Block Low Rank Single Precision Coarse Grid Solvers for Extreme Scale Multigrid Methods

Alfredo Buttari, Markus Huber, Philippe Leleux, Théo Mary, Ulrich Ruede, Barbara Wohlmuth

HAL Id: hal-02528532

https://hal.archives-ouvertes.fr/hal-02528532

Preprint submitted on 1 Apr 2020

Block Low Rank Single Precision Coarse Grid Solvers for Extreme Scale Multigrid Methods

Alfredo Buttari[1], Markus Huber[2], Philippe Leleux[3*], Theo Mary[4], Ulrich Rüde[3,5], and Barbara Wohlmuth[2]

[1] CNRS, IRIT, Toulouse, France
[2] Inst. for Numerical, Mathematics, Technical Universtiy Munich, Garching, Germany
[3] Parallel Algorithms Team, CERFACS, Toulouse, France
[4] Sorbonne Université, CNRS, LIP6, Paris, France
[5] Lehrstuhl für Systemsimulation, Friedrich-Alexander Universität, Erlangen, Germany

Abstract. Extreme scale simulation requires fast and scalable algorithms, such as multigrid methods. To achieve asymptotically optimal complexity it is essential to employ a hierarchy of grids. The cost to solve the coarsest grid system can often be neglected in sequential computings, but cannot be ignored in massively parallel executions. In this case, the coarsest grid can be large and its efficient solution becomes a challenging task. We propose solving the coarse grid system using modern, approximate sparse direct methods and investigate the expected gains compared with traditional iterative methods. Since the coarse grid system only requires an approximate solution, we show that we can leverage block low-rank techniques, combined with the use of single precision arithmetic, to significantly reduce the computational requirements of the direct solver. In the case of extreme scale computing, the coarse grid system is too large for a sequential solution, but too small to permit massively parallel efficiency. We show that the agglomeration of the coarse grid system to a subset of processors is necessary for the sparse direct solver to achieve performance. We demonstrate the efficiency of the proposed method on a Stokes-type saddle point system. We employ a monolithic Uzawa multigrid method. In particular, we show that the use of an approximate sparse direct solver for the coarse grid system can outperform that of a preconditioned minimal residual iterative method. This is demonstrated for the multigrid solution of systems of order up to 10^{11} degrees of freedom on a petascale supercomputer using 43 200 processes.

Keywords: Efficient coarse level solver · geometric multigrid · block low-rank · high-performance computing · hierarchical hybrid grids · multifrontal · sparse direct solver · MUMPS

1 Introduction

Simulations of engineering applications are often based on elliptic partial differential equations (PDEs) and require discretizations with high resolution meshes that result in huge sparse systems of equations. Multigrid (MG) methods are techniques of choice in a parallel context. MG methods can be asymptotically optimal, in the sense that the complexity to solve a linear system with sufficient accuracy grows only linearly with the number of unknowns. This has been shown for the full multigrid method [12].

[*] Corresponding author: Philippe Leleux, CERFACS, Toulouse; E-mail: leleux@cerfacs.fr

To efficiently compute the solution, MG methods combine iterative processes on a hierarchy of meshes. Algebraic multigrid (AMG) methods can construct this hierarchy for completely unstructured meshes. At the cost of a time and memory expensive setup of the coarse level matrices, the system size can then be reduced arbitrarily such that the coarsest level equations can be solved by a direct method [6]. In the case of geometric multigrid (GMG), a mesh hierarchy is required such that meshes resolve the computational domain with a certain accuracy. Hence, systems that are still relatively large must be processed even on the coarsest mesh level.

Here, we employ the hierarchical hybrid grids (*HHG*) [24,11] framework that implements GMG methods for the solution of linear systems and which achieves excellent performance on state-of-the-art petascale supercomputers where problems with more than 10^{13} degrees of freedom (*DOFs*) [19] have been solved. We study the solution of saddle point problems arising from the Stokes equation. These problems are often solved using a Schur complement conjugate gradient (*CG*) algorithm, or a preconditioned minimum residual method. Our method of choice is a monolithic MG method using an Uzawa smoother, see Sect. 2, combined with a classical mildly variable multigrid V-cycle, where the number of smoothing steps is linearly increased on coarser levels. This *all-at-once* multigrid treatment features lower memory requirements and achieves a faster time-to-solution for Stokes flow equations, see [16,19].

The problems under study here permit the use of Krylov space methods as solver on the coarsest level. These methods will require a number of iterations growing mildly with the size of the coarsest grid. Though asymptotically not optimal, the incurred overhead is in many cases still acceptable as long as the runtime is dominated by the multigrid processing of finer grids. However, for numerically challenging problems and when the coarsest grid size is relatively large, such simple coarse grid solvers may become a bottleneck, especially since each iteration incurs a significant overhead. In [32], GMG and AMG methods are combined to obtain scalability. First, a mesh refinement of the input grid is generated such that GMG method can be applied. Then at the coarsest level (the input mesh), the AMG method is used to further reduce the system's complexity and solve the coarse grid problem of the GMG method. In the latter AMG method, the solution of the coarsest grid problem is delegated to a sequential direct solver that can deliver an accurate solution in a robust and reliable way. This, however, comes at the price of a high operational and memory cost.

In this work, we consider the use of a modern, approximate sparse direct solver based on low-rank approximations — a technique which can significantly reduce the asymptotic complexity of the solver at the price of a controlled loss of accuracy. An approximate direct solver is acceptable for the purpose of solving the MG coarse grid problem [27,12]. Even though several low-rank techniques have been proposed in the past, to the best of our knowledge, this article represents the first attempt at applying them to extreme scale multigrid solvers.

Scaling MG methods on the largest supercomputers is a challenging task and requires well-designed software structures and advanced performance aware implementation techniques. The deterioration of the parallel efficiency on coarser grid levels is especially problematic in MG solvers. On coarser grid levels, the amount of computation decreases at a faster rate than the communication volume, and so the communication overhead becomes larger. To alleviate this trend, a better load-balancing and possibly a redistribution of the grids on fewer processes, namely agglomeration, may become necessary. The need for agglomeration techniques is emphasized, e.g., in [28] where recursive process agglomeration is used to scale MG in PETSc [7]. The general idea here is to adapt the number of working processes to the size of the problem to achieve a better balance between communication and computation. The coarser the problem, the smaller the number of processes involved, in order to avoid an unnecessary large volume of communication. While in this article, we base our agglomeration strategy on heuristics, a performance model is used in [30] for a structured MG scenario to predict the best agglomeration strategy. In [31], the GMG solver is constructed by starting from a coarse grid. In each successive step of refinement, the grids are then distributed to the available processes.

In this article, we study the efficiency of MG with an agglomeration method applied to an approximate direct solver as coarse level solver for examples with increasing size and model complexity. We use the MUMPS solver [5,4] that is based on the block low-rank (BLR) format [1]. We combine this approach with an additional reduction of memory and computational cost through the use of single precision floating-point arithmetic. Scaling results are presented for up to 43 200 processes on the Hazel Hen petascale supercomputer. Our results demonstrate that the BLR method, combined with agglomeration and single precision arithmetic, can be used for approximating the coarse level problem in large scale simulations with improved efficiency. The total time spent on the coarse grid over the MG cycles is decreased by up to 50% when using MUMPS in single precision with block low-rank approximation compared to using a Krylov based iterative solver.

