Massively Parallel Multigrid with Direct Coarse Grid Solvers

M. Huber, N. Kohl, P. Leleux, U. Rüde, D. Thönnes, B. Wohlmuth

published in

NIC Symposium 2020
M. Müller, K. Binder, A. Trautmann (Editors)

Forschungszentrum Jülich GmbH,
John von Neumann Institute for Computing (NIC),
Schriften des Forschungszentrums Jülich, NIC Series, Vol. 50,
ISBN 978-3-95806-443-0, pp. 335.
http://hdl.handle.net/2128/24435

Massively Parallel Multigrid with Direct Coarse Grid Solvers

Markus Huber[1], **Nils Kohl**[2], **Philippe Leleux**[3], **Ulrich Rüde**[2,3], **Dominik Thönnes**[2], **and Barbara Wohlmuth**[1]

[1] Lehrstuhl für Numerische Mathematik, Technische Universität München, Garching, Germany
E-mail: {wohlmuth, huber}@ma.tum.de

[2] Lehrstuhl für Systemsimulation, Friedrich-Alexander Universität, Erlangen, Germany
E-mail: {ulrich.ruede, dominik.thoennes, nils.kohl}@fau.de

[3] Parallel Algorithms Team, CERFACS, Toulouse, France
E-mail: leleux@cerfacs.fr

Multigrid methods play an important role in the numerical approximation of partial differential equations. As long as only a moderate number of processors is used, many alternatives can be used as solver for the coarsest grid. However, when the number of processors increases, then standard coarsening will stop while the problem is still large and the communication overhead for solving the corresponding coarsest grid problem may dominate. In this case, the coarsest grid must be agglomerated to only a subset of the processors. This article studies the use of sparse direct methods for solving the coarsest grid problem as it arises in a multigrid hierarchy. We use as test case a Stokes-type model and solve algebraic saddle point systems with up to $\mathcal{O}(10^{11})$ degrees of freedom on a current peta-scale supercomputer. We compare the sparse direct solver with a preconditioned minimal residual iteration and show that the sparse direct method can exhibit better parallel efficiency.

1 Introduction

The numerical approximation of partial differential equations as obtained from models in science or engineering results in large sparse systems of algebraic equations. The solution of such systems requires techniques that can be efficiently executed on high-performance computing systems. In many cases, multigrid (MG) methods are asymptotically optimal and fast in the sense that their complexity grows only linearly with the number of unknowns and the constant of proportionality is only moderate.

However, the scaling of MG methods on modern supercomputers presents challenges and requires well-designed software structures and a performance aware implementation. In MG solvers it can be problematic that the parallel efficiency deteriorates when coarser grid levels are processed. When proceeding to successively coarser grids both computation and communication decrease, however, computation decreases faster than communication. As a consequence the communication overhead on coarser grids is larger despite the fact that the coarser grids demand progressively less work. To alleviate this trend, better load-balancing and a redistribution of the grids onto a subset of the processes, may become necessary to improve the performance.[1] The process of collecting the grid data and the associated linear system to a smaller number of processes is called agglomeration.

In this article, we apply heuristic strategies in combination with the hierarchical hybrids grids (HHG)[2] framework that realises a hybrid meshing approach. The scalability of the geometric MG solver (GMG) in the HHG framework was analysed and shown capable

to solve problems with more than 10^{13} degrees of freedom (DOF),[3] *i. e.* a problem with a solution vector that is 80 TByte large. It was also shown that the multigrid method delivers an excellent fine grid performance but that the used coarse grid Krylov type solver inhibits the overall parallel efficiency for very large scale applications. In this article we will present agglomeration strategies in combination with a direct coarse grid solver to compensate for this shortcoming.

The rest of this article is structured as follows: In Sec. 2, we introduce the model, the finite element discretisation and the general solver setup. Then, Sec. 3.1 details the geophysical setting and Sec. 3.2 analyses the deteriorating scalability of the standard coarse level solver of the HHG framework that is based on Krylov techniques. In Sec. 4.1, we describe the conversion routines from he HHG data-structures to standard sparse matrix data formats so that external solver frameworks can be employed. In Sec. 4.2, we introduce a master-slave agglomeration technique. The MUMPS framework is presented in Sec. 4.3. Sec. 5.1 considers MUMPS as standalone solver in several scaling experiments and Sec. 5.2 analyses the weak scaling behaviour of the combined method.

