting sound emissions induced from the turbulent flow, it is decisive to include not only the turbulent flow, but also the structure deformation and the interaction between both. High accuracy requires the use of highly resolved grids. As a consequence, the use of massively parallel supercomputers is inevitable. When we are interested in the sound effects far away from a flow induced fluttering structure, the simulation becomes too expensive, even for supercomputing architectures. Hence, we introduce an assumption, we call it the "far-field". Far from the structure and, thus, the noise generation, we assume a homogeneous background flow and restrict the simulation in this part of the domain to the propagation of acoustic waves. This results in an overall setup with two coupling surfaces—between the elastic structure and the surrounding flow, and between the near-field and the far-field in the flow domain (see Fig. 1 for an illustrative example). Such a complex simulation environment implies several new challenges compared to

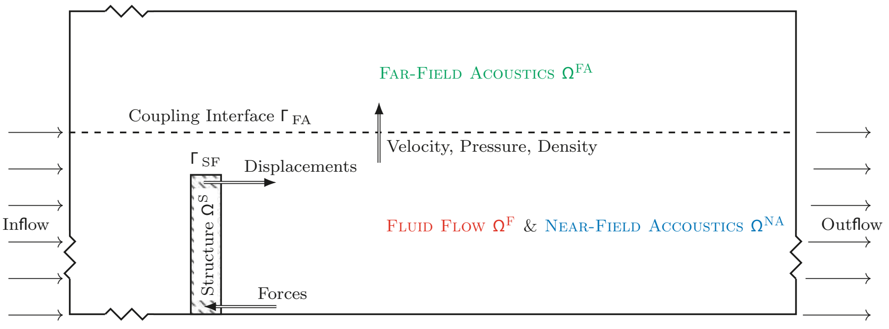

Far-Field Acoustics Ω^{FA}
Coupling Interface Γ_{FA}
Velocity, Pressure, Density
Γ_{SF}
Displacements
Structure Ω^S
Inflow
Fluid Flow Ω^F & Near-Field Accoustics Ω^{NA}
Outflow
Forces

Fig. 1 Multiphysics fluid-structure-acoustic scenario as used in our simulations in Sect. 6. The domain is decomposed into a near-field 'incompressible flow region' $\Omega^F = \Omega^{NA}$, a far-field 'acoustic only region' Ω^{FA}, and an 'elastic structure region' Ω^S. Note that the geometry is not scaled correctly for better illustration

"single-physics" simulations: (a) multi-scale properties in space and time (small-scale processes around the structure, multi-scale turbulent flow in the near-field, and large-scale processes in the acoustic far-field), (b) different optimal discretization and solver choices for the three fields, (c) highly ill-conditioned problem, if formulated and discretized as a single large system of equations, (d) challenging load-balancing due to different computational load per grid unit depending on the local physics.

Application examples for fluid-structure-acoustic simulations can be found in several technical systems: wind power plants, fans in air conditioning systems of buildings, cars or airplanes, car mirrors and other design details of a car frame, turbines, airfoil design, etc.

Fluid-structure interaction simulations as a sub-problem of our target system have been in the focus of research in computational engineering for many years, mainly aiming at capturing stresses in the structure more realistically than with a pure flow simulation. A main point of discussion in this field is the question whether monolithic approaches—treating the coupled problem as a single large system of equations—or partitioned methods—glueing together separate simulation modules for structures and fluid flow by means of suitable coupling numerics and tools—are more appropriate and efficient. Monolithic approaches require a new implementation of the simulation code as well as the development of specialized iterative solvers for the ill-conditioned overall system of equations, but can achieve very high efficiency and accuracy [3, 12, 19, 23, 38]. Partitioned approaches, on the other hand, offer large flexibility in choosing optimal solvers for each field, adding additional fields, or exchanging solvers. The difficulty here lies in both a stable, accurate, and efficient coupling between independent solvers applying different numerical methods and in establishing efficient communication and load balancing between the used parallel codes. For numerical coupling, numerous efficient data mapping methods [5, 26, 27, 32] have been published along with efficient iterative solvers [2, 7, 13, 20, 29, 35, 39, 41]. In [6], various monolithic and partitioned approaches have been proposed and evaluated in terms of a common benchmark problem. Three-field fluid-structure-acoustic interaction in the literature has so far been restricted to near-field simulations due to the intense computational load [28, 33].

To realize a three-field fluid-structure-acoustic interaction including the far-field, we use a partitioned approach and couple existing established "single-physics" solvers in a black-box fashion. We couple the finite volume solver FASTEST [18], the discontinuous Galerkin solver Ateles [42], and the finite element solver CalculiX [14] by means of the coupling library preCICE [8]. We compare this approach to a less flexible white-box coupling implemented in APESmate [15] as part of the APES framework and make use of the common data-structure within APES [31]. The assumption which is confirmed in this paper is, that the white-box approach is more efficient, but puts some strict requirements on the codes to be coupled, while the black-box approach is a bit less efficient, but much more flexible with respect to the codes that can be used. Our contributions to the field of fluid-structure-acoustic interaction, which we summarize in this paper, include:

1. For the near-field flow, we introduce a volume coupling between background flow and acoustic perturbations in FASTEST accounting for the multi-scale properties in space and time by means of different spatial and time resolution.
2. For both near-field flow and far-field acoustics, we achieved portability and performance optimization of Ateles and FASTEST for vector machines by means of code transformation.
3. In terms of inter-field coupling, we
   (a) increased the efficiency of inter-code communication by means of a new hierarchical implementation of communication initialization and a modified communicator concept,
   (b) we improved the robustness and efficiency of radial basis function mapping,
   (c) we identified correct interface conditions between near-field and far-field, optimized the position of the interface, and ensured correct boundary conditions by overlapping near-field and far-field,
   (d) we developed and implemented implicit quasi-Newton coupling numerics that allow for a simultaneous execution of all involved solvers.
4. For a substantially improved inter-code load balancing, we use a regression-based performance model for all involved solvers and perform an optimization of assigned cores.
5. We present a comparison of our black-box and to the white-box approach for multi-physics coupling.

These contributions have been achieved as a result of the project ExaFSA—a cooperation between the Technische Universität Darmstadt, the University of Siegen, the University of Stuttgart, and the Tohoku University (Japan) in the Priority Program SPP 1648—Software for Exascale Computing of the German Research Foundation (DFG) in close collaboration with the Technical University of Munich. In the first funding phase (2013–2016), we showed that efficient yet robust coupled simulations are feasible and can be enhanced with an in-situ visualization component as an additional software part, but we still reached limits in terms of scalability and load balancing [4, 9]. This paper focuses on results of the second funding phase (2016–2019) and demonstrates significant improvements in scalability and accuracy as well as robustness based on the above-mentioned contributions.

In the following, we introduce the underlying model equations of our target scenarios in Sect. 2 and present our solvers and their optimization in Sect. 3 as well as the black-box coupling approach and new contributions in terms of coupling in Sect. 4. In Sect. 5, we compare black-box coupling to an alternative, efficient, but solver-specific and, thus, less flexible white-box coupling for uni-directional flow-acoustic coupling. Finally, results for a turbulent flow over a fence scenario are presented in Sect. 6.

2 Model

In this section, we shortly introduce the underlying flow, acoustic and structure models of our target application. We use the Einstein summation convention throughout this section.

2.1 Governing Equations

The multi-physics scenario we investigate describes an elastic structure embedded in a turbulent flow field. The latter is artificially decomposed into a near-field and a far-field. See Fig. 1 for an example.

Near-Field Flow In the near-field region $\Omega^{\mathrm{F}} = \Omega^{\mathrm{NA}}$, the compressible fluid flow is modeled by means of the density $\overline{\rho}$, the velocity $\overline{u_i}$ and the pressure $\overline{p}$. As we focus on a low Mach number regime, we can split these variables into an incompressible part ρ, u_i, p, and acoustic perturbations ρ', u_i', p':

$$\overline{\rho} = \rho + \rho', \quad \overline{u_i} = u_i + u_i', \quad \overline{p} = p + p' \,. \tag{1}$$

The incompressible flow is described by the Navier-Stokes equations[1]

$$\frac{\partial u_i}{\partial x_i} = 0, \tag{2a}$$

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_j}\left(\rho u_i u_j - \tau_{ij}\right) = \rho f_i \,, \tag{2b}$$

where ρ is the density of the fluid, and f_i summarizes external force density terms. The incompressible stress tensor τ_{ij} for a Newtonian fluid is described by

$$\tau_{ij} = -p\delta_{ij} + \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \,, \tag{3}$$

with μ representing the dynamic viscosity and δ_{ij} the Kronecker-Delta.

[1]To capture the moving structure within the near-field, we actually formulate all near-field equations in an arbitrary Lagrian-Eulerian perspective. For the relative mesh velocity, we use a block-wise elliptic mesh movement as described in [30]. As we do not show fluid-structure interaction in this contribution, however, we formulate all near-field equations in a pure Eulerian perspective for the sake of simplicity.