The rest of this article is structured as follows: In Sect. 2, we briefly introduce the model, the finite element discretization and the general multigrid setup. Then, in Sect. 3.1, the data structure, the parallelization and the matrix-free techniques of the HHG framework are presented. Sect. 3.2 focuses on the deteriorating scalability of a simple Krylov based coarse level solver. In Sect. 4.1, we describe the conversion of the HHG data structure to standard sparse matrix data formats such that external solver libraries can be interfaced. In Sect. 4.2, we describe the master-slave agglomeration technique. Then, we introduce the MUMPS framework in Sect. 5.1. The BLR method within the MUMPS framework is presented in Sect. 5.2. Sect. 6.1 considers MUMPS as standalone solver in several scaling experiments and Sect. 6.2 analyses the combined solvers in a weak scaling scenario. We show that the new coarse grid solvers leads to a faster overall time to solution when the model problem is sufficiently large and numerically hard.

2 Model problem, discretization and solver

Let $\Omega \subset \mathbb{R}^3$ be an open and bounded domain. We consider the Stokes-type problem with velocity $\mathbf{u}$ and pressure $\mathbf{p}$ of the form

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

with the forcing term $\mathbf{f}$, the Dirichlet boundary conditions $\mathbf{g}$ and the positive scalar viscosity ν. Here we assume that $\mathbf{g}$ satisfies the compatibility condition $\int_{\partial\Omega} \mathbf{g}\cdot\mathbf{n}\; ds = 0$ where $\mathbf{n}$ is the unit outer-normal. Problems of this structure can be found in mantle convection simulations where they represent the most time-consuming computational tasks [8]. The viscosity in such problems can vary by several orders of magnitude and is typically non-linearly depending on $\mathbf{u}$. For simplicity, we here neglect the non-linearity for the following consideration. We impose non-homogeneous Dirichlet boundary conditions derived from plate velocity data obtained by [29] on the surface and no-slip conditions at the core-mantle boundary.

We discretize Ω by 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 discretization of (1), we apply the equal-order linear finite elements for velocity and pressure, see e.g. [17]. This equal-order discretization is known to be unstable and must be stabilized [13]. To this end, we apply the pressure stabilization Petrov-Galerkin (PSPG) technique [23]. Using (component-wise) nodal basis functions for velocity and pressure, we obtain a hierarchy of 2×2-block structured linear systems

$$\begin{pmatrix} A_\ell & G_\ell \\ D_\ell & -C_\ell \end{pmatrix} \begin{pmatrix} \mathbf{u}_\ell \\ \mathbf{p}_\ell \end{pmatrix} = \begin{pmatrix} \mathbf{f}_\ell \\ \mathbf{g}_\ell \end{pmatrix} \tag{2}$$

with $\mathbf{u}_\ell \in \mathbb{R}^{n_{\mathbf{u};\ell}}$ and $\mathbf{p}_\ell \in \mathbb{R}^{n_{\mathbf{p};\ell}}$. The dimensions of the velocity and the pressure space are denoted by $n_{\mathbf{u};\ell}$ and $n_{\mathbf{p};\ell}$. The divergence of the deviatoric stress operator in (1) can be associated with A_ℓ, the gradient with G_ℓ, and the divergence operator with D_ℓ. The C_ℓ-block originates from the PSPG-stabilization.

The efficient solution of this block system on large scale computations is studied in a number of recent articles [14,35]. In [19], different types of solvers are compared and it is found that a monolithic multigrid method for velocity and pressure combined performs best in terms of time-to-solution and memory. The family of uniformly refined meshes $\mathcal{T}$ is used to construct the multigrid mesh hierarchy. We will use this hierarchy to construct a geometric multigrid method in the form of a mildly variable $V-$cycle that we will denote by V_{var}. The $V_{\text{var}}-$cycle has the same form as a $V-$cycle but adds for each coarser level an additional number of smoothing steps. In our case, we add for each level two additional steps, each in the pre- and post-smoothing.

In this method, an Uzawa-type smoother is used that acts on velocity and pressure unknowns separately, see [36] and [16]. In the following, we will refer to this monolithic MG variant as the *all-at-once* Uzawa MG method. The transfer operators are defined as linear interpolation for each component and their adjoint operators for restriction. Since this multigrid method acts on the whole Stokes system, we have to solve on the coarsest grid level again a saddle point problem. Although this problem is by several orders of magnitude smaller than the fine grid problem, it can still become large for extreme scale simulations. In theoretical considerations, one often assumes that the coarsest grid problem is solved exactly. Getting this high accuracy can become computationally expensive in practice, e.g. using a direct solver. A popular alternative is to solve this coarse problem approximately. In this case, the tolerance has to be carefully selected to keep a mesh independent convergence of the multigrid scheme. Our present study will explore different strategies to efficiently get such an approximated solution on the coarse grid problem in large scale and extreme scale computations.

3 Hierarchical hybrid grids

For our studies the *hierarchical hybrid grids* (HHG) framework [11] will be used that provides data structures, parallelization and matrix-free concepts for extreme scale geometric multigrid computations. Here HHG serves as test environment to explore coarse level strategies in large scale simulations. For similar data structure concepts, we refer to [25,18].

3.1 Data structure, parallelization, matrix-free assembly

In this section, we briefly review the data structures and the parallel implementation of the considered multigrid framework. For more details, we refer to [20,19,8] and the references therein. HHG organizes the nodal points of the mesh by employing the hierarchy of uniformly structured meshes $\mathcal{T}$. Through the refinement, grid points are generated on the edges, faces and within each input grid tetrahedron, also called macro tetrahedron. In the case of two tetrahedra this is illustrated in Fig. 1 (left). Each of these nodes on each level of refinement are located on either the vertex, edge, face or the volume of the original macro tetrahedra. This is used to classify the nodes on each mesh level and to define container data structures that guarantee a unique assignment of each node to one container. In a distributed memory architecture, we can assign each container uniquely to one processor.

To enable an efficient parallel communication across process boundaries, an additional layer of halos (ghost layers) is introduced that holds copies of *master* data, i.e. the original data, on other memory units. The data in these ghost layers can only be read and the values must be updated when the master data is modified so that they hold consistent values. When the respective container is located on the same memory unit, this is achieved with a simple memory copy routine, otherwise, when the data is on different memory units in the network, it is sent by message passing using the message passing interface (MPI). Eventually, it is necessary to introduce additional copies of vertex, edge and face data structure to implement the MPI communication efficiently. In Fig. 1 (right), the ghost layer enrichment for two input mesh tetrahedra and the face container between them is illustrated.