2 Model Problem, Discretisation and Solver

Let $\Omega \subset \mathbb{R}^3$ be an open and bounded domain. We consider the Stokes-type problem

$$\begin{aligned} -\operatorname{div}\left(\frac{\nu}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^\top)\right) + \nabla p &= \mathbf{f} \quad \text{in } \Omega \\ \operatorname{div}(\mathbf{u}) &= 0 \quad \text{in } \Omega \\ \mathbf{u} &= \mathbf{g} \quad \text{on } \partial\Omega \end{aligned} \tag{1}$$

where $\mathbf{u}$ denotes the velocity, p stands for the pressure, and $\mathbf{f}$ is a given forcing term. For simplicity of notation we restrict ourselves to Dirichlet boundary conditions $\mathbf{g}$ and the viscosity ν is assumed to be uniformly positive definite and scalar but has possibly large jumps. Problems of this structure can be found in many applications.

We use an initial tetrahedral mesh $\mathcal{T}_0$ and construct by uniform mesh refinement a hierarchy of meshes $\mathcal{T}_0 = \{\mathcal{T}_\ell, \ell = 0, \ldots, L\}$, $L > 0$ for the MG scheme, linked with linear interpolation for each component as prolongators and their adjoint as restrictions. For the moment we assume that Ω is resolved by the initial triangulation but generalisations are possible.[4] For the discretisation, we apply the lowest equal-order finite elements method stabilised with the pressure stabilisation Petrov-Galerkin (PSPG) technique.[5] Using (component-wise) nodal basis functions for velocity and pressure, we obtain a hierarchy of 2×2-block structured algebraic systems.

In a previous work,[3] we have compared different types of solvers and have found that a monolithic multigrid method with Uzawa type smoother combined for velocity and pressure achieves the best in time-to-solution and has the lowest memory requirements. In the following studies, we will use a geometric multigrid solver in form of a mildly variable V–cycle, *i. e.* we add for each coarser level 2 additional smoothing steps. This is denoted by V_{var}–cycle. The performance of the Uzawa smoother has also been theoretically and numerically investigated.[6]

3 Hierarchical Hybrid Grids (HHG)

Geometric multigrid methods are often considered as difficult to implement efficiently in parallel because of the reduced computational load on coarser grids. For our study the HHG framework[2] will be used. It provides parallel data structures and algorithms. In particular it features matrix-free concepts that enable the implementation of efficient parallel geometric multigrid methods.[7, 8]

3.1 Geophysical Setting

We study a problem that is motivated by geophysical simulations.[4] This leads to the Stokes problem Eq. 1 on the spherical shell $\Omega = \{\mathbf{x} \in \mathbb{R}^3 \colon 0.55 < \|\mathbf{x}\|_2 < 1\}$ and force term $\mathbf{f} = \mathrm{Ra}\,\tau \frac{x}{\|x\|}$, where $\mathrm{Ra} = 3.49649 \cdot 10^4$ is the dimensionless Rayleigh number and τ the normalised Earth's mantle temperature, as obtained from real-world measurements.[9] The mesh hierarchy has depth $L = 6$, plus 2 refinement levels used to get a structured coarse grid.

We consider either the iso-viscous case ($\nu(\mathbf{x}, T) \equiv 1$) or a viscosity profile given by lateral and radial variations:[10]

$$\nu(\mathbf{x}, T) = \exp\left(2.99\frac{1 - \|\mathbf{x}\|_2}{1 - r_{\text{cmb}}} - 4.61T\right) \begin{cases} \frac{1}{10} \cdot 6.371^3 d_a^3 & \text{for } \|\mathbf{x}\|_2 > 1 - d_a \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

where d_a is the relative thickness of the asthenosphere. Thus, the Earth mantle is assumed to have layers with different viscosity characteristics. In particular, the asthenosphere, *i. e.* the outermost layer is assumed to be mechanically weaker. In the geophysics community, determining its size is still a field of active research.[4, 10] Here, we choose a thickness of 410 km for the asthenosphere and a corresponding viscosity jump by about a factor of 145. To close the system, we apply Dirichlet boundary conditions: on the surface we use plate velocity data,[11] and at the core-mantle boundary we apply no-slip conditions. The problem can now be solved by the monolithic Uzawa multigrid method by iterating until a residual reduction by five orders of magnitude has been reached. Note that for the simulations of systems subject to several types of error, *e. g.* model or measurement error, the specified tolerance is suitable to obtain a solution with an appropriate accuracy.