Acoustic Wave Propagation The propagation of acoustic perturbations in both the near-field and the far-field is modeled by the linearized Euler equations, where in the far-field a constant background state is assumed (which implies $\frac{\partial p}{\partial t} = 0$).

$$\frac{\partial \rho'}{\partial t} + \rho \frac{\partial u_i'}{\partial x_i} + u_i \frac{\partial \rho'}{\partial x_i} = 0 \tag{4a}$$

$$\rho \frac{\partial u_i'}{\partial t} + \rho u_j \frac{\partial u_i'}{\partial x_j} + \frac{\partial p'}{\partial x_i} = 0 \tag{4b}$$

$$\frac{\partial p'}{\partial t} + \rho c^2 \frac{\partial u_i'}{\partial x_i} + u_i \frac{\partial p'}{\partial x_i} = -\frac{\partial p}{\partial t} \tag{4c}$$

Here c is the speed of sound. In the near-field, the background flow quantities u_i and p are calculated from (2), whereas they are assumed to be constant in the acoustic far-field. The respective constant value is read from the coupling interface with the near-field, which implies that the interface has to be chosen such that the background flow values are (almost) constant at the coupling interface. In both cases, the coupling between background-flow and acoustic perturbations is unidirectional from the background flow to the acoustic equations (4) by means of p and u_i.

Elastic Structure The structural subdomain Ω^{S} is governed by the equations of motion, here in Lagrangian description:

$$\rho^{\mathrm{S}} \frac{\partial^2 \vartheta_i}{\partial t^2} = \frac{\partial S_{jk} F_{ik}}{\partial X_j^{\mathrm{S}}} + \rho^{\mathrm{S}} f_i^{\mathrm{S}}. \tag{5}$$

With $x_i^{\mathrm{S}} = X_i^{\mathrm{S}} + \vartheta_i$ being the position of a particle in the current configuration, X_i^{S} is the position of a particle in the reference configuration, and ϑ_i the displacement. F_{ij} is the deformation gradient. S_{ij} is the second Piola-Kirchhoff tensor, and ρ^{S} describes the structural density. The Cauchy stress tensor τ_{ij}^{S} relates to S_{ij} via

$$\tau_{ij}^{\mathrm{S}} = \frac{1}{\det\left(F_{ij}\right)} F_{ik} S_{kl} F_{jl}\,. \tag{6}$$

We assume linear elasticity to describe the stress-strain relation.

The coupling between fluid and structure is bi-directional by means of dynamic and kinematic conditions, i.e., equality of interface displacements/velocities and stresses, i.e.,

$$u_i^{\Gamma^{\mathrm{F}}} = \frac{\partial \vartheta_i^{\Gamma^{\mathrm{S}}}}{\partial t}, \qquad \tau_{ij}^{\Gamma^{\mathrm{F}}} = \tau_{ij}^{\Gamma^{\mathrm{S}}} \tag{7}$$

at $\Gamma^{\mathrm{I}} = \Gamma^{\mathrm{S}} \cap \Gamma^{\mathrm{F}}$ with $\Gamma^{\mathrm{F}} = \partial\Omega^{\mathrm{F}}$ and $\Gamma^{\mathrm{S}} = \partial\Omega^{\mathrm{S}}$.

3 Solvers and Their Optimization

Following a partitioned approach, the respective subdomains of the multi-physics model as described in Sect. 2 (elastic structure domain, near-field, and far-field) are treated by different solvers. We employ the flow solver FASTEST presented in Sect. 3.1 to solve for the incompressible flow equations, Eq. (2), and near-field acoustics equations, Eq. (4), the Ateles solver described in Sect. 3.2 for the far-field acoustics equations, Eq. (4), and finally the structural solver CalculiX introduced in Sect. 3.3 for the deformation of the obstacle, Eq. (5). For performance optimization of FASTEST and Ateles, we make use of the Xevolver framework, which has been developed to separate system-specific performance concerns from application codes. We report on the optimization of both solvers further below.

3.1 FASTEST

FASTEST is used to solve both the incompressible Navier-Stokes equations (2) and the linearized Euler equations (4) in the near-field.

Capabilities and Numerical Methods The flow solver FASTEST [24] solves the three-dimensional incompressible Navier-Stokes equations. The equations are discretized utilizing a second-order finite-volume approach with implicit time-stepping, which is also second order accurate. Field data are evaluated on a non-staggered, body-fitted, and block-structured grid. The equations are solved according to the SIMPLE scheme [11], and the resulting linear equation system is solved by ILU factorization [36]. Geometrical multi-grid is employed for convergence acceleration. The code generally follows a hybrid parallelization strategy employing MPI and OpenMP. FASTEST can account for different flow phenomena, and has the capability to model turbulent flow with different approaches. In our test case example, we employ a detached-eddy simulation (DES) based on the $\zeta - \mathrm{f}$ turbulence model [30].

In addition, FASTEST contains a module to solve the linearized Euler equations to describe low Mach number aeroacoustic scenarios, which are solved by a second order Lax-Wendroff scheme with various limiters.

Since all equation sets are discretized on the same numerical grid, advantage can be taken from the multi-grid capabilities to account for the scale discrepancies of the fluid flow and the acoustics. Since the spatial scales of the acoustics are considerably larger than those of the flow, a coarser grid level can be used for them. In return, the finer temporal scales can be considered by sub-cycling a CFD time step with various CAA time steps. This way a very efficient implementation of the viscous/acoustic splitting approach can be realized.

Performance Optimization Concerning performance optimization, one interesting point of FASTEST is that some of its kernels were once optimized for old vector machines, and thus important kernels have their vector versions in addition to the default ones. The main difference between the two versions is that nested loops in the default version are collapsed into one loop in the vector version. Since the loops skip accessing halo regions, the compiler is not able to automatically collapse the loops, resulting in short vector lengths even if the compiler can vectorize them. To efficiently run the solver on a vector system, performance engineers usually need to manually change the loop structures. In this project, Xevolver is used to express the differences between the vector and default versions as code transformation rules. In other words, vectorization-aware loop optimizations are expressed as code transformations. As a result, the default version can be transformed to its vector version, and the vector version does not need to be maintained any longer to achieve high performance on vector systems. That is, the FASTEST code can be simplified without reducing the vector performance by using the Xevolver approach. Ten rules are defined to transform the default kernels in FASTEST to their vector kernels. Those code transformations plus some system-independent minor code modifications for removing vectorization obstacles can reduce the execution time on the NEC SX-ACE vector system by about 85%, when executing a simple test case that models a three-dimensional Poiseuille flow through a channel based on the Navier-Stokes equations, in which the mesh contains two blocks with 426,000 cells each. The code execution on the SX-ACE vector processor works about 2.7 times faster than on the Xeon E5-2695v2 processor, since the kernel is memory-intensive and the memory bandwidth of SX-ACE is 4$\times$ higher than that of Xeon. Therefore, it is clearly demonstrated that the Xevolver approach is effective to achieve both high performance portability and high code maintainability for FASTEST.

3.2 Ateles

In our project, Ateles is used for the simulation of the acoustic far-field. Since acoustics scales need to be transported over a large distance, Ateles' high-order DG scheme can show its particular advantages of low dissipation and dispersion error in this test case.

Capabilities and Numerical Methods The solver Ateles is integrated in the simulation framework APES [31]. Ateles is based on the Discontinuous Galerkin (DG) discretization method, which can be seen as a hybrid method, combining the finite-volume and finite-element methods. DG is well suited for parallelization and the simulation of aero-acoustic problems, due to its inherent dissipation and dispersion properties. This method has several outstanding advantages, that are among others the high-order accuracy, the faster convergence of the solution with increasing scheme order and fewer elements compared to a low order scheme with a higher number of elements, the local h-p refinement as well as orthogonal hierarchical bases. The DG solver Ateles includes different equation systems, among others the compressible Navier-Stokes equations, the compressible inviscid Euler equations and the linearized Euler equations (used in this work for the acoustics far-field). For the time discretization, Ateles makes use of the explicit Runge-Kutta time stepping scheme, which can be either second or fourth order.

Performance Optimization Analyzing the performance of Ateles, originally developed assuming x86 systems, we found out that four kinds of code optimization techniques are needed for a total of 18 locations of the code in order to migrate the code to the SX-ACE system. Those techniques are mostly for collapsing the kernel loop and also for directing the NEC compiler to vectorize the loop. In this project, all the techniques are expressed as one common code transformation rule. The rule can take the option to change its transformation behaviors appropriately for each code location. This means that, to achieve performance portability between SX-ACE and x86 systems, only one rule needs to be maintained in addition to the Ateles application code. We executed a small testcase solving Maxwell equations with an 8th order DG scheme on 64 grid cells. The code transformation leads to $7.5\times$ higher performance. The significant performance improvement is attributed to loop collapse and insertion of appropriate compiler directives, which increases the vectorization length by a factor of 2 and the vectorization ratio from 71.35% to 96.72%. Finally, in terms of the execution time, the SX-ACE performance is 19% the performance of Xeon E5-2695v2. The code optimizations for SX-ACE reduce the performances of Xeon and Power8 by 14% and 6%, respectively. In this way, code optimizations for a specific system are often harmful to other systems. However, by using Xevolver, such a system-specific code optimization is expressed separately from the application code. Therefore, the Xevolver approach is obviously useful for achieving high performance portability across various systems without complicating the application code.

3.3 CalculiX

As structure solver, we use the well-established finite element solver CalculiX[14], developed by Guido Dhont und Klaus Wittig.[2] While CalculiX also supports static and thermal analysis, we only use it for dynamic non-linear structural mechanics. As our main research focus is not the structural computation per se, but the coupling within a fluid-structure-acoustic framework, we merely regard CalculiX as a black box. The preCICE adapter of CalculiX has been developed in [40].

[2]www.calculix.de.