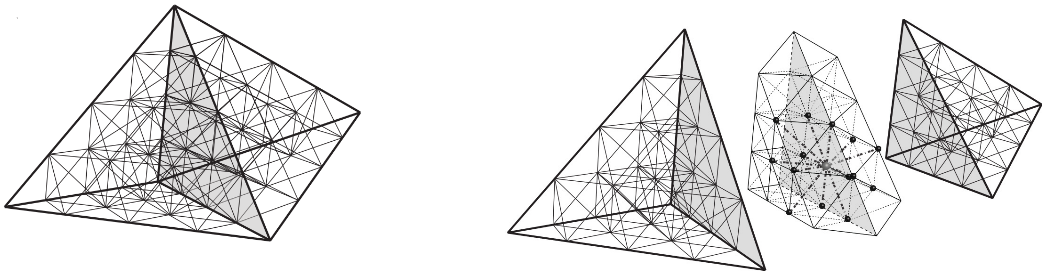

Fig. 1. left: two refined input elements; right: ghost layer structure of two input elements.

To enable efficient parallel computations, load-balancing is also an important aspect. In HHG, the computational load can be identified with the dimensional complexity of the container data structures. Asymptotically, the volume containers produce the largest computational load, since they hold 3D data. Therefore, they are equally distributed to computing processes. The lower dimensional containers are then assigned to the processes with the largest process id of the physically neighboring volume containers. The same distribution is used for the whole mesh hierarchy such that the communication structure does not change between mesh levels. A drawback is that lower dimensional container are not equally distributed, but in our previous studies, no consequent major load imbalance was observed. However, we have found that the coarse grid may become a performance bottleneck. This will be considered in more detail in Sect. 4.

Matrix-free techniques are applied within HHG to avoid storing the FE matrices. In the classical assembly of the HHG framework only one stencil (i.e. a matrix row) needs to be stored per container so that superior performance [11] can be achieved. However, for curved domains such as the spherical shell that we consider in the following, the nodes that are generated through refinement do not reside on the boundary and thus do not fit with a simple single stencil representation. In this case, the fine grid nodes must be projected onto the spherical surface, leading to different entries in each row of the stiffness matrix, i.e. for each stencil. Thus the FE stencils have to repeatedly be recomputed for each node and each time they are applied, e.g., for a matrix-vector multiplications. In [9,8], we have developed an efficient method that recovers the performance of the original HHG implementation also on curved domains and for problems with variable viscosity. This *surrogate* assembly technique replaces the computation of the local element matrices by an inexpensive polynomial approximation.

3.2 Matrix-free Krylov based coarse level solver

We study a problem that is motivated by geophysical simulations [8]. This leads to the Stokes problem (1) on the spherical shell $\Omega = \{\mathbf{x} \in \mathbb{R}^3 \colon r_{\text{cmb}} < \|\mathbf{x}\|_2 < r_{\text{srf}}\}$, where $r_{\text{cmb}} = 0.55$ and $r_{\text{srf}} = 1$ correspond to the inner and outer mantle boundary, and force term $\mathbf{f} = \text{Ra}\,\tau \frac{x}{\|x\|}$, where $\text{Ra} = 3.49649 \cdot 10^4$ is the dimensionless Rayleigh number and τ the normalized Earth's mantle temperature, as obtained from real-world measurements [33]. We discretize the spherical domain by an initial mesh $\mathcal{T}_0$ and apply the uniform refinement strategy to construct the hierarchy of meshes $\mathcal{T}$. The mesh hierarchy has a depth of $L = 8$, where 2 levels give a properly defined coarsest grid problem, and 6 levels are used in the MG hierarchy.

On finer grids, the boundary grid points would not reside on the curved boundary so that we use the previously described projection method and apply the matrix-free surrogate assembly technique for problems with different viscosity variations. In particular, we consider either the iso-viscous case ($\nu(\mathbf{x}, T) \equiv 1$) or a viscosity profile, similar to the one used in [15], given by lateral and radial variations

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

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 depth is still an open research question [15,8]. Here, we choose a depth of 410 km that corresponds to a viscosity jump of a factor 145. To close the system, we apply the suitable Dirichlet boundary conditions introduced in Sect. 2. Eq. (2) can now be solved using the monolithic Uzawa multigrid method by iterating until a residual reduction of five orders of magnitude has been reached. Note, in geodynamic simulations that are subject to several types of error, e.g., model or measurement error, the specified tolerance is suitable to obtain a solution with an approriate accuracy.

It now remains to choose a coarse level solver. Initially, we employ the standard method provided by the HHG package, i.e., a block-preconditioned minimal residual (*PMINRES*) iteration. This choice is motivated by the fact that Krylov space methods can be easily implemented and parallelized. This is executed until the coarse level problem in each V-cycle has been solved with an accuracy corresponding to a reduction of the preconditioned residual by three orders of magnitude. The preconditioner here consists of *velocity* and *pressure* block preconditioner. For the velocity block, a Jacobi-preconditioned conjugate gradient (PCG) method is applied and for the pressure block a scaling by the lumped mass-matrix preconditioner for the pressure is used. The accuracy of the PCG method is specified by a relative residual reduction of two orders of magnitude. However, the error reduction depends on the condition number of the system matrix which deteriorates with the mesh size, and an increasing number of iterations becomes necessary to solve the coarse grid problem with sufficient accuracy. The efficiency of the approach in many cases of interest has been

Table 1. Total run-times (in seconds) of the V_{var} application: total, fine and coarse grid timings for the asthenosphere scenarios iso-viscous and jump-410. The number of iterations of the MG method (*it*) and the average number of iterations of the coarse grid solver (*C.it*) are also displayed.

proc.	**DOFs**		**iso-viscous**						**jump-410**					
	fine	**coarse**	**it**	**total**	**fine**	**coarse**	**eff.**	**C.it**	**it**	**total**	**fine**	**coarse**	**eff.**	**C.it**
1 920	$5.37 \cdot 10^{9}$	$9.22 \cdot 10^{4}$	4	312.8	309.0	3.8	1.00	25.13	15	1186.0	1132.6	53.4	1.00	68.13
15 360	$4.29 \cdot 10^{10}$	$6.96 \cdot 10^{5}$	5	439.5	429.8	9.7	0.89	18.40	13	1188.0	1091.3	96.8	0.87	48.62
43 200	$1.21 \cdot 10^{11}$	$1.94 \cdot 10^{6}$	8	735.0	713.1	21.9	0.85	17.00	14	1404.0	1241.5	162.5	0.79	48.43

demonstrated in previous publications [19], but also its limitations have been shown when viscosity models as (3) are considered [8].

We carry out our experiments on Hazel Hen, a petascale supercomputer at the HLRS in Stuttgart ranked on position 35 of the TOP500[‖] list (November 2019). Hazel Hen is a Cray XC40 system with Haswell Intel Xeon E5-2680 v3 processors. Each compute node is a 2-socket system, where the 12 cores of each processor constitute a separate NUMA (non-uniform memory access) domain. Hazel Hen offers 64 GB per NUMA domain, which means around 5.3 GB per core. Hazel Hen uses the Cray Aries interconnect. The supercomputer has 185 088 cores in total for a theoretical peak performance of 7.42 PFLOPS/s.