3.2 Standard Iterative HHG Coarse Level Solver

Initially, we employ the standard Krylov iterative method provided by the HHG package, *i. e.* a block-preconditioned minimal residual (*pminres*) iteration. This coarsest grid iteration is executed within each V-cycle with a stopping criterion enforcing that the preconditioned residual is reduced by three orders of magnitude.

The block-preconditioner here consists of a *velocity preconditioner*. For this the velocity block is approximately solved by a Jacobi-preconditioned conjugate gradient (PCG) method and we apply a lumped mass-matrix preconditioner for the pressure. The accuracy of the PCG method is specified by a relative residual reduction of two orders of magnitude. The use of this method is motivated by the fact that Krylov space methods can be easily implemented and parallelised. However, the error reduction depends on the condition number of the system matrix that deteriorates with the mesh size tending to zero. In

extreme scale computing, even the coarsest mesh can be large and have a small mesh size, when the number of processors is very large. Consequently, an increasing number of iterations become necessary to solve the problem on the coarsest grid with sufficient accuracy. Additionally, the number of iterations can depend sensitively on the viscosity variations.

We carry out our experiments on JUWELS a peta-scale supercomputers at the Jülich Supercomputing Centre ranked on position 30 of the TOP500[a] list (June 2019). The compute nodes are Intel Xeon Platinum 8168 processors with 2-sockets, each of 24 cores. The nodes have 96 GB memory and are linked with an Infiniband interconnection system.

In Tab. 1, we present the run-times (in seconds) averaged over the iterations of the MG scheme for the scenarios *iso-viscous* and the asthenosphere scenario with 410 km thickness, denoted by *jump-410*. We observe that the scalability is overall acceptable, but somewhat better in the iso-viscous case than for variable viscosity when the efficiency decreases to less than 75 %. For a more detailed analysis, we differentiate between the fine and the coarse grid average compute times. We observe then that the average run-times for the fine grids behave robustly across the scaling to large processor numbers, but the average timings for the coarse grid solver deteriorates. Note that this is expected, since we are using a sub-optimal algorithm to solve the system on the coarsest level.

The number of iterations required to reduce the residual by five orders of magnitude depends also on the shape of the elements in the triangulation of the input mesh. Thus, the iteration number is not constant when refining the mesh due to the viscosity variation and when unfavourable element shapes are created. For the iso-viscous case, we observe that 8 iterations are needed for a moderately sized problem and that even only 4 iterations are needed for the largest problems. For the jump-410 scenario the number of iterations is around 15 iterations in all cases. Using this as the baseline performance, we now turn to study an alternative robust coarse level solver.

scenarios			iso-viscous					jump-410				
proc.	DOF	DOF coarse	iter	total	fine	coarse	par. eff.	iter	total	fine	coarse	par. eff.
1 920	$2.1 \cdot 10^{10}$	$9.22 \cdot 10^{4}$	8	58.6	57.6	1.0	1.00	15	61.0	57.9	3.1	1.00
15 360	$4.3 \cdot 10^{10}$	$6.96 \cdot 10^{5}$	4	66.1	63.2	2.9	0.89	13	83.0	62.0	21.0	0.73
43 200	$1.7 \cdot 10^{11}$	$1.94 \cdot 10^{6}$	4	68.7	65.3	3.4	0.85	14	82.0	63.7	18.3	0.74

Table 1. Average time (in seconds) over the iterations of the mildly variable V–cycle: total, fine and coarse grid timings for iso-viscous and 410 asthenosphere scenario.