4 A Black-Box Partitioned Coupling Approach Using preCICE

Our first and general coupling approach for the three-field simulation comprising (a) the elastic structure, (b) the near-field flow with acoustic equations, and (c) the far-field acoustic propagation follows a black-box idea, i.e., we only use input and output data of dedicated solvers at the interfaces between the respective domains for numerical coupling. Such a black-box coupling requires three main functional coupling components: intercode-communication, data-mapping between non-matching grids of independent solvers, and iterative coupling in cases with strong bi-directional coupling. preCICE is an open source library[3] that provides software modules for all three components. In the first phase of the ExaFSA project, we ported preCICE from a server-based to a fully peer-to-peer communication architecture [9, 39], increasing the scalability of the software from moderately to massively parallel. To this end, all coupling numerics needed to be parallelized on distributed data. During the second phase of the ExaFSA project, we focused on several costly initialization steps and further necessary algorithmic optimizations. In the following, we shortly sketch all components of preCICE with a particular focus on innovations introduced in the second phase of the ExaFSA project and on the actual realization of the fluid-acoustic coupling between near-field and far-field and the fluid-structure coupling.

4.1 (Iterative) Coupling

To simulate fluid-structure-acoustic interactions such as in the scenario shown in Fig. 1, two coupling interfaces have to be considered with different numerical and physical properties: (a) the coupling between fluid flow and the elastic structure requires an implicit bi-directional coupling, i.e., we exchange data in both directions and iterate in each time step until convergence; (b) the coupling between fluid flow and the acoustic far-field is uni-directional (neglecting reflections back into the near-field domain), i.e., results of the near-field fluid flow simulation are propagated to the far-field solver as boundary values once per time step. In order to fulfil the coupling conditions at the fluid-structure interface as given in Sect. 2, we iteratively solve the fixed-point equation

$$\begin{pmatrix} S(f) \\ F(u) \end{pmatrix} = \begin{pmatrix} u \\ f \end{pmatrix}, \tag{8}$$

[3]www.precice.org.

where f represents the stresses, u the velocities at the interface Γ_{FS}, S the effects of the structure solver on the interface (with stresses as an input and velocities as an output), F the effects of the fluid solver on the interface (with interface velocities as an input and stresses as an output). preCICE provides a choice of iterative methods accelerating the plain fixed-point iteration on Eq. (8). The most efficient and robust schemes are our quasi-Newton methods that are provided in a linear complexity (in terms of interface degrees of freedom) and fully parallel optimized versions [35]. As most of our achievements concerning iterative methods fall within the first phase of the ExaFSA project, we omit a more detailed description and refer to previous reports instead [9].

For the uni-directional coupling between the fluid flow in the near-field and the acoustic far-field, we transfer perturbation in density, pressure, and velocity from the flow domain to the far-field as boundary conditions at the interface. We do this once per acoustic time step, which is chosen to be the same for near-field and far-field acoustics, but which is much smaller than the fluid time step size (and the fluid-structure coupling), as described in Sect. 3.1.

Both domains are time-dependent and subject to mutual influence.

In an aeroacoustic setting, the near-field subdomain Ω^{NA} and far-field subdomain Ω^{FA}, with boundaries $\Gamma^{\mathrm{NA}} = \partial\Omega^{\mathrm{NA}}$ and $\Gamma^{\mathrm{FA}} = \partial\Omega^{\mathrm{FA}}$ are fixed, which means, all background information in the far-field are fixed to a certain value. Therefore there is only influence of Ω^{NA} onto Ω^{FA}, as backward propagation can be neglected. Then the continuity of shared state variables on the interface boundary $\Gamma^{\mathrm{IA}} = \Gamma^{\mathrm{NA}} \cap \Gamma^{\mathrm{FA}}$ is

$$\rho_i'^{\Gamma^{\mathrm{FA}}} = \rho_i'^{\Gamma^{\mathrm{NA}}}, u_i'^{\Gamma^{\mathrm{FA}}} = u_i'^{\Gamma^{\mathrm{NA}}}, p_i'^{\Gamma^{\mathrm{FA}}} = p_i'^{\Gamma^{\mathrm{NA}}} \quad . \tag{9}$$

4.2 Data Mapping

Our three solvers use different meshes adapted to their specific problem domain. To map data between the meshes, preCICE offers three different interpolation algorithms: (a) Nearest-neighbor interpolation is based on finding the geometrically nearest neighbor, i.e. the vertex with the shortest distance from the target or source vertex. It excels in its ease of implementation, perfect parallelizability, and low memory consumption. (b) Nearest-projection mapping can be regarded as an extension to the nearest-neighbor interpolation, working on nearest mesh elements (such as edges, triangles or quads) instead of merely vertices and interpolating values to the projection points. The method requires a suitable triangulation to be provided by the solver. (c) Interpolation by radial-basis functions is provided. This method works purely on vertex data and is a flexible choice for arbitrary mesh combinations with overlaps and gaps alike.

In the second phase of the ExaFSA project, we improved the performance of the data mapping schemes in various ways. All three interpolation algorithms contain a lookup-phase which searches for vertices or mesh elements near a given set of positions. As there is no guarantee regarding ordering of vertices, this resulted in $O(n \cdot m)$ lookup operations, $n, m \in \mathbb{N}$ being the size of the respective meshes. In the second phase, we introduced a tree-based data structure to facilitate efficient spatial queries. The implementation utilizes the library `Boost Geometry`[4] and uses an rtree in conjunction with the `r-star` insertion algorithm. The integration of the tree is designed to fit seamlessly into preCICE and avoids expensive copy operations for vertices and mesh elements of higher dimensionality. Consequently, the complexity of the lookup-phase was reduced to $O\left(\log_a n\right) \cdot m$ with a being a parameter of the tree, set to $\approx$5. The tree index is used by nearest-neighbor, nearest-projection, and RBF interpolation as well as other parts in preCICE and provides a tremendous speedup in the initialization phase of the simulation.

In the course of integrating the index, the RBF interpolation profited from a second performance improvement. In contrast to the nearest-neighbor and nearest-projection schemes it creates an explicit interpolation matrix. Setting values one by one results in a large number of small memory allocations with a relatively large per-call overhead. To remedy this, a preallocation pattern is computed with the help of the tree index. This results in a single memory allocation, speeding up the process of filling the matrix. A comparison of the accuracy and runtime of the latter two interpolation methods is provided in Sect. 5.

4.3 Communication

Smart and efficient communication is paramount in a partitioned multi-physics scenario. As preCICE is targeted at HPC systems, a central communication instance would constitute a bottleneck and has to be avoided. At the end of phase one, we implemented a distributed application architecture. The main objective in its design is not a classical speed-up (as it is for parallelism) but not to deteriorate the scalability of the solvers and rendering a central instance unnecessary. Still, a so-called *master* process exists, which has a special purpose mainly during the initialization phase.

At initialization time, each solver gives its local portion of the interface mesh to preCICE. By a process called re-partitioning, the mesh is transferred to the coupling partner and partitioned there, i.e., the coupling partner's processes select interface data portions that are relevant for their own calculations. The partitioning pattern is determined by the requirements of the selected mapping scheme. The outcome of this process is a sparse communication graph, where only links between participants exist that share a common portion of the interface. While this process was basically in place at the end of phase one, it was refined in several ways.

MPI connections are managed by means of a communicator which represents an n-to-m connection including an arbitrary number of participants. The first imple-

[4]www.boost.org.

mentation used only one communication partner per communicator, essentially creating only 1-to-1 connections. To establish the connections, every connected pair of ranks had to exchange a connection token generated by the accepting side. This exchange is performed using the network file system, as the only a-priori existing communication space common to both participants. However, network file systems tend to perform badly with many files written to a single directory. To reduce the load on the file system, a hash-based scheme was introduced as part of the optimizations in phase two. With that, writing of the files is distributed among several directories, as presented in [26]. This scheme features a uniform distribution of files over different directories and, thus, minimizes the files per directory.

However, this obviously resulted in a large number of communicators to be created. As a consequence, large runs hit system limits regarding the number of communicators. Therefore, a new MPI communication scheme was created as an alternative. It uses only one communicator for an all-to-all communication, resulting in significant performance improvements for the generation of the connections. This approach also solves the problem of the high number of connection tokens to be published, though only for MPI. As MPI is not always available or the implementation is lacking, the hash-based scheme of publishing connection tokens is still required for TCP based connections.

4.4 Load Balancing

In a partitioned coupled simulation solvers need to exchange boundary data at the beginning of each iteration, which implies a synchronization point. If computational cores are not distributed in an optimal way among solvers, one solver will have to wait for the other one to finish its time step. Thus, the load imbalance reduces the computational performance. In addition, in a one way coupling scenario, if the data receiving solver is much slower than the other one, the sending partner has to wait until the other one is ready to receive (in synchronized communication) or store the data in a buffer (in asynchronous communication). In the first phase, the distribution of cores over solvers was adjusted manually and only synchronized communication was implemented, resulting in idle times.

Regression Based Load Balancing We use the load balancing approach proposed in [37] to find the optimal core distribution among solvers: we first model the solver performance against the number of cores for each domain and then optimize the core distribution to minimize the waiting time. Since mathematical modeling of the solvers' performance can be very complicated, we use an empirical approach as proposed in [37], first introduced in [10], to find an appropriate model.