In Tab. 1, we present the total run-times (in seconds) of a V_{var}–cycle application for the scenario *iso-viscous* and the scenario *jump-410*, where the asthenosphere has a depth of 410 km. The displayed parallel efficiency is equal to the average total timing per iteration for the middle and large test cases compared to the average total timing per iteration for the smallest one. We observe that the scalability is better in the iso-viscous case than for variable viscosity jump-410, where the efficiency decreases to less than 80%. For a more detailed analysis of the run-time behavior, we also distinguish between the finer grids and coarsest grid compute times. While the average run-time for the fine grids stays stable for both scenarios, resp. 89.14s and 88.68s for the largest problem, the average run-time for the coarse grid solution is getting worse with 11.61s with the scenario jump-410, compared to 2.74s in the case of iso-viscous. This is explained by the increased average number of iterations (*C.it*) for the convergence of PMINRES in the jump-410 scenario. Also, we observe that the average run-time per iteration for the fine grids are robust in the weak scaling while the average timing per iteration for the coarse grid deteriorates. Note that this is expected, since we are using a sub-optimal coarse level solver and since the coarse grid problem size grows when scaling to larger number of processors.

The number of iterations required for the V_{var}–cycle to reduce the residual by five orders of magnitude depends also on the shape of the elements in the triangulation of the input mesh. The iteration number is not constant when refining the mesh due to the viscosity variation and possibly ill-shaped elements. For the iso-viscous case, we observe that the iteration number decreases to only four iterations, while for the scenario jump-410 it still remains below 15 iterations. In the following, we propose an alternative fast and robust coarse level solver.

[‖] `https://www.top500.org`

4 Coarse level strategies

Scaling GMG on large computing systems is hard in the sense that on the coarse grid levels the granularity deteriorates. Commonly, an agglomeration of data onto fewer processors is used to compensate for this effect. Thus the coarsest grid problems are redistributed. The remaining processors can either perform redundant computations or the unneeded processors stay idle [31,30]. To make different coarse grid solvers available and thus help improve the HHG multigrid efficiency, we link an external software library to the HHG framework. The efficiency of these external solvers is also limited with excessive number of processors so that agglomerating the coarse grid data to a suitable number of processors is essential. In our implementation, processors which are not used by the external solver will stay idle. Within the MUMPS solver, for instance, some of the idle cores from the agglomeration could be employed for an increased MPI distributed parallelism or for a shared memory parallelism using OpenMP. However, in practice the granularity of each parallel task could then decrease further, thus slowing down the execution. A systematic study of shared versus distributed memory parallelism in MUMPS is out of the scope of this paper.

Our solution strategy at the coarse level consists of the following four steps:

1. Convert HHG format to sparse matrix data-format.
2. Apply the agglomeration technique.
3. Solve the coarse level problem by the external library.
4. Redistribute and convert the approximation to the HHG format.

4.1 Interfacing the coarse level solver

One of the biggest advantages of the HHG framework is the highly efficient data format, which allows to treat systems with a large number of DOFs. On the other hand, the data structure is not directly suited to link other software packages like HYPRE, MUMPS, PARDISO, PETSc, or Trilinos , as they rely on assembled matrice contrary to HHG. Being interoperable and making the flexibility and efficiency of these libraries available is, of course, an important feature for HHG.

Standard sparse matrix data formats have been defined as interfaces to solver libraries. We will employ the *compressed-row storage* (CRS) format. Also other sparse matrix formats will be supported like the *coordinate list* (COO) format since it is also used within the MUMPS solver. Note here that despite the compressed formats, setting up the sparse matrix can be a costly operation both in terms of processing to create the index structures, data copying, as well as memory consumption. Also, if we link the software implemented in double precision to a single precision coarse grid solver, overhead from static casting to lower precision arithmetic appears. On the fine mesh levels, the matrix-free HHG methods can be much more efficient than processing the stored matrices.

Essential for all sparse matrix formats is a unique global numbering of the DOFs. We proceed with an order ascending with the process rank. We first number all DOFs of one process and then continue with the next one. By these identifications, the HHG matrices can easily be locally converted on each process in array-like data structures.

4.2 Master-slave agglomeration

As discussed before, the number of DOFs per process decreases on coarser grid levels. When the balance between computation and communication worsens, and the communication overhead becomes a concern, then we propose to accumulate the data of several processes onto a single process.

We achieve this by a *master-slave organization* that is similar to [28] but is here implemented within the HHG framework. In this method, we collect the data from several *slave* processes and accumulate it to one *master* process. This defines a reduction factor $r \in \mathbb{N}_{\geq 1}$ denoting how much the overall process count $|\mathcal{P}|$ is reduced such that we get

$$m = |\mathcal{P}|/r \tag{4}$$

master processes. Here, we assume, for simplicity, that the reduction factor r is a divisor of $|\mathcal{P}|$. A case with reduction factor $r = 3$ and $m = 2$ is shown in Fig. 2. The master processes then execute the computations on the accumulated data. The slave processes stay idle during these computations and wait for the master processes to finish. Once the master processes have computed the results, these must still be re-distributed to the slave processes. The application to sparse matrix data formats involves array-like data structures like C++ vectors, which make the agglomeration technique easy to implement and efficient to apply, since we only need to concatenate vectors and let the external solver be executed by a reduced communicator.

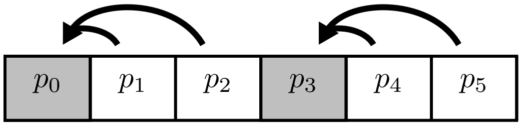

Fig. 2. Showcase for master-slave agglomeration with reduction factor $r = 3$.

Note that the selection of a suitable r is challenging. It depends on the granularity of the problems solved by the direct solver and may have to be determined on a case by case basis. Using the factor r, the agglomeration method can be adapted to the parallel architecture of the machine: in our case, we have chosen to agglomerate all the data that resides in the same node. This makes it possible to perform the agglomeration with small communication overhead, but may then put extra communication burden on the parallel coarse grid solver. At the other extreme, compacting all the processes inside a same node puts in practice the burden on the memory because of the memory bound dense kernels used by MUMPS, which is not advisable. These arguments are very pragmatic, as well as the agglomeration approach. 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 time dependent problems or when the viscosity of the problem changes, the agglomerated matrix may have to be updated in each iteration, which is not the case for the problem considered here.

5 MUMPS: a parallel sparse direct solver

MUMPS (MUltifrontal Massively Parallel direct Solver)[**] [5,4] is a package for solving sparse systems of linear equations like (2) with symmetric (positive-definite or indefinite) or unsymmetric matrices, using single or double precision real/complex arithmetic. It is based on a direct method where the matrix is factorized into the product of triangular matrices which are then used to compute the solution through triangular system solves. In our multigrid context, the use of a sparse direct solver such as MUMPS as coarse level solver provides two distinct advantages:

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

1. Robustness in the sense of the accuracy of the solution, with stable run-times, in cases where standard iterative solvers show slow convergence.
2. Saving the analysis and factorization in memory, the most time consuming parts, in a preprocessing step. Fast coarse level solves through application of the stored factorization for each multigrid cycle.