4 Coarse Level Strategies

To reduce the overhead of the coarse grid solution phase, we first develop an agglomeration algorithm such that the coarse grid solution can be executed on fewer processors. This reduces communication overhead and helps to improve the scaling behaviour for very large computations.

[a] `https://www.top500.org`

Our solution strategy on the coarse level consists of the following three steps:

1. Convert HHG format to sparse matrix data-format and agglomeration,
2. Solve the coarse level problem with an external library,
3. Redistribute and convert the approximation into HHG format.

4.1 Interfacing the Coarse Level Solver

One of the major advantages of the HHG framework is the highly-efficient data format combined with matrix-free techniques, which allows treating very large systems. On the other hand, these data-structures are not naturally compatible with software packages like HYPRE, MUMPS, or PETSc. An interoperability of HHG with these packages would be useful to benefit from the functionality, variability, and efficiency of such packages. To flexibly use these different software packages, the data from HHG must be converted in a classical sparse matrix data format using arrays for indices and values, *e. g.* the coordinate list (COO) format as used by the MUMPS solver. Essential for these formats is a unique global numbering of the DOFs across the processes.[2] We proceed by an ascending order per process and number first all DOFs of one process and then continue with the next one. By these identifications, the HHG matrices can be easily converted per process into array-like data structures.

4.2 Master-Slave Agglomeration

In a multigrid iteration the number of DOFs per process decreases when the algorithm proceeds to coarser grid levels. In the case of HHG, the load of the whole multigrid hierarchy is completely distributed based on the input mesh. Hence, the balance between computation and communication worsens on coarser grid levels. We will address this problem here by executing the coarse grid subroutines on fewer processes while letting the other processes remain idle.

To achieve this, we propose a *master-slave agglomeration*,[1] where data from several *slave* processes is accumulated to few *master* processes. Here we define the reduction factor $r \in \mathbb{N}_{\geq 1}$ as the size by which the number of computing processes $|\mathcal{P}|$ is divided such that we run on: $m = |\mathcal{P}|/r$ master processes with the accumulated data and the slave processes stay idle. We assume for simplicity that the reduction factor r is a divisor of $|\mathcal{P}|$. A case with $|\mathcal{P}| = 5$ and a reduction factor $r = 3$ and $m = 2$ is shown in Fig. 1. Once the

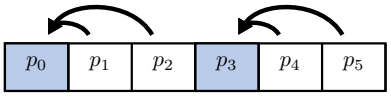

Figure 1. Master-slave agglomeration with reduction factor $r = 3$.

master processes have computed the results, they re-distribute them to the slave processes. The application to sparse matrix data formats involves array-like data-structures such as

C++ vectors. This simplifies the implementation of the agglomeration algorithm, since we only need to concatenate vectors.

Note that the selection of a suitable r is challenging and should respect the parallel architecture. In our case, we have chosen to agglomerate all the data from the same node. This makes it possible to perform the agglomeration with minimal communication overhead, but may then put extra communication burden on the parallel coarse grid solver. For our problems the agglomeration of the system matrix is performed only once, and is then kept in memory on the master processes. Note that for problems with evolving matrices, *e. g.* with time dependence or changing viscosity, the agglomerated matrix may have to be updated in each iteration.

4.3 MUMPS: A Parallel Sparse Direct Solver

MUMPS (MUltifrontal Massively Parallel direct Solver)[b] is a package for solving sparse systems of linear equations with symmetric (positive-definite or indefinite) or unsymmetric matrices. It is based on a direct method where the matrix is factorised into the product of triangular matrices which are then used to compute the solution using forward and backward substitution. The package is being developed by the MUMPS consortium. In the MG context, MUMPS has the advantage of being more robust than iterative methods, both in terms of run-time and accuracy. Also, with this solver the most consuming computations have to be performed only once for the whole MG scheme, when the factorisation of the matrix on the coarsest level can be reused.

MUMPS is a solver based on the multifrontal scheme where the factorisation of the input sparse matrix is achieved through a sequence of operations on relatively small dense matrices called "fronts".[12] Like most direct solvers, MUMPS achieves the solution of a system in three steps. Starting with a preprocessing followed by a symbolic factorisation (Analysis step), MUMPS then performs the actual factorisation of the fronts (Factorisation step). Most of the computational cost is spent in these 2 phases which are performed only once for a specific matrix. Finally, for each actual right hand side, the forward elimination and backward substitution operations are performed (Solve step).