Assuming we have a given set of m data points, consisting of pairs (p, f_p) mapping the number of ranks p to the run-time f_p, we want to find a function $f(p)$ which predicts the run-time against p. Therefore, we use the Performance Model

Normal Form (PMNF) [10] as a basis for our prediction model:

$$f^i(p) = \sum_{k=1}^{n} c_k p^{i_k} \log_2^{j_k}(p), \tag{10}$$

where the superscript i denotes the respective solver, n is a a-priori chosen number of terms, $i_k, j_k \in \mathbb{N}_0$ and c_k is the coefficient for the kth regression term. The next step is to optimize the core distribution such that we achieve minimal overall run time which can be expressed by the following optimization problem:

$$\begin{aligned} &\underset{p_1,\dots,p_l}{\text{minimize}} \quad F(p_1,\dots,p_l) \quad \text{with } F(p_1,\dots p_l) = \max_i(f^i(p_i)) \\ &\text{subject to} \quad \sum_{i=1}^{l} p_i \leq P. \end{aligned}$$

This optimization problem is a nonlinear, possibly non-convex integer program. It can be solved by the use of branch and bound techniques. But, if we assume that the f^i are all monotonically decreasing, i.e., assigning more cores to a solver never increases the run-time, we can simplify the constraints to $P = \sum_{i=0}^{l} p_i$ and solve the problem by brute-forcing all possible choices for p_i. That is, we iterate over all possible combinations of core numbers and choose the pair that minimizes the total run-time. For more details, please refer to [37].

Asynchronous Communication and Buffering For our fluid-structure-acoustic scenario shown in Fig. 1, we perform an implicitly coupled simulation of the elastic structure interacting with the incompressible flow over a given discrete time step (marked simply as ‘Fluid’ in Fig. 2). This is followed by many small time steps for the acoustic wave propagation in the near-field, which are coupled in a loose, uni-directional way to the far-field acoustic solver (executing the same small time steps). To avoid waiting times of the far-field solver while we compute the fluid-structure interactions in the near-field, we would like to ‘stretch’ the far-field calculations such that they consume the same time as the sum of fluid-structure time steps and acoustic steps in the near-field (see Fig. 2). To achieve this, we introduced a fully asynchronous buffer layer, by which the sending participant was decoupled from the receiving participant, as shown in Fig. 2. Special challenges to tackle were the preservation of the correct ordering of messages, especially for TCP communication which does not implement such guarantees in the protocol.

4.5 Isolated Performance of preCICE

In this section, we show numerical results for preCICE only. This isolated approach is used to show the efficiency of the communication initialization. In addition,

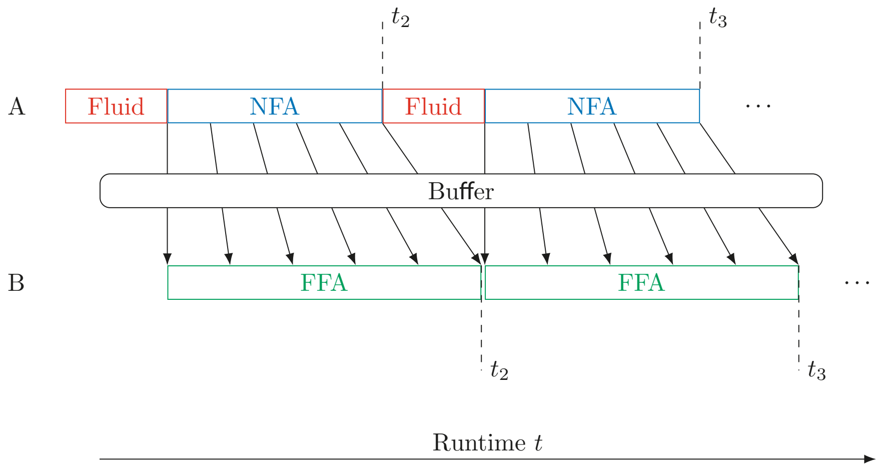


Fig. 2 Coupling scenario between participant A (performing a time step for the incompressible fluid (or fluid-structure interaction) followed by many time steps of the near-field acoustic simulation (NFA)) and participant B (performing the same small acoustic steps for the far-field (FFA) after receiving acoustic data from the near-field solver). Without buffering, inevitable idle times for participant B are created. NFA is linked to FFA through send operations. Therefore, the runtimes of NFA and FFA are matched through careful load-balancing. Shown here: A send buffer decouples NFA and FFA solver for send operations, prevents idle times, and allows for a more flexible processor assignment

we show stand-alone upscaling results. Other aspects are considered elsewhere: (a) the mapping accuracy is analyzed in Sect. 5, (b) the effectiveness of our load balancing approach as well as the buffering for uni-directional coupling are covered in Sect. 6. If not denoted otherwise, the following measurements are performed on the supercomputing systems SuperMUC[5] and HazelHen.[6]

Mapping Initialization: Preallocation and Matrix Filling As described previously, one of the key components of mapping initialization is the spatial tree which allows for performance improvements by accelerating the interpolation matrix construction. Figure 3 compares different approaches to matrix filling and preallocation: (a) no preallocation: using no preallocation at all, i.e., allocating each entry separately, (b) explicitly computed: calculate matrix sparsity pattern in a first mesh traversal, allocate entries afterwards, and finally fill the matrix in a second mesh traversal, (c) computed and saved: additionally cache mesh element/data point relations from the first mesh traversal and use them in the second traversal to fill the matrix with less computation, (d) spatial tree: use the spatial tree instead of brute-force pairwise comparisons to determine mesh components relevant for the

[5]28× Intel-Xeon-E5-2697 cores, 64 GB memory per node.

[6]24× Intel Xeon-E5-2680 cores, 128 GB memory per node.

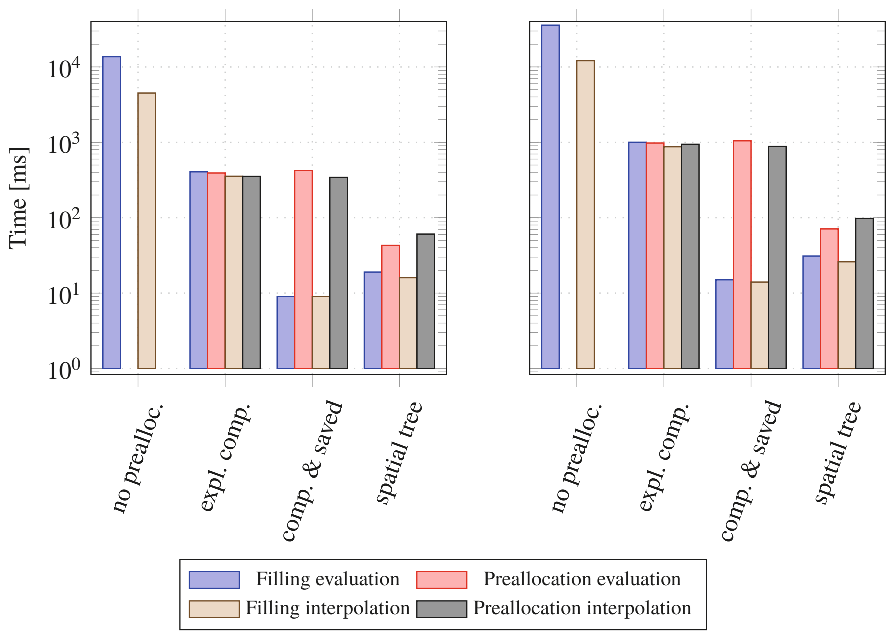


Fig. 3 Mapping initialization. Comparison of different preallocations methods for mesh sizes 6400 (left sub-figure) and 10,000 (right sub-figure) on two ranks per participant. The plot compares times spent in the stages of preallocation and filling of matrices for both the evaluation matrix and the interpolation matrix of an RBF mapping with localized Gaussian basis functions including 6 vertices of the mesh. The total time required is the sum of all bars of one measurement. Note the logarithmic scaling of the y-axis. The measurements were performed on one node of the `sgscl1` cluster, using 4× Intel Xeon E3-1585 CPUs

mapping. Each method can be considered as an enhancement of the previous one. As it becomes obvious from Fig. 3, the spatial tree was able to provide us a with an acceleration of more than two orders of magnitude.

Communication For communication and its initialization, we only present results for the new single-communicator MPI based solution. For TCP socket communication that still requires the exchange of many connection tokens by means of the file system, we only give a rough factor of 2.5 that we observed in terms of acceleration of communication initialization. Note that this factor can be potentially higher as the number of processes and, thus, connections grows larger, and that the hash-based approach removed the hard limit of ranks per participant inherent to the old approach.