5.1 Method

MUMPS's solver is based on the multifrontal scheme where the factorization of the input sparse matrix is achieved through a sequence of operations on relatively small dense matrices called fronts; we refer the reader to [5] for a thorough description of this approach. Like most direct solvers, MUMPS achieves the solution of a system in three steps:

1. *Analysis*: at this step a pre-processing of the matrix is performed in order to reduce the fill-in (i.e., zero coefficients that are turned into nonzeros by the numerical factorization) and improve certain numerical properties of the system (scaling, permutation to a zero-free diagonal); this is followed by a symbolic factorization which defines the dependencies between the unknowns of the system and organizes the computations to be performed in the next step.
2. *Factorization*: this step computes the numerical factorization of the system matrix, based on the outcome of the analysis phase. Two levels of parallelism are available here: one is intrinsic to the dense linear algebra operations that are applied to each front, the other level comes from the fact that fronts that do not depend on one another (this relationship is established in the analysis phase) can be processed concurrently.
3. *Solve*: in this step the factors computed by the factorization are used to compute the solution by means of forward elimination and backward substitution operations.

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.

5.2 Block low-rank approximation

In multigrid methods, the coarse grid is assumed to be solved exactly in theoretical considerations; in practice, it is common to approximate the coarse grid solution up to a given tolerance. For this reason, iterative methods are often preferred over standard direct ones which do not allow to control the accuracy of the solution. However, with the *block low-rank* (BLR) method, MUMPS offers a mechanism that allows for reducing the cost of the solution if a lower accuracy is acceptable.

Full rank sparse matrices also result in full rank fronts in the sparse factorization. Nonetheless, it can be proven that for problems in a very broad class of applications, conveniently defined off-diagonal blocks of the fronts can be approximated with accuracy ε using a low-rank product [10]. Depending on the desired approximation accuracy, this representation can be more or less compact but in most cases, even with an accuracy close to the working precision, this mechanism allows for considerably reducing the cost of the linear algebra algorithms both in terms of memory consumption and floating point operations. Several approaches have been proposed in the literature to take advantage of this low-rank property. The MUMPS solver is based on a matrix format called BLR [1,4] where the matrix is partitioned into blocks in a checkerboard fashion and blockwise low-rank approximations are exploited to significantly reduce the theoretical complexity [2] and practical cost of the factorization and solve phases. Although even lower theoretical complexities

can be achieved by using multilevel [3] or hierarchical [21] approximations, the flexible BLR format has proven to be very efficient in the context of a general purpose, fully-featured sparse solver such as MUMPS [4,26].
Additionally, running MUMPS in single precision arithmetic decreases the overall memory usage and time consumption of the solver. This arithmetic can be combined with the BLR approximation freely with no loss in the accuracy of the solution, as soon as the choice of the parameter ε gives an approximation with accuracy lower or equal to single precision [22].

6 Scaling experiments

In this section, we study the performance of the multigrid solver when combined with a block low-rank method using single precision arithmetic as coarse level solver for the Stokes problem introduced in Sect. 3.2. We start with a performance study of the MUMPS solver standalone on the coarse level system in the Sect. 6.1. Then, we use the best obtained configurations to improve the overall multigrid solver performance in a weak scaling test in Sect. 6.2.

6.1 Performance of the approximate sparse direct solver with agglomeration

While HHG performs well on the fine grids with a large number of processes, the performance of MUMPS deteriorates in practice when run on too many processes, because of the lower granularity and memory-bound behaviour of the internal linear algebra kernels. Therefore, we use the agglomeration technique introduced in Sect. 4.2 as a means to reduce the number of processes when executing MUMPS. To find efficient agglomeration factors, we first study MUMPS as a standalone solver, with exact double precision accuracy, for the coarse grid matrices extracted from HHG. In Tab. 2, we display the data for *reverse* strong scaling studies and for all problem sizes of Sect. 3.2. The focus lies on the more challenging problem variant, i.e. the jump-410 scenario.

Reverse strong scaling differs from classical strong scaling in the sense that here, we start with the same number of processes and nodes as they arise for the finest grid, and then reduce the number of processes by an increasing reduction factor r (see (4)). We consider that r is a divisor of the original number of processes. The goal is to find a suitable choice for the number of processes m that can be used for the coarse grid problem, in order to minimize the run-time. We are looking for a trade-off between a high number of active cores (r not too high) and a combination of large granularity and low memory concurrency for the solver (r not too low), both cases leading to a shorter run-time. Note, that the number of executing processes is limited by the number of processes used for the fine grid problem. Using all fine grid processes for the coarse grid produces a huge run-time due to the excessively high volume of communications. In this scenario, the set of unknowns for each process is very small (less than 152 DOFs). This is clear for the smallest problem and $r = 1$, for which the total timing is more than 10 times higher than after a reduction by $r = 24$, which corresponds to removing concurrent memory access by accumulating the data of a whole node to only one process. This explains why a reduction factor $r \geq 24$ is reasonable for all sizes. Then, increasing r further will decrease the overhead in communication until a too high r gives low parallelization compared to the granularity of the problem. We observe this effect for the biggest case where having $r = 192$ decreases the total timing by around 40% compared to $r = 24$, and having $r = 576$ then increases the total run-time again slightly.

When integrated into HHG, many processes will stay idle when executing the MUMPS solver and when we run the problem only with the reduced number of processes. For instance, for the smallest test case and $r = 48$, only 40 processes execute MUMPS, each on a single node. Furthermore, to obtain minimal run-times, while increasing the problem size, we must further increase the reduction factor: for the problem with 1 920 processes the best choice is $r = 48$, with 15 360 processes, $r = 92$, and with 43 200 processes, $r = 192$. For a more detailed analysis of the MUMPS solver, we report the timings for analysis, factorization, and solve phase separately.

While all the timings increase when increasing the problem sizes, we observe only a very small run-time for MUMPS solve phase, which is performed at each cycle. Thus, when the cost of analysis and factorization, required only once in HHG, can be amortized over several multigrid iterations, the overall compute times will compare favorably with the PMINRES coarse grid solver (see Table 1). In the case of a dynamic or non-linear problem, multiple MG solve would be needed and, for each of these, new factorization and possibly analysis steps of MUMPS are required again. Additionally, good start iterates can be derived from previous time step or Newton step, thus the number of iterations is reduced. In such a case, the ratio between set-up costs and total costs shifts as the timing of these early steps of MUMPS becomes prohibitive.