Parallelism in MUMPS is implemented through a hybrid MPI/OpenMP model which makes the solver suited to modern distributed memory machines equipped with multicore processors.

5 Scaling Behaviour

In this section, we study the performance of the monolithic Uzawa multigrid method when a direct method is used as the coarsest level solver. The combined algorithm is implemented in the HHG framework, see Sec. 3, and the MUMPS library (version 5.1.2) is used for the coarsest grid problem, see Sec. 4.3. As example problem we study the discretised Stokes equation 1 with the settings described in Sec. 3.2. We use agglomeration techniques of 4.2 to run MUMPS on fewer processes. The computations are executed on the peta-scale supercomputer JUWELS. We start with a performance evaluation of the MUMPS solver in standalone mode for the coarsest level system of Sec. 5.1. This study is used to determine the best configuration to be used within the multigrid method of Sec. 5.2.

[b] `http://MUMPS-solver.org/`

5.1 Performance of the Direct Solver

In Tab. 2, we display results for a strong scaling study for different problem sizes of Sec. 3.2 and focus on the jump-410 problem that was found to be the most challenging in Sec. 3.2. Note that this strong scaling experiment is reversed in the sense that we start with the number of processes used for the finest grid problem such that the linear system is spread across all processors. In this mode, the coarsest grid computation suffers from too small granularity. Then we *reduce* the number of processes by an increasing reduction factor r, see equation in Sec. 4.2. We therefore find the best choice for the number of processes m that can be used for the coarsest grid problem, in order to minimise the run-time. Without agglomeration, the set of unknowns on each process is very small (less than 152 DOFs) leading to large communication overhead. This is clearly observed for the smallest problem for which the total timing with $r = 1$ is 30 times higher than with a reduction by $r = 48$. Thus using the reduced number of processes will be beneficial even if it will lead to idle processes when used within HHG. For a more detailed analysis, we also report the timings for analysis, factorisation, and solve phase separately.

To obtain minimal run-times for the problems with 1 920, 15 360 and 43 200 processes, the best choice is $r = 48$, $r = 96$ and $r = 192$, respectively. While all timings increase with the problem size, the solve step is always much faster than the factorisation and analysis, so that in cases where the factorisation can be re-used, this will lead to significantly better efficiency. Thus, when the cost of analysis and factorisation can be amortised over several multigrid iterations, the overall compute times will become competitive with using pminres as coarsest grid solver.

r	1920				15 360				43 200			
	proc.	ana.	fac.	solve	proc.	ana.	fac.	solve	proc.	ana.	fac.	solve
1	1 920	4.97	25.69	31.84	15 360	–	–	–	43 200	–	–	–
24	80	1.52	1.26	0.02	640	13.89	26.08	0.23	1 800	–	–	–
48	40	1.41	0.67	0.02	320	12.63	17.55	0.21	900	48.70	127.90	1.40
96	20	1.44	0.80	0.02	160	12.47	15.32	0.18	450	40.96	116.00	1.04
192	10	1.49	1.28	0.03	80	12.88	16.81	0.18	225	38.63	97.72	1.02
576	-	-	-	-	-	-	-	-	75	38.49	98.29	0.97

Table 2. Strong scaling study of the MUMPS sparse direct solver separated into analysis, factorisation and solve.