In Figs. 4, 5 and 6, we compare performance results for establishing an MPI connection among different ranks using many-communicators for 1-to-1 connections with using a single communicator representing an n-to-m connection. In our academic setting, both Artificial Solver Testing Environment (ASTE) participants run on n cores. On SuperMUC, each rank connects to $0.4n$ ranks, on HazelHen, with

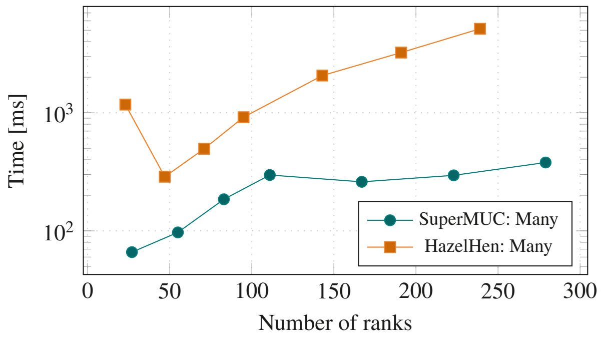


Fig. 4 Communication. Publishing of MPI connection information from participant A for the many-communicator approach. The timings of the new single-communicator approach are not shown, as they are almost negligible with a maximum of 2 ms

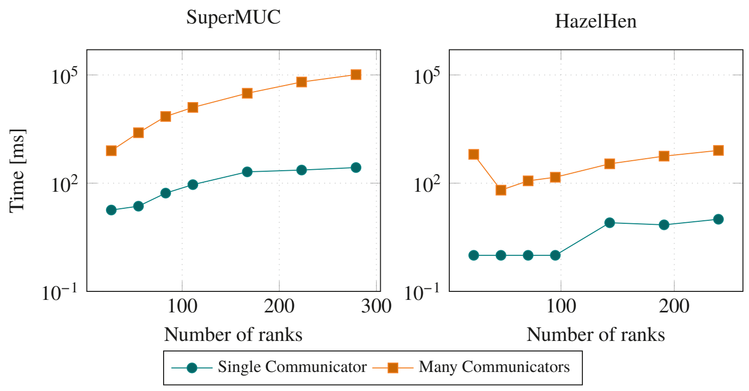


Fig. 5 Communication. Runtime for establishing the connection between the participants using `MPI_Comm_accept` and `MPI_Comm_connect`

a higher number of ranks per node, each rank connects to $0.3n$ ranks. The amount of data transferred between each connected pair of ranks is held constant with 1000 rounds of transfer of an array of 500, and 4000 double values from participant B to participant A. Each measurement is performed five times of which the fastest and the slowest runs are ignored and the remaining three are averaged. We present timings from rank zero, which is synchronized with all other ranks by a barrier, making the measurements from each rank identical. Note, that the measurements are not directly comparable between SuperMUC and HazelHen due to the different number of cores per node and that the test case is even more challenging than actual coupled simulations. In an actual simulation, the number of partner ranks per rank of a participant is constant with increasing number of cores on both sides.

Figure 4 shows the time to publish the connection token. The old approach requires to publish many tokens, which obviously becomes a performance bottleneck as the simulation setup moves to higher number of ranks. The new approach,

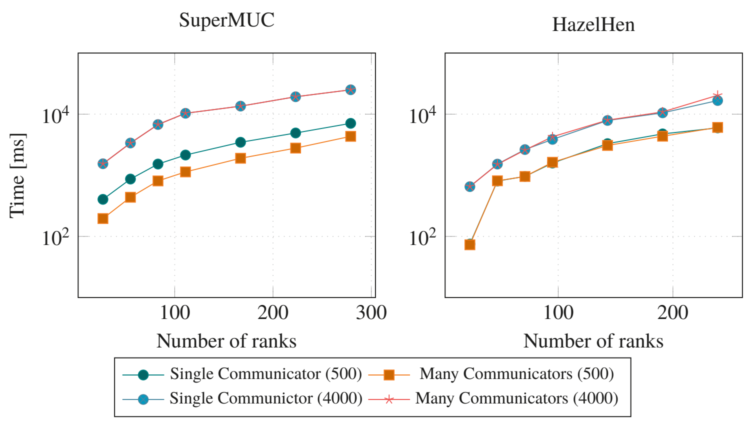


Fig. 6 Communication. Times for 1000 rounds of data transfer of a vector of 500 or 4000 doubles, respectively, from participant B to A. For the transfer, the synchronous MPI routines (`MPI_Send` and `MPI_Recv`) have been used

on the other hand, only publishes one token. It is omitted in the plot, as the times are negligible (<2 ms). In Fig. 5, the time for the actual creation of the communicator is presented. The total number of communication partners per communicator is smaller with the old many-communicator concept (as the communication topology is sparse). However, the creation of many 1-to-1 communicators is substantially slower than the creation of one all-to-all communicator for both HPC systems. Finally, in Fig. 6 the performance for an exchange of data sets of two different sizes is presented. The results for single- and many-communicator approaches are mostly on par with the notable exception of the SuperMUC system. There, the new approach suffers a small but systematic slow-down for small message sizes. We argue that this is a result of vendor specific settings of the MPI implementation.

Data Mapping As described above, we have further improved the mapping initialization, in particular by applying a tree-based approach to identify data dependencies induced by the mapping between grid points of the non-matching solver grids and to assemble the interpolation matrix for RBF mapping. Accordingly, we show both the reduction of the matrix assembly runtime (Fig. 3) and the scalability of the mapping, including setting up the interpolation system and the communication initialization.

These performance tests of preCICE are measured using a special testing application called ASTE.[7] This application behaves like a solver to preCICE but provides artificial data. It is used to quickly generate input data and decompose it for upscaling tests. ASTE generates uniform, rectangular, two-dimensional meshes on $[0, 1] \times [0, 1]$ embedded in three-dimensional space with the z-dimension always

[7] https://github.com/precice/aste.

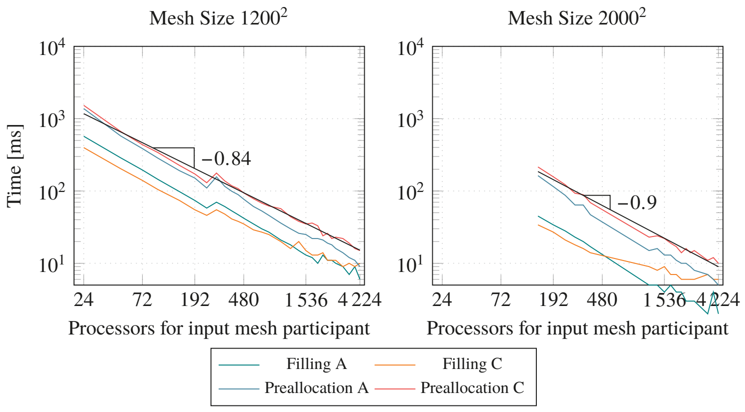


Fig. 7 Data mapping initialization. Strong scaling of the initialization of the interpolation matrix C and the evaluation matrix A for RBF interpolation on mesh sizes 1200^2 and 2000^2 with Gaussian $(m = 6)$ basis functions on up to 4224 processors on the HazelHen HPC system

set to zero. The mesh is then decomposed using a *uniform* approach, thus producing partitions of same size as far as possible. Since we mainly look at the mapping part which is only executed as one of the participants, we limit the upscaling to this participant. The other participant always uses one node (28 resp. 24 processors). The mesh size is kept constant, i.e., we perform a strong scaling. The upscaling of an RBF mapping with Gaussian basis functions is shown in Fig. 7.

5 Black-Box Coupling Versus White-Box Coupling with APESMate

In the above section, we have evaluated the performance of the black-box coupling tool preCICE. In this section, we introduce an alternative approach that allows to couple different solvers provided within the framework APES [31]. Black-box data mapping in preCICE only requires point values (nearest neighbor and RBF mapping), and in some cases (nearest projection) connectivity information on the coupling interface. The white-box coupling approach of APESmate [25] has knowledge about the numerical schemes within the domain, since it is integrated in the APES suite, and has access to the common data-structure TreELM [22]. APESmate can directly evaluate the high order polynomials of the underlying Discontinuous Galerkin scheme. Thus, the mapping in preCICE is more generally applicable, while the approach in APESmate is more efficient in the context of high order scheme. Furthermore, APESmate allows the coupling of all solvers of the APES framework, both in terms of surface and in terms of volume coupling. The communication between solvers can be done in a straightforward way as all coupling participants can be compiled as modules into one single application. Each subdomain defines its own MPI sub-communicator, a global communicator is used for the communication between the subdomains. During the initialization process, coupling requests are locally gathered from all subdomains and exchanged in a round-robin fashion. As all solvers in APES are based on an octree data-structure and a space-filling curve for partitioning, it is rather easy to get information about the location of each coupling point on the involved MPI ranks. In the following, we compare both accuracy and runtime of the two coupling approaches for a simple academic test case that allows to control the 'difficulty' of the mapping by adjusting order and resolution of the two participants.

Test Case Setup We consider the spreading of a Gaussian pressure pulse over a cubic domain of size $5 \times 5 \times 5$ unit length, with an ambient pressure of 100,000 Pa and a density of 1.0 kg/m^3. The velocity vector is set to 0.0 for all spatial directions. To generate a reference solution, this test case is computed monolithically using the inviscid Euler equations.

For the coupled simulations, we decompose the monolithic test case domain into an inner and an outer domain. The resolution and the discretization order of the inner domain are kept unchanged. In the outer domain, we choose the resolution and the order such that the error is balanced with that of the inner domain. See [15] for the respective convergence study. To be able to determine the mapping error at the coupling interface between inner and outer domain, we choose the time horizon such that the pressure pulse reaches the outer domain, but is still away from the outer boundaries to avoid any influences from the outer boundaries. The test case is chosen in a way, that the differences between the meshes at the coupling interface increase, thus increasing the difficulty to maintain the overall accuracy in a black-box coupling approach. Table 1 provides an overview of all combinations of resolution and order in the outer domain used for our numerical experiments, where the total number of elements per subdomain is given as nElements, the number of coupling points with nCoupling points and the scheme order by nScheme order, respectively. For time discretization, we consider the explicit two stage Runge-Kutta scheme with a time step size of 10^{-6} for all simulations.