Table 2. Reverse strong scaling study of the MUMPS sparse direct solver, with exact double precision accuracy: analysis, factorizaton and solve run-times in seconds.

r	**1920**				**15 360**				**43 200**			
	m	**analysis**	**fac.**	**solve**	**m**	**analysis**	**fac.**	**solve**	**m**	**analysis**	**fac.**	**solve**
1	1 920	6.51	10.80	13.66	-	-	-	-	-	-	-	-
24	80	1.62	1.08	0.03	640	15.56	31.09	0.86	1 800	66.56	248.23	1.87
48	**40**	1.55	0.88	0.03	320	14.62	20.74	0.28	900	44.98	199.51	0.67
96	20	1.61	1.19	0.03	**160**	13.74	19.58	0.20	450	53.61	173.04	0.73
192	10	1.72	1.66	0.07	80	14.36	24.05	0.22	**225**	41.02	134.61	0.56
576	-	-	-	-	-	-	-	-	75	42.92	158.41	0.59

In a next step, we fix the optimal reduction factor as found in the previous study for MUMPS in full-rank, assuming it is a good choice also with different parameters. Then, we continue by comparing the performance of the BLR method, in double and single precision, with the standard sparse direct solver. In order to have the same setup for the coarse grid solver as in HHG after agglomeration, each process runs on a separate node. As it is commonly assumed and many experiments have shown, the coarse level problem within a multigrid method does not require an exact solve. Thus the BLR method can help to further improve the run-times. Here, the choice of the BLR ϵ parameter is important, since it controls the accuracy of the approximation and the performance of the factorization. Furthermore, we are interested in the robustness of the BLR ϵ threshold for different problem setups in terms of solution accuracy, when increasing the problem sizes.

In Tab. 3, we consider three different resolutions of the problem for the iso-viscous and for the jump-410 scenarios. We compare three settings: the Full Rank setting does not exploit BLR approximations and computes a sparse direct solve to machine precision accuracy, whereas the other two setting use the BLR solver. The accuracy of the approximated solution for the different setups is given by the scaled residual. With $\epsilon = 10^{-8}$, the scaled residual is about 10^{-10}, while with $\epsilon = 10^{-3}$ it is about 10^{-4}. This is the accuracy level that we typically need in order to keep the convergence of the MG scheme unchanged. As we will see in the next section, the largest problem is the most critical one and a more detailed study is required such that we also include data for BLR with $\epsilon = 10^{-4}, 10^{-5}$. The processor specification remains unchanged compared to the previous experiments. In all cases, the BLR ϵ gives an upper bound for the scaled residual. In all cases here, the true residual is found to be more than a factor of 3 smaller than the BLR ϵ parameter. Hence, as theoretically proven in [22], the accuracy can be controlled by the BLR ϵ and stays robust for different problem sizes when including viscosity variations. Furthermore, as long as ϵ stays safely below 10^{-8}, we can use single precision arithmetic without changing the quality of the solution [22]. This can help to further decrease the cost in memory and computation.

When applying the Full Rank solve in comparison to the BLR method for different ϵ parameters, we observe a small increase in the timings for the analysis, around 15% for the biggest case. This overhead comes from the identification of blocks for the factorization, necessary to the BLR method. The biggest effect of the BLR approximations on the performance of MUMPS can be found in the factorization. The first evidence is the reduction of FLOPS for the factorization: when using BLR $\epsilon = 10^{-3}$ in comparison to the Full Rank factorization, the FLOPS are reduced up to a factor 10 for the biggest test case. Then, the direct consequence is a reduction of the run-time for the factorization when applying the BLR method, e.g. the factorization is 4 times faster with BLR $\epsilon = 10^{-3}$ on the biggest test case. Additionaly, the use of BLR approximations also reduces the cost of the solve phase, by about a factor 2 for the biggest test case.

To further accelerate the computation, we also turn to single precision arithmetic. In this case, the analysis and solve phases stay almost unchanged, while we get a reduction of 30% of the factorization, whose run-time was originally dominating. Overall, using BLR combined with single precision reduces the total run-time of a complete MUMPS computation by a factor up to 2.6 for the largest problem.

Finally, for a fixed problem size, the run-times for all phases of the solver as well as the accuracy of the solution appear robust with respect to the problem type iso-viscous or jump-410. Comparing MUMPS results on these 2 problem types, the accuracy of the solution only varies by up to an order of magnitude, while the run-times are in the same ranges. For this reason, and because it is the worst case scenario for the PMINRES solver, we focus on the most challenging scenario jump-410 in the rest of the paper.

6.2 Performance of the multigrid solver with the approximate coarse level solver

In this section, we use the MUMPS sparse direct solver and its BLR variant to approximate the coarsest level problem within the Uzawa multigrid solver in the HHG framework. We compare the run-times of one V_{var}–cycle for this strategy with the ones using the PMINRES solver on the coarsest grid of Sect. 3.2 in a weak scaling test.

In Tab. 4, we present the total run-times with up to 43 200 processes. We present again the total run-time over the V_{var}-cycle iterations. To study the run-times in detail, we display the fine grid run-time, the run-time for MUMPS analysis and factorization cumulated and the coarse grid run-time (i.e. MUMPS solve phase) separately. Additionally, we include the total time for data transfer, that is, the time for agglomeration as well as converting HHG to MUMPS data and vice versa. For the performance of the coarse level solver, it is essential to choose efficient agglomeration factors. According to our study in Tab. 2, we use the factors $r = 48, 96$ and 192 respectively for agglomeration.

Table 3. Study of the influence of the viscosity scenario and BLR ϵ parameter, with double and single precision, on the accuracy and the run-time of the direct solver. Run-times are separated in analysis, factorization and solve steps. Each process runs on a separate node.

proc.	**DOFs coarse**	**type**	**BLR ϵ**	**analysis**	**factorization**		**solve**	**scaled res.**
				time (s)	**Flops red.**	**time (s)**	**time (s)**	
40	$9.22 \cdot 10^4$	iso-viscous	Full Rank	1.51	100.0	0.86	0.03	$1.2 \cdot 10^{-18}$
			10^{-3}	1.73	26.2	0.86	0.02	$6.6 \cdot 10^{-5}$
			10^{-3} + single	1.74	26.2	0.70	0.01	$6.7 \cdot 10^{-5}$
		jump-410	Full Rank	1.55	100.0	0.88	0.03	$6.0 \cdot 10^{-18}$
			10^{-3}	1.81	28.5	0.91	0.02	$3.7 \cdot 10^{-4}$
			10^{-3} + single	1.74	26.0	0.67	0.01	$2.5 \cdot 10^{-4}$
160	$6.96 \cdot 10^5$	iso-viscous	Full Rank	13.73	100.0	20.65	0.21	$1.1 \cdot 10^{-18}$
			10^{-3}	15.82	10.8	9.61	0.10	$2.2 \cdot 10^{-4}$
			10^{-3} + single	15.91	10.8	6.81	0.09	$2.3 \cdot 10^{-4}$
		jump-410	Full Rank	13.74	100.0	19.58	0.20	$4.8 \cdot 10^{-18}$
			10^{-3}	16.03	10.7	9.95	0.10	$2.1 \cdot 10^{-4}$
			10^{-3} + single	15.86	10.5	6.62	0.09	$7.5 \cdot 10^{-5}$
225	$1.94 \cdot 10^6$	iso-viscous	Full Rank	41.10	100.0	139.17	0.55	$7.9 \cdot 10^{-19}$
			10^{-5}	47.24	12.9	35.51	0.28	$4.4 \cdot 10^{-7}$
			10^{-5} + single	47.31	12.9	25.17	0.26	$4.8 \cdot 10^{-7}$
			10^{-3}	47.27	7.7	30.97	0.26	$2.1 \cdot 10^{-4}$
			10^{-3} + single	47.40	7.7	21.07	0.20	$2.1 \cdot 10^{-4}$
		jump-410	Full Rank	41.02	100.0	134.61	0.56	$1.5 \cdot 10^{-18}$
			10^{-5}	47.56	13.0	36.98	0.30	$2.4 \cdot 10^{-6}$
			10^{-5} + single	47.65	13.2	25.63	0.27	$1.4 \cdot 10^{-6}$
			10^{-3}	47.55	7.5	31.11	0.24	$5.0 \cdot 10^{-5}$
			10^{-3} + single	47.62	7.6	21.16	0.19	$4.7 \cdot 10^{-5}$