In a next set of experiments, we compare the efficiency of the direct solver for several configurations of the Stokes problem. In Tab. 3, we consider three different mesh resolutions of the problem for the iso-viscous and jump-410 scenarios. We use the agglomeration factors as previously obtained for each problem size. Different from the behaviour of pminres, we now observe that for a fixed problem size, the run-times remain stable for both scenarios. This is expected with an elimination based method whose run-time behaviour depends primarily only on the matrix nonzero structure. We finally observe that the direct solver produces results with an accuracy up to 10^{-18} in all cases, contrary to the the iterative pminres method. In practice, such an exact solution is not needed. Techniques to approximate the solution while decreasing the computational and memory complexity of

proc.	DOF coarse	scenario	analysis (s)	factorisation (s)	solve (s)	scaled residual
40	$9.22 \cdot 10^4$	iso-viscous	1.47	0.69	0.02	$1.8 \cdot 10^{-18}$
		jump-410	1.42	0.68	0.02	$1.9 \cdot 10^{-17}$
160	$6.96 \cdot 10^5$	iso-viscous	12.46	15.18	0.19	$5.7 \cdot 10^{-19}$
		jump-410	12.47	15.32	0.18	$1.2 \cdot 10^{-18}$
225	$1.94 \cdot 10^6$	iso-viscous	37.04	117.1	0.50	$5.31 \cdot 10^{-19}$
		jump-410	37.13	125.4	0.47	$1.27 \cdot 10^{-18}$

Table 3. Study of the influence of the viscosity scenario on the accuracy and run-time of the direct solver. Run-times are separated in analysis, factorisation and solve step. Each process runs on a separate node.

the method could be considered, such as single precision computation[13] or a suitable low rank matrix approximation.[14]

5.2 Performance Multigrid with Coarse Level Direct Solver

In this section, we apply the MUMPS sparse direct solver for the coarsest level problem in the HHG multigrid framework. We focus again on the jump-410 scenario using the two different coarsest level solvers, *i. e.* pminres from Sec. 3.2 and MUMPS as described in Sec. 4.3. In the following we will compare the average run-times of the V_{var}–cycle in a weak-scaling scenario.

In Tab. 4, we present the result of experiments with up to 43 200 processes. For a more detailed analysis, we display he fine grid and the coarse grid (*i. e.* MUMPS solve phase) average times separately for the iterations, and we sum the total time of analysis and factorisation for MUMPS. Additionally, we measure the total time for data transfer, that is, the time to perform the agglomeration and conversion between HHG and MUMPS data. According to the results from Tab. 2, we use the $r = 48, 96$ and 192, respectively, for agglomeration.

proc.	DOF		iter	time (s)					par. eff.
	fine	coarse		total	fine	coarse	ana. & fac.	trans.	
1 920	$2.1 \cdot 10^{10}$	$9.22 \cdot 10^4$	15	60.91	60.73	0.02	2.20	0.04	1.00
15 360	$4.3 \cdot 10^{10}$	$6.96 \cdot 10^5$	13	69.90	67.28	0.20	31.11	0.25	0.87
43 200	$1.7 \cdot 10^{11}$	$1.94 \cdot 10^6$	14	80.06	69.25	1.02	136.36	0.65	0.76

Table 4. Weak scaling of the V_{var}–cycle with a sparse direct and a simple Krylov coarse level solver. The run-times for total, fine and coarse are averages over the iterations. The run-times for analysis, factorisation and data transfer are the total timing.

In the scaling experiment we observe only a moderate increase in the average run-time of the fine grids, and the total time for the data transfer is neglectable in the overall timing. Moreover, the total coarse grid solves consume a fraction of less than 1.5 % of the overall time to solution for the biggest test case. However, the analysis and factorisation are much more costly, and thus these times may be a concern. Their timing grows up to $136.36\,s$

for the biggest test case which becomes $9.74\,s$ per iteration, when considering the whole solution process. For a fair comparison, this must be added to the $1.02\,s$ of the coarse level solve.

Using MUMPS decreases the times to process the coarsest grid by 40 % (resp. 80 %) as compared to the pminres solver for the largest (resp. the middle) test case for the scenario jump-410. Overall, the parallel efficiency of the solver has then been improved by 2 % points for the biggest case and 13 % points for the middle case problem.

Finally, we note that solving the coarsest grid problem very accurately with MUMPS has no effect on the multigrid convergence rate. This emphasises again the fact that such a high accuracy would not be needed and it would be beneficial to reduce the cost of the direct solver using appropriate approximation techniques.