Table 1 White-box coupling test scenario with Gaussian pressure pulse combinations of orders and resolution used for the evaluation of the mapping methods

	Test case a		Test case b		Test case c		Test case d	
	Inner	Outer	Inner	Outer	Inner	Outer	Inner	Outer
nElements	32,768	124,000	32,768	7936	32,768	992	32,768	124
nCoupling points	55,296	9600	55,296	3456	55,296	1536	55,296	1176
nScheme order	3	4	3	6	3	8	3	14

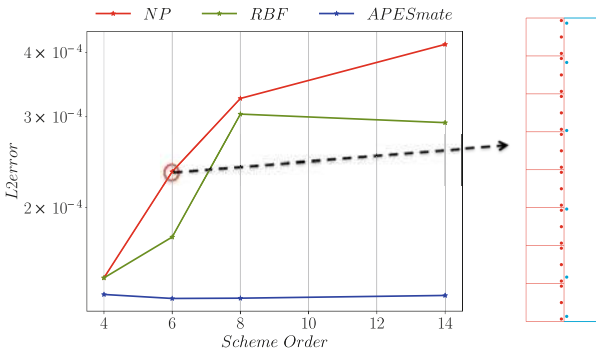


Fig. 8 Mapping accuracy black-box (APESmate) versus white-box (preCICE) approach. Comparison of the L2 error (with the analytical solution as reference) for the Gaussian pressure pulse test case variants and exemplary illustration of the coupling point distribution when using DG for test case (right figure). We compare the black-box data mapping methods Radial-Basis Functions (RBF) interpolation and Nearest Projection (NP) with the direct white-box evaluation of APESmate. The RBF mapping uses local Gaussian basis functions covering three mesh points in every direction

Mapping Accuracy In terms of mapping accuracy, it is expected, that the APESmate coupling is order-preserving, and by that not (much) affected by the increasing differences between the non-matching grids at the coupling interface, while preCICE should show an increasing accuracy drop when the points become less and less matching. This is the case for increasing order of the discretization in the outer domain. Figure 8 illustrates first results. As can be clearly seen, the white-box coupling approach APESmate provides outstanding results by maintaining the overall accuracy of the monolithic solution for all different variations of the coupled simulations, independent of the degree to which the grids are non-matching (increasing with increasing order used in the outer domain). For the interpolation methods provided by preCICE, the error increases considerably with increasing differences between the grids at the interface. As the error of the interpolation methods depends on the distances of the points (see Fig. 8), the error is dominated by the large distance of the integration points in the middle of the surface of an octree grid cell in the High Order Discontinuous Galerkin discretization.

Accuracy Improvement by Regular Subsampling We can decrease the L2 error of NP and RBF and improve the solution of the coupled simulation by providing values at equidistant points on the Ateles side as interpolation support points. The number of equidistant points is equal to the number of coupling points, hence as high as the scheme order. With this new implementation, the error shown in Fig. 9a decreases considerably compared to the results in Fig. 8. We achieve an acceptable accuracy for all discretization order combinations. However, the regular subsampling of values in the Ateles solver increases the overall computational time substantially as can be seen in Fig. 9b.

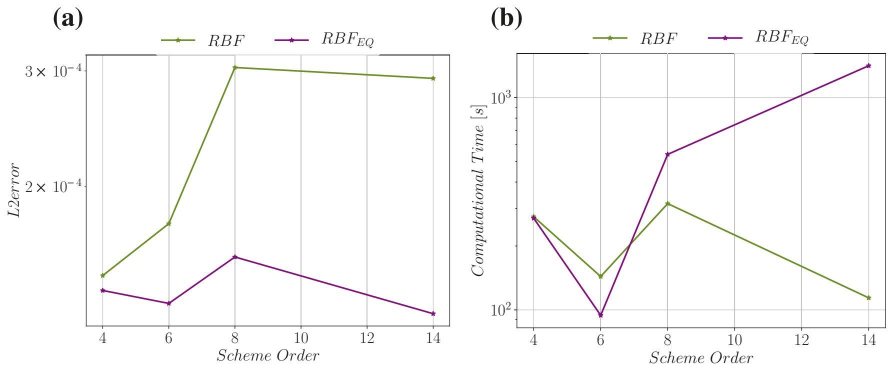


Fig. 9 Mapping accuracy and runtime of black-box with equidistant subsampling versus white-box approach. L2 error behavior (**a**) and computational time (**b**) for the RBF interpolation, when using equidistant (RBF_{EQ}) and non-equidistant (RBF) point distributions for data mapping

To improve the NP interpolation, it turned out that in addition to providing equidistant points, oversampling was required to increase the accuracy. Our investigation showed, that an oversampling factor of 3 is needed to achieve almost the same accuracy as APESmate. In spite of the additional cost of many newly generated support points, the runtime does not increase as much as for RBF, since for the RBF a linear equation system has to be computed, while for NP a simple projection needs to be done.

Summary and Runtime Comparison Figure 10 shows a summary of all tested methods for the interpolation/evaluation before and after improvements. The integrated coupling approach APESmate provides not just very accurate results, but also low runtimes. At this point, we want to recall that this is as expected—the white-box approach makes use of all internal knowledge, which gives it advantages in

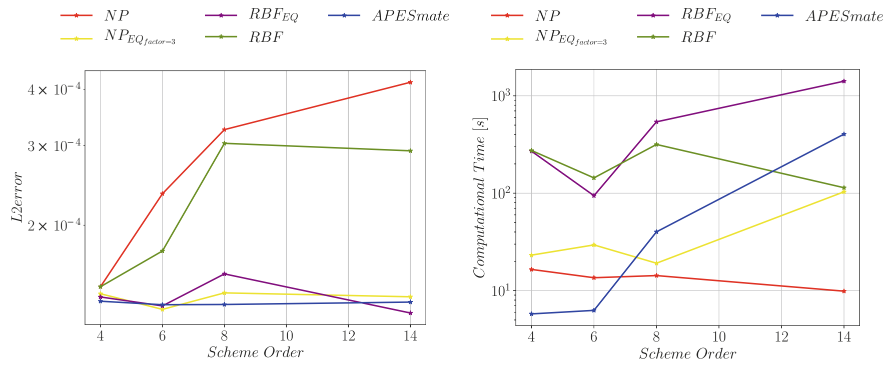


Fig. 10 Mapping accuracy and runtime summary of black-box versus white-box approach. L2 error and computational time for all methods

terms of accuracy and efficiency. On the other hand, this internal knowledge binds it to the solvers available in the framework, while preCICE can be applied to almost all available solvers. Further details regarding this investigation can be found in [15, 16].

6 Results

This section presents a more realistic test case, the turbulent flow over a fence, to assess the overall performance of our approach. Analyses for accuracy and specific isolated aspects are integrated in the sections above.

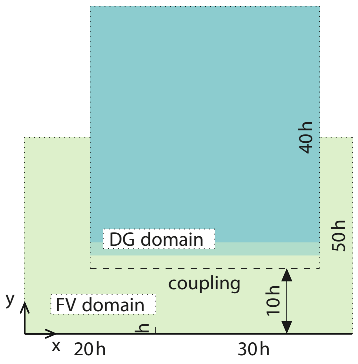
6.1 Flow over a Fence Test Case Setup

As a test case to assess the overall scalability, we simulate the turbulent flow over a (flexible) fence and the induced acoustic far-field as already shown schematically in Fig. 1. The FSI functionality of FASTEST has been demonstrated earlier many times, e.g. in [34]. Thus we focus on the acoustic coupling.

As boundary conditions, we use a no-slip wall at the bottom and the fence surface, an inflow on the left with u_{bulk}, outflow convective boundary conditions on the right, periodic boundary conditions in y-directions, and slip conditions at the upper boundary for the near-field flow. For the acoustic perturbation, we apply reflection conditions at the bottom and the fence surface, zero-gradient condition at all other boundaries. The acoustic far-field solver uses Dirichlet boundary conditions at its lower boundary (see also Eq. (9)). Therefore, the upper near-field boundary is not the coupling interface, but we instead overlap near-field and far-field as shown in Fig. 11. Figures 12 and 13 show a snapshot of the near-field flow and the near-field and far-field acoustic pressure, respectively.

6.2 Fluid-Acoustics Coupling with FASTEST and Ateles

To demonstrate the computational performance of our framework using FASTEST for the flow simulation in the near-field, the high-order DG solver Ateles for the far-field acoustic wave propagation, and preCICE for coupling, we show weak scalability measurements for the interaction between near-field flow simulation and far-field acoustics. We keep both the mesh and the number of MPI ranks in the near-field flow simulation fixed. In the far-field computed with Ateles, we refine the mesh to better capture the acoustic wave propagation. We use a multi-level mesh with a fine mesh at the coupling interface to allow a smooth solution at the coupling interface between the near-field and the far-field. We refine the far-field mesh in



Fence height	$h = 0.05m$
Inflow velocity	$u_{\text{bulk}} = 50m/s$
Fluid material	air
Fluid density	$\rho = 1.184kg/m^3$
Dynamic viscosity	$\mu = 1.586 \cdot 10^{-5}kg/(ms)$
Speed of sound	$c = 343.3m/s$
Young's modulus	$E = 16MPa$
Poisson ratio	$\nu = 0.48$
Structure density	$\rho_S = 1360kg/m^3$

Fig. 11 Flow over a fence test case. Schematic view of the computational domain (left) and applied parameters (right). Colors indicate spatial discretization order in the various regions: The FV domain is completely second order, while the DG domain has second order at the coupling interface for reduced coupling interpolation errors, and subsequently increases the order in various layers for the far-field transport

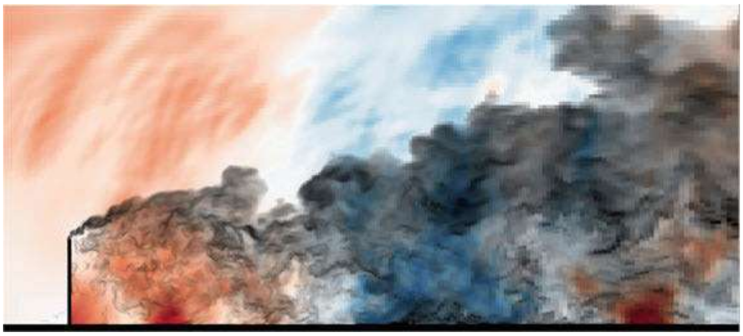


Fig. 12 Flow over a fence test case with FASTEST. Snapshot of the flow in the recirculation area behind the fence. Red/blue indicate acoustic pressure, grey shades show the modelled turbulent kinetic energy (for a $\zeta - \text{f}$ DES model)

two main steps: in the first step, we only refine the mesh at the coupling interface. In the next step, we first refine the whole mesh, and again the mesh at the coupling interface in the third and fourth step. Due to the refinement at the coupling interface, the number of Ateles ranks participating in the interface increases such that this study also shows that the preCICE communication does not deteriorate scalability. Table 2 gives an overview of the configurations used for the weak scaling study.