First, as expected, we observe that the average run-time for the processing of the fine grids is very similar to those we observe when using PMINRES. There are still small variations, with a maximum of 3%, even between runs with different BLR ε parameters. These variations are quite usual at this extreme scale as some unstabilities can appear which are partly due to differences from the placement of the processes at runtime as well as asynchrone MPI communication, and differences in the use of the cache between two runs [34].

The coarse grid solve stays below 1s that is less than 0.1% of the total run-time over the iterations. Only the portion of the analysis and factorization step should be seen critical. As stated above, these steps may become problematic in a problem for dynamic or non-linear settings, but in our specific case these are in the range of the acceptable. The compute times decrease from $176.2\,s$ for the sparse direct solve to $79.3\,s$ with the BLR method in single precision on the biggest case. The run-time of the single precision BLR method is accelerated by a factor of 2.3 from the complexity reduction in comparison to the direct solve, while the number of iterations stays unchanged. For the largest test case, we had to reduce ε from 10^{-3} to 10^{-5} in the BLR method to obtain the same iteration number as in the Full Rank case. The value of the scaled residual is still comparable with the other problem sizes when using $\varepsilon = 10^{-3}$. This justifies that an accuracy around 10^{-5} for the coarsest grid problem is the bare minimum needed for the considered class of problems. Comparing the coarse grid timings with MUMPS, including the analysis, factorization and data transfer, to the ones of the PMINRES solver solver of Sect. 3.2, we observe a reduction of the total run-time of 50% from 162.5s to 83.6s, and a 6% points improvement of the overall parallel efficiency of the multigrid scheme with 43 200 processes.

This improvement is summarized in Fig. 3, where for each problem size and coarse grid solver, we plot the run-time ratios for fine, coarse, analysis+factorization, and data transfer. The white portion of the inner ring, corresponding to HHG with MUMPS on the coarse grid, reprensents the gain in run-time compared to using PMINRES, i.e. the outer ring. As expected, we observe that this gain increases with the size of the problem because the scalability of the proposed solution is better. Also, the run-times for coarse, i.e. MUMPS solve phase, and data transfer between HHG and MUMPS appear completely negligeable.

Table 4. Weak scaling of the V_{var}–cycle with a sparse direct and a block low-rank coarse level solver. The parallel efficiency compares the average total run-time of each run to the average total run-time of the smallest case with no BLR.

proc.	**DOFs**		**BLR ϵ**	**it**	**time (s)**					**par. eff.**	**scaled res.**
	fine	**coarse**			**total**	**fine**	**ana. & fac.**	**coarse**	**trans.**		
1 920	$5.37 \cdot 10^{9}$	$9.22 \cdot 10^{4}$	Full Rank	15	1169.0	1166.1	2.4	0.4	0.1	1.00	$1.9 \cdot 10^{-17}$
			10^{-3}	15	1179.0	1175.9	2.7	0.3	0.1	0.99	$3.4 \cdot 10^{-04}$
			10^{-3} + single	15	1139.0	1136.2	2.5	0.3	0.1	1.03	$1.5 \cdot 10^{-03}$
15 360	$4.29 \cdot 10^{10}$	$6.96 \cdot 10^{5}$	Full Rank	13	1120.0	1080.7	36.3	2.8	0.3	0.90	$3.1 \cdot 10^{-18}$
			10^{-3}	13	1117.9	1091.6	24.8	1.3	0.2	0.90	$1.4 \cdot 10^{-04}$
			10^{-3} + single	13	1091.0	1066.9	22.3	1.1	0.7	0.93	$2.4 \cdot 10^{-04}$
43 200	$1.21 \cdot 10^{11}$	$1.94 \cdot 10^{6}$	Full Rank	14	1382.0	1197.3	176.2	8.2	0.3	0.79	$1.0 \cdot 10^{-18}$
			10^{-5}	14	1297.0	1205.7	87.1	4.0	0.3	0.84	$3.5 \cdot 10^{-07}$
			10^{-5} + single	14	1282.0	1193.6	79.3	3.3	1.0	0.85	$3.6 \cdot 10^{-07}$
			10^{-3}	19	1755.0	1671.8	78.4	4.4	0.3	0.84	$1.4 \cdot 10^{-04}$

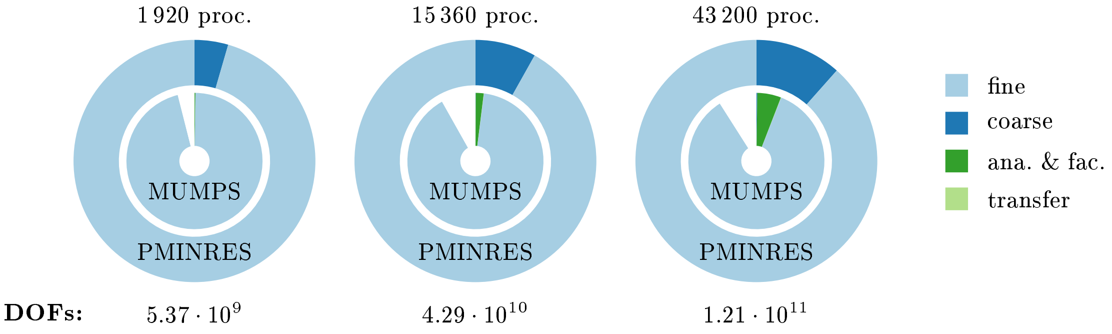

Fig. 3. Difference between an HHG run with PMINRES and MUMPS, using BLR ($\epsilon = 10^{-3}$) and single precision, as solvers on the coarse grid for the three different sizes of problem. from using MUMPs.