6 Conclusion

Hierarchical hybrid grids multigrid iterative solvers in combination with an agglomeration of the processors on the coarse grid show an excellent parallel performance for large scale problems. Using a direct solver based on the MUMPS library results in an overall parallel efficiency of 76 %. As a large test case, a Stokes type system with up to $\mathcal{O}(10^{11})$ degrees of freedom has been solved with to $43\,200$ processes on the peta-scale supercomputer JUWELS.

Acknowledgements

The authors thank Patrick Amestoy, Jean-Yves L'excellent and Daniel Ruiz for their supporting discussion on efficient usage of the MUMPS framework. This work was partly supported by the German Research Foundation through the Priority Programme 1648 "Software for Exascale Computing" (SPPEXA) and WO671/11-1. The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (`www.gauss-centre.eu`) for funding this project by providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS Supercomputer JUWELS at Jülich Supercomputing Centre (JSC).

References

1. D. A. May, P. Sanan, K. Rupp, M. G. Knepley, and B. F. Smith, *Extreme-Scale Multigrid Components within PETSc*, in Proceedings of the Platform for Advanced Scientific Computing Conference, PASC 2016, Lausanne, Switzerland, Article No. 5, 2016.
2. B. Bergen and F. Hülsemann, *Hierarchical hybrid grids: Data structures and core algorithms for multigrid*, Numer. Linear Algebra Appl. **11**, 279–291, 2004.
3. B. Gmeiner, M. Huber, L. John, U. Rüde, and B. Wohlmuth, *A quantitative performance study for Stokes solvers at the extreme scale*, J. Comput. Sci. **17**, 509–521, 2016.
4. S. Bauer, M. Huber, S. Ghelichkhan, M. Mohr, U. Rüde, and B. Wohlmuth, *Large-scale simulation of mantle convection based on a new matrix-free approach*, J. Comput. Sci. **31**, 60–76, 2019.

5. T. J. R. Hughes, L. P. Franca, and M. Balestra, *A New Finite Element Formulation for Computational Fluid Dynamics: V. Circumventing the Babuška-Brezzi Condition: A Stable Petrov-Galerkin Formulation of the stokes problem accommodating equal-order interpolations*, Comput. Methods Appl. Mech. Eng. **59**, 85–99, 1986.
6. W. Zulehner, *Analysis of Iterative Methods for Saddle Point Problems: A Unified Approach*, Math. Comput. **71**, 479–505, 2002.
7. F. Hülsemann, M. Kowarschik, M. Mohr, and U. Rüde, *Parallel Geometric Multigrid*, in Numerical Solution of Partial Differential Equations on Parallel Computers, A. M. Bruaset and A. Tveito (Editors), Springer, Lecture Notes in Computational Science and Engineering **51**, 165–208, 2005.
8. B. Gmeiner, U. Rüde, H. Stengel, C. Waluga, and B. Wohlmuth, *Towards textbook efficiency for parallel multigrid*, Numer. Math. Theory Methods Appl. **8**, 22–46, 2015.
9. N. A. Simmons, S. C. Myers, G. Johannesson, E. Matzel, and S. P. Grand, *Evidence for long-lived subduction of an ancient tectonic plate beneath the southern Indian Ocean*, Geophys. Res. Lett. **42**, 9270–9278, 2015.
10. D. Rhodri Davies, S. Goes, J. H. Davies, B. S. A. Schuberth, H.-P. Bunge, and J. Ritsema, *Reconciling dynamic and seismic models of Earth's lower mantle: The dominant role of thermal heterogeneity*, Earth and Planetary Science Letters **353-354**, 253–269, 2012.
11. R. D. Müller, M. Sdrolias, C. Gaina, and W. R. Roest, *Age, spreading rates, and spreading asymmetry of the world's ocean crust*, Geochem. Geophys. Geosyst. **9**, 1525–2027, 2008.
12. P. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary, *On the Complexity of the Block Low-Rank Multifrontal Factorization*, SIAM Journal on Scientific Computing **39**, A1710–A1740, 2017.
13. M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, and S. Tomov, *Accelerating scientific computations with mixed precision algorithms*, Computer Physics Communications **180**, 2526–2533, 2009.
14. M. Bebendorf and W. Hackbusch, *Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L_∞-coefficients*, Numerische Mathematik **95**, 1–28, 2003.