To find the optimal core distribution for all setups, the load balancing approach proposed in Sect. 4 is used. This analysis shows that for the smallest mesh resolution with 24,864 elements in the far-field, the optimal core distribution is 424 cores for the near-field domain and 196 cores for the far-field. For all other setups, we assume perfect scalability, i.e., we choose the number of cores proportional to the number of degrees of freedom in the weak scaling study and increase the number of cores

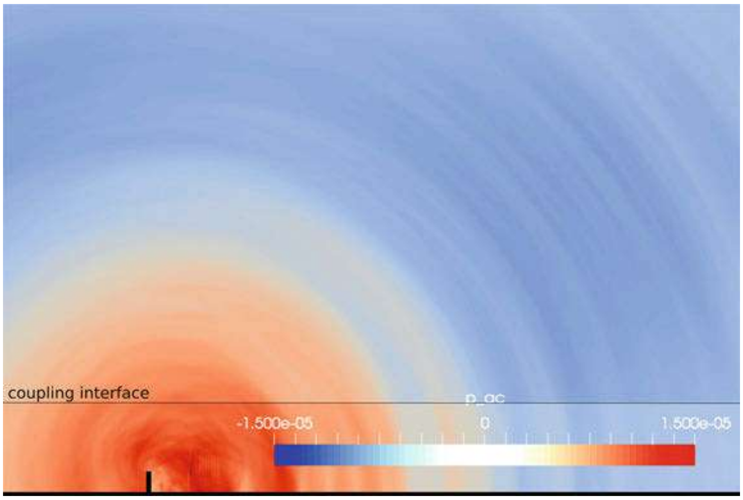


Fig. 13 Flow over a fence test case. Snapshot of the acoustic pressure in a coupled simplified setup

Table 2 Flow over a fence test case

Cores in FASTEST/Ateles	Ateles degrees of freedom	Ateles elements
424/196	16,116,480	24,864
424/756	62,535,840	89,376
424/1428	116,524,800	177,408
424/3136	254,150,400	607,488
424/15,372	1,245,054,720	3,704,064

Scalability study for the interaction between the near-field flow simulation and the far-field acoustics: Summary of mesh details and core numbers for weak scaling. In the FASTEST simulation of the near-field flow simulation, we use 52,822,016 elements. In the far-field, Ateles uses discretization order 9

simultaneously by a factor of two in both fields for strong scaling. The scalability measurements are shown in Fig. 14. The results show that the framework scales almost perfectly up to 6528 cores.

6.3 *Fluid-Acoustics Coupling with Only Ateles*

In Sect. 5 we investigated the suitability of different interpolation methods for our simulations. In this section, we present a strong scaling study for an Ateles-Ateles coupled simulation of the flow over a fence test case. The fence is modelled in

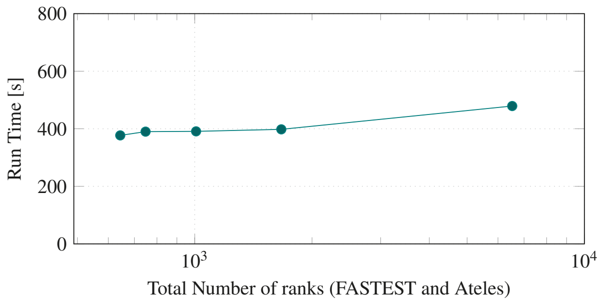


Fig. 14 Flow over a fence test case. Weak scalability measurement of the fluid-acoustic interaction simulation for the fence test case

Ateles using the newly implemented immersed boundary method, enabling high-order representation of complex geometries in Ateles [1]. We solve the compressible Navier-Stokes equations in the flow domain with a scheme order of 4 and a four step mixed implicit-explicit Runge-Kutta time stepping scheme, with a time step size of 10^{-7}. The total number of elements in the flow domain is 192,000. For the far-field, we use the same setup as for the FASTEST-Ateles coupling.

The linearized Euler equations in the far-field can be solved in a DG setting in the modal formulation, which makes the solver very cheap even for very high order. In the near-field domain, the non-linear Navier-Stokes equations are solved with a more expensive hybrid nodal-modal approach. Due to this, and the different spatial discretizations and scheme orders, both domains have different computational load, which requires load balancing. We use static load balancing, since neither the mesh nor the scheme order vary during runtime. As both solvers are instances of Ateles, we apply the SpartA algorithm [21], which allows re-partitioning of the workload according to weights per elements, which are computed during runtime. Those weights are then used to re-distribute the elements according to the workload among available processes (see [17] for more details). The total number of processes used for this test case are 14,336 processes, which is equal to one island on the system. As mentioned previously, the total workload per subdomain does not change, therefore we start our measurements by providing the lower subdomain 100 processes and the upper subdomain 12 processes, which is equal to 4 nodes on the system. This number per subdomain is then doubled for each run, the ratio is kept the same.

Figure 15 shows the strong scaling measurements for both coupling approaches (APESmate and preCICE) executed on the SuperMUC Phase2 system. As can be clearly seen, both coupling setups Ateles-APESmate-Ateles and Ateles-preCICE-Ateles scale almost ideally, however with a lower absolute runtime for the APES-mate coupling as expected.

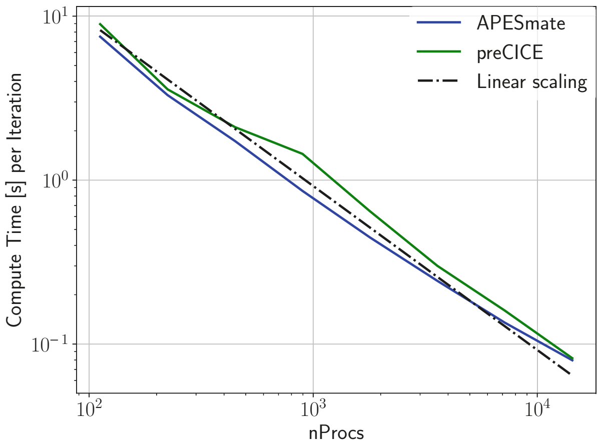


Fig. 15 Flow over a fence test case. Strong scaling measurement for Ateles-Ateles coupling using both APESmate and preCICE—and the ideal linear scaling as reference

7 Summary and Conclusion

We have presented a partitioned simulation environment for the massively parallel simulation of fluid-structure-acoustic interactions. Our setup uses the flow and acoustic solvers in the finite volume software FASTEST, the acoustic solvers in the discontinuous Galerkin framework Ateles as well as the black-box fully parallel coupling library preCICE. In particular, we could show that with a careful design of the coupling tool as well as of solver details, we can achieve a bottleneck-free numerically and technically highly scalable solution. It turned out that efficient initialization of point-to-point communication relations and mapping matrices between the involved participants, sophisticated inter-code load balancing and asynchronous communication using message buffering are crucial for large-scale scenarios. With these improvements, we advanced the limits of scalability of partitioned multi-physics simulations from less than a hundred cores to more than 10,000 cores. Beyond that, we reach a problem size that is not required by the given problem as well as scalability limits of the solvers. The coupling itself is not the limiting factor for the given problem size and degree of parallelism. To be able to use also vector architectures in an efficient sustainable way, we adapted our solvers with a highly effective code transformation approach.

Acknowledgments We thankfully acknowledge funding from SPPEXA, the German Science Foundation Priority Program 1648—Software for Exascale Computing and the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 754462, and compute time granted on the LRZ SuperMUC system and the HLRS HazelHen system.