7 Conclusion

In this article, we studied the impact of advanced solution techniques on the coarsest level of a multigrid scheme. To increase the granularity of computations on the coarse grid, we discuss the use of an agglomeration technique such that the coarse level solver is executed on only a fraction of the processors of the fine grid. By doing so, we significantly reduce the time for communication and memory access, thus reducing the overall run-time of the coarse level solver. To increase the speed-up, we employ single precision arithmetic and the block low-rank approximation feature of the MUMPS parallel sparse direct solver that can lead to improved compute times at the cost of a reliably controlled loss of accuracy in the solution of the coarsest mesh. The efficiency of the solver is tested on the petascale supercomputer Hazel Hen on up to 43 200 processes and compared to Krylov space solvers for the coarsest grid level. For a 3D Stokes problem, the new solver can achieve a 6% points overall parallel efficiency improvement compared to a simple Krylov solver by reducing the run-time of the coarsest level solver. This scalability is achieved thanks to the fact that the factorization of the coarsest grid matrix need only be computed once and that the factorization can then be re-used in all later V-cycle iterations. This would no longer be the case in the context of dynamic or non-linear problems. A fine tuning of the MUMPS parameters, the use of hybrid shared/distributed memory parallelism in the sparse direct solver or more advanced agglomeration schemes might lead to a further acceleration of the setup phase. The study has shown that the new coarse grid solver is robust with respect to the viscosity variations for PDE problem, i.e. contrary to Krylov-type iterative methods it does not require longer compute times for larger viscosity jumps. To the best of our knowledge, these are the first results showcasing the potential of modern, approximate sparse direct solvers to accelerate the coarse grid solution of extreme scale MG solvers. This is an essential improvement e.g. for Earth Mantle simulation scenarios.

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 on Hazel Hen at the High Performance Computing Center Stuttgart (`www.hlrs.de`).

Conflict of interest The authors declare no potential conflict of interests.

References

1. P. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L'Excellent, and C. Weisbecker. Improving multifrontal methods by means of block low-rank representations. *SIAM J. Sci. Comput.*, 37(3):A1451–A1474, 2015.
2. 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(4):A1710–A1740, 2017.
3. P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary. Bridging the gap between flat and hierarchical low-rank matrix formats: The multilevel block low-rank format. *SIAM J. Sci. Comput.*, 41(3):A1414–A1442, 2019.

4. P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary. Performance and scalability of the block low-rank multifrontal factorization on multicore architectures. *ACM Trans. Math. Software*, 45(1):2:1–2:26, Feb. 2019.
5. P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Anal. Appl.*, 23(1):15–41, 2001.
6. A. Baker, R. Falgout, H. Gahvari, T. Gamblin, W. Gropp, T. V. Kolev, K. E. Jordan, M. Schulz, and U. M. Yang. Preparing algebraic multigrid for exascale. Technical Report LLNL-TR-533076, 2012.
7. S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. Gropp, et al. Petsc users manual. *See http://www.mcs.anl.gov/petsc*, 2019.
8. 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.
9. S. Bauer, M. Mohr, U. Rüde, J. Weismüller, M. Wittmann, and B. Wohlmuth. A two-scale approach for efficient on-the-fly operator assembly in massively parallel high performance multigrid codes. *Appl. Numer. Math.*, 122:14–38, 2017.
10. 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):1–28, 2003.
11. B. Bergen and F. Hülsemann. Hierarchical hybrid grids: Data structures and core algorithms for multigrid. *Numer. Linear Algebra Appl.*, 11:279–291, 2004.
12. A. Brandt and O. E. Livne. *Multigrid techniques: 1984 guide with applications to fluid dynamics*, volume 67. SIAM, 2011.
13. F. Brezzi and J. Douglas. Stabilized mixed methods for the Stokes problem. *Numer. Math.*, 53(1):225–235, 1988.
14. C. Burstedde, G. Stadler, L. Alisic, L. C. Wilcox, E. Tan, M. Gurnis, and O. Ghattas. Large-scale adaptive mantle convection simulation. *Geophys. J. Internat.*, 192(3):889–906, 2013.
15. D. R. Davies, S. Goes, J. Davies, B. 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.
16. D. Drzisga, L. John, U. Rüde, B. Wohlmuth, and W. Zulehner. On the analysis of block smoothers for saddle point problems. *SIAM J. Matrix Anal. Appl.*, 39(2):932–960, 2018.
17. H. C. Elman, D. J. Silvester, and A. J. Wathen. *Finite elements and fast iterative solvers: With applications in incompressible fluid dynamics.* Oxford University Press., Oxford, 2014.
18. R. D. Falgout and J. E. Jones. Multigrid on massively parallel architectures. In E. Dick, K. Riemslagh, and J. Vierendeels, editors, *Multigrid Methods VI*, pages 101–107. Springer Berlin Heidelberg, 2000.
19. 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.
20. B. Gmeiner, U. Rüde, H. Stengel, C. Waluga, and B. Wohlmuth. Towards textbook efficiency for parallel multigrid. *Numer. Math. Theory Methods Appl.*, 8(1):22–46, 2015.
21. W. Hackbusch. *Hierarchical matrices : algorithms and analysis*, volume 49 of *Springer series in computational mathematics*. Springer, Berlin, 2015.
22. N. J. Higham and T. Mary. Solving block low-rank linear systems by LU factorization is numerically stable. MIMS EPrint 2019.15, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, Sept. 2019.
23. 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. *Comput. Methods Appl. Mech. Eng.*, 59(1):85–99, 1986.
24. F. Hülsemann, B. Bergen, and U. Rüde. Hierarchical hybrid grids as basis for parallel numerical solution of PDE. In *European Conference on Parallel Processing*, pages 840–843. Springer, 2003.
25. F. Hülsemann, M. Kowarschik, M. Mohr, and U. Rüde. Parallel Geometric Multigrid. In A. M. Bruaset and A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, number 51 in Lecture Notes in Computational Science and Engineering, pages 165–208. Springer, 2005.

26. T. Mary. *Block Low-Rank multifrontal solvers: complexity, performance, and scalability.* PhD thesis, Université de Toulouse, 2017.
27. D. May, J. Brown, and L. L. Pourhiet. A scalable, matrix-free multigrid preconditioner for finite element discretizations of heterogeneous stokes flow. *Computer Methods in Applied Mechanics and Engineering*, 290:496 – 523, 2015.
28. 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, June 8-10, 2016*, page 5, 2016.
29. 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(4):1525–2027, 2008.
30. A. Reisner, L. Olson, and J. Moulton. Scaling structured multigrid to 500k+ cores through coarse-grid redistribution. *SIAM Journal on Scientific Computing*, 40(4):C581–C604, 2018.
31. S. Reiter, A. Vogel, I. Heppner, M. Rupp, and G. Wittum. A massively parallel geometric multigrid solver on hierarchically distributed grids. *Comput. Visual. Sci.*, 16(4):151–164, 2014.
32. J. Rudi, A. C. I. Malossi, T. Isaac, G. Stadler, M. Gurnis, P. W. J. Staar, Y. Ineichen, C. Bekas, A. Curioni, and O. Ghattas. An Extreme-scale Implicit Solver for Complex PDEs: Highly Heterogeneous Flow in Earth's Mantle. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 5:1–5:12, 2015.
33. 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(21):9270–9278, 2015.
34. D. Skinner and W. Kramer. Understanding the causes of performance variability in hpc workloads. In *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.*, pages 137–149. IEEE, 2005.
35. H. Sundar, G. Biros, C. Burstedde, J. Rudi, O. Ghattas, and G. Stadler. Parallel geometric-algebraic multigrid on unstructured forests of octrees. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–11, Nov. 2012.
36. W. Zulehner. Analysis of iterative methods for saddle point problems: A unified approach. *Math. Comput.*, 71(238):479–505, Apr. 2002.