References

1. Anand, N., Pour, N.E., Klimach, H., Roller, S.: Utilization of the brinkman penalization to represent geometries in a high-order discontinuous Galerkin scheme on octree meshes. Symmetry **11**(9), 1126 (2019). https://doi.org/10.3390/sym11091126
2. Banks, J.W., Henshaw, W.D., Schwendeman, D.W.: An analysis of a new stable partitioned algorithm for FSI problems. Part I: incompressible flow and elastic solids. J. Comput. Phys. **269**, 108–137 (2014)
3. Bazilevs, Y., Takizawa, K., Tezduyar, T.E.: Computational Fluid-Structure Interaction: Methods and Applications. Wiley, Hoboken (2012)
4. Blom, D., Ertl, T., Fernandes, O., Frey, S., Klimach, H., Krupp, V., Mehl, M., Roller, S., Sternel, D.C., Uekermann, B., Winter, T., van Zuijlen, A.: Partitioned fluid-structure-acoustics interaction on distributed data – numerical results and visualization. In: Bungartz, H.J., Neumann, P., Nagel, E.W. (eds.) Software for Exa-scale Computing – SPPEXA 2013–2015. Springer, Berlin (2016)
5. Boer, A.D., van Zuijlen, A., Bijl, H.: Comparison of conservative and consistent approaches for the coupling of non-matching meshes. Comput. Methods Appl. Mech. Eng. **197**(49), 4284–4297 (2008)
6. Bungartz, H., Mehl, M., Schäfer, M.: Fluid Structure Interaction II: Modelling, Simulation, Optimization. Lecture Notes in Computational Science and Engineering. Springer, Berlin (2010)
7. Bungartz, H.J., Lindner, F., Mehl, M., Uekermann, B.: A plug-and-play coupling approach for parallel multi-field simulations. Comput. Mech. **55**(6), 1119–1129 (2015)
8. Bungartz, H.J., Lindner, F., Gatzhammer, B., Mehl, M., Scheufele, K., Shukaev, A., Uekermann, B.: preCICE – a fully parallel library for multi-physics surface coupling. Comput. Fluids **141**, 250–258 (2016). Advances in Fluid-Structure Interaction
9. Bungartz, H.J., Lindner, F., Mehl, M., Scheufele, K., Shukaev, A., Uekermann, B.: Partitioned fluid-structure-acoustics interaction on distributed data – coupling via preCICE. In: Bungartz, H.J., Neumann, P., Nagel, E.W. (eds.) Software for Exa-scale Computing – SPPEXA 2013–2015. Springer, Berlin (2016)
10. Calotoiu, A., Beckinsale, D., Earl, C.W., Hoefler, T., Karlin, I., Schulz, M., Wolf, F.: Fast multi-parameter performance modeling. In: 2016 IEEE International Conference on Cluster Computing (CLUSTER), pp. 172–181 (2016). https://doi.org/10.1109/CLUSTER.2016.57
11. Caretto, L.S., Gosman, A.D., Patankar, S.V., Spalding, D.B.: Two calculation procedures for steady, three-dimensional flows with recirculation. In: Proceedings of the Third International Conference on Numerical Methods in Fluid Dynamics. Springer, Paris (1972)
12. Crosetto, P., Deparis, S., Fourestey, G., Quarteroni, A.: Parallel algorithms for fluid-structure interaction problems in haemodynamics. SIAM J. Sci. Comput. **33**(4), 1598–1622 (2011)
13. Degroote, J., Bathe, K.J., Vierendeels, J.: Performance of a new partitioned procedure versus a monolithic procedure in fluid-structure interaction. Comput. Struct. **87**(11–12), 793–801 (2009)
14. Dhondt, G.: The Finite Element Method for Three-Dimensional Thermomechanical Applications. Wiley, Chichester (2004)
15. Ebrahimi Pour, N., Roller, S.: Error investigation for coupled simulations using discontinuous galerkin method for discretisation. In: Proceedings of ECCM VI/ECFD VII, Glasgow, June 2018
16. Ebrahimi Pour, N., Krupp, V., Klimach, H., Roller, S.: Coupled simulation with two coupling approaches on parallel systems. In: Resch, M.M., Bez, W., Focht, E., Gienger, M., Kobayashi, H. (eds.) Sustained Simulation Performance 2017, pp. 151–164. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-66896-3_10
17. Ebrahimi Pour, N., Krupp, V., Roller, S.: Load balancing for coupled simulations. In: Resch, M.M., Bez, W., Focht, E., Gienger, M., Kobayashi, H. (eds.) Sustained Simulation Performance 2019. Springer International Publishing, Cham (2019)

18. Fachgebiet für Numerische Berechnungsverfahren im Maschinenbau: FASTEST Manual (2005)
19. Gee, M.W., Küttler, U., Wall, W.A.: Truly monolithic algebraic multigrid for fluid–structure interaction. Int. J. Numer. Methods Eng. **85**(8), 987–1016 (2011)
20. Haelterman, R., Degroote, J., van Heule, D., Vierendeels, J.: The quasi-Newton least squares method: a new and fast secant method analyzed for linear systems. SIAM J. Numer. Anal. **47**(3), 2347–2368 (2009)
21. Harlacher, D.F., Klimach, H., Roller, S., Siebert, C., Wolf, F.: Dynamic load balancing for unstructured meshes on space-filling curves. In: 2012 IEEE 26th International on Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), pp. 1661–1669 (2012)
22. Klimach, H.G., Hasert, M., Zudrop, J., Roller, S.P.: Distributed octree mesh infrastructure for flow simulations. In: Eberhardsteiner, J. (ed.) ECCOMAS 2012 - European Congress on Computational Methods in Applied Sciences and Engineering, e-Book Full Papers (2012)
23. Kong, F., Cai, X.C.: Scalability study of an implicit solver for coupled fluid-structure interaction problems on unstructured meshes in 3D. Int. J. High Perform. Comput. Appl. **32**, 207–219 (2016)
24. Kornhaas, M., Schäfer, M., Sternel, D.: Efficient numerical simulation of aeroacoustics for low Mach number flows interacting with structures. Comput. Mech. **55**, 1143–1154 (2015). https://doi.org/10.1007/s00466-014-1114-1
25. Krupp, V., Masilamani, K., Klimach, H., Roller, S.: Efficient coupling of fluid and acoustic interaction on massive parallel systems. In: Sustained Simulation Performance 2016, pp. 61–81 (2016)
26. Lindner, F.: Data transfer in partitioned multi-physics simulations: interpolation & communication. PhD thesis, University of Stuttgart (2019)
27. Lindner, F., Mehl, M., Uekermann, B.: Radial basis function interpolation for black-box multiphysics simulations. In: Papadrakakis, M., Schrefler, B., Onate, E. (eds.) VII International Conference on Computational Methods for Coupled Problems in Science and Engineering, pp. 1–12 (2017)
28. Link, G., Kaltenbacher, M., Breuer, M., Döllinger, M.: A 2D finite-element scheme for fluid-solid-acoustic interactions and its application to human phonation. Comput. Methods Appl. Mech. Eng. **198**(41–44), 3321–3334 (2009)
29. Mehl, M., Uekermann, B., Bijl, H., Blom, D., Gatzhammer, B., van Zuijlen, A.: Parallel coupling numerics for partitioned fluid-structure interaction simulations. Comput. Math. Appl. **71**(4), 869–891 (2016)
30. Reimann, T.: Numerische simulation von fluid-struktur-interaktion in turbulenten strömungen. Ph.D. thesis, Technische Universität Darmstadt (2013)
31. Roller, S., Bernsdorf, J., Klimach, H., Hasert, M., Harlacher, D., Cakircali, M., Zimny, S., Masilamani, K., Didinger, L., Zudrop, J.: An adaptable simulation framework based on a linearized octree. In: Resch, M., Wang, X., Bez, W., Focht, E., Kobayashi, H., Roller, S. (eds.) High Performance Computing on Vector Systems 2011, pp. 93–105. Springer, Berlin (2012)
32. Ross, M.R., Felippa, C.A., Park, K., Sprague, M.a.: Treatment of acoustic fluid-structure interaction by localized Lagrange multipliers: formulation. Comput. Methods Appl. Mech. Eng. **197**(33–40), 3057–3079 (2008)
33. Schäfer, F., Müller, S., Uffinger, T., Becker, S., Grabinger, J., Kaltenbacher, M.: Fluid-structure-acoustic interaction of the flow past a thin flexible structure. AIAA J. **48**(4), 738–748 (2010)
34. Schäfer, M., Sternel, D.C., Becker, G., Pironkov, P.: Efficient numerical simulation and optimization of fluid-structure interaction. In: Bungartz, H.J., Mehl, M., Schäfer, M. (eds.) Fluid Structure Interaction II. Modelling, Simulation, Optimization. Lecture Notes in Computational Science and Engineering, vol. 73, pp. 131–158. Springer, Berlin (2010)
35. Scheufele, K., Mehl, M.: Robust multisecant quasi-Newton variants for parallel fluid-structure simulations – and other multiphysics applications. SIAM J. Sci. Comput. **39**(5), 404–433 (2017)
36. Stone, H.L.: Iterative solution of implicit approximations of multidimensional partial differential equations. SIAM J. Numer. Anal. **5**(3), 530–558 (1968)

37. Totounferoush, A., Ebrahimi Pour, N., Schroder, J., Roller, S., Mehl, M.: A new load balancing approach for coupled multi-physics simulations (2019). https://doi.org/10.1109/IPDPSW.2019.00115
38. Turek, S., Hron, J., Madlik, M., Razzaq, M., Wobker, H., Acker, J.: Numerical simulation and benchmarking of a monolithic multigrid solver for fluid-structure interaction problems with application to hemodynamics. In: Bungartz, H.J., Mehl, M., Schäfer, M. (eds.) Fluid Structure Interaction II: Modelling, Simulation, Optimization, p. 432. Springer, Berlin (2010)
39. Uekermann, B.: Partitioned fluid-structure interaction on massively parallel systems. Ph.D. thesis, Department of Informatics, Technical University of Munich (2019)
40. Uekermann, B., Bungartz, H.J., Cheung Yau, L., Chourdakis, G., Rusch, A.: Official preCICE adapters for standard open-source solvers. In: 7th GACM Colloquium on Computational Mechanics, pp. 1–4 (2017)
41. Vierendeels, J., Lanoye, L., Degroote, J., Verdonck, P.: Implicit coupling of partitioned fluid-structure interaction problems with reduced order models. Comput. Struct. **85**(11–14), 970–976 (2007)
42. Zudrop, J., Klimach, H., Hasert, M., Masilamani, K., Roller, S.: A fully distributed CFD framework for massively parallel systems. In: Cray User Group Conference. Cray User Group, Stuttgart (2012)