




# A New Matrix-Free Approach for Large-Scale Geodynamic Simulations and its Performance

Simon Bauer<sup>1</sup>, Markus Huber<sup>2</sup>, Marcus Mohr<sup>1</sup>(✉) , Ulrich Rude<sup>3,4</sup>,  
and Barbara Wohlmuth<sup>2</sup>

<sup>1</sup> Department of Earth and Environmental Sciences,  
Ludwig-Maximilians-Universität München, Munich, Germany  
{simon.bauer,marcus.mohr}@lmu.de

<sup>2</sup> Institute for Numerical Mathematics (M2), Technische Universität München,  
Munich, Germany

<sup>3</sup> Department of Computer Science 10, FAU Erlangen-Nürnberg,  
Erlangen, Germany

<sup>4</sup> Parallel Algorithms Project, CERFACS, Toulouse, France

**Abstract.** We report on a two-scale approach for efficient matrix-free finite element simulations. The proposed method is based on surrogate element matrices constructed by low-order polynomial approximations. It is applied to a Stokes-type PDE system with variable viscosity as is a key component in mantle convection models. We set the ground for a rigorous performance analysis inspired by the concept of parallel textbook multigrid efficiency and study the weak scaling behavior on SuperMUC, a peta-scale supercomputer system. For a complex geodynamical model, we achieve a parallel efficiency of 95% on up to 47 250 compute cores. Our largest simulation uses a trillion ( $\mathcal{O}(10^{12})$ ) degrees of freedom for a global mesh resolution of 1.7 km.

**Keywords:** Two-scale PDE discretization  
Massively parallel multigrid · Matrix-free on-the-fly assembly  
Large scale geophysical application

## 1 Introduction

The surface of our planet is shaped by processes deep beneath our feet. Phenomena like earthquakes, plate tectonics, crustal evolution up to the geodynamo are governed by forces in the Earth's mantle that transport heat from the interior of our planet to the surface in a planetwide solid-state convection. For this reason, the study of the dynamics of the mantle is critical to our understanding of how the entire planet works.

There is a constant demand for ever more realistic models. In the case of mantle convection models (MCMs), this includes, e.g., compressible flow formulations, strongly non-linear rheologies, i.e., models in which the fluid viscosity

depends not only on pressure and temperature, but also on the flow velocity, the inclusion of phase transitions or the tracking of chemical composition. A discussion of current challenges is, e.g., given in [15]. Another trend is the growing use of MCMs to perform inverse computations via adjoint techniques in order to link uncertain geodynamic modeling parameters to geologic observables and, thus, improve our understanding of mantle processes, see e.g. [7]. These advanced models require efficient software frameworks that allow for high spatial resolutions and combine sophisticated numerical algorithms with excellent parallel efficiency on supercomputers to provide fast time-to-solution. See [11, 15, 21] for recent developments.

We will focus here on the most compute-intensive part of any MCM, which is the solution of the generalized Stokes problem, where  $\mathbf{f}$  represents the buoyancy forces,  $\mathbf{u}$  velocity,  $p$  pressure,  $T$  temperature and  $\nu(\mathbf{u}, T)$  is the viscosity of the mantle.

$$-\operatorname{div} \left[ \frac{1}{2} \nu (\nabla \mathbf{u} + (\nabla \mathbf{u})^\top) \right] + \nabla p = \mathbf{f}, \quad \operatorname{div} \mathbf{u} = 0. \quad (1)$$

Problem (1) needs to be solved repeatedly as part of the time-stepping and/or as part of a non-linear iteration, if  $\nu$  depends on  $\mathbf{u}$ . Note that in (1) we assume an incompressible fluid, as the best way to treat the compressibility of the mantle is an open question, [15], outside the scope of this contribution.

Most current global convection codes are based on finite element (FE) discretizations, cf. [8, 15, 21]. While traditional FE implementations are based on the assembly of a global system matrix, there is a trend to employ matrix-free techniques, [2, 4, 17, 19]. This is motivated by the fact that storing the global matrix increases the memory consumption by an order of magnitude or more even when sparse matrix formats are used. This limits the resolution and results in a much increased memory traffic when the sparse matrix must be re-read from memory repeatedly. Since the cost for data movement has become a limiting factor for all high performance supercomputer architectures both in terms of compute time and energy consumption, techniques for reducing memory footprint and traffic must receive increased attention in the design of modern numerical methods.

In this contribution, we report on the prototype of a new mantle convection framework that is implemented based on Hierarchical Hybrid Grids (HHG) [1, 4, 11, 14]. HHG employs an unstructured mesh for geometry resolution which is then refined in a regular fashion. The resulting mesh hierarchy is well suited to implement matrix-free geometric multigrid methods. Multigrid techniques play an important role in any large-scale Stokes solver, most commonly as preconditioner for the momentum operator in a Krylov solver, or as inner solver in a Schur complement approach. We employ a geometric Uzawa-type multigrid solver that treats the full Stokes system all-at-once [12]. We present a new approach that allows to assemble the resulting FE stencils in the case of curved geometries and variable viscosity on-the-fly as a core component of matrix-free multigrid solvers. It is based on a polynomial approximation of the local element matrices, extending our work in [2].

We will carry out a systematic performance analysis of our HHG-based implementation and investigate parallel performance with respect to run-time, memory consumption and parallel efficiency of this new numerical approach for a real-world geophysical application. It will be investigated and tuned on the SuperMUC peta-scale system of the Leibniz Supercomputing Center (LRZ).

## 2 Software Framework and Discretization

Here we consider the thick spherical shell  $\Omega = \{\mathbf{x} \in \mathbb{R}^3 : r_{\text{cmb}} < \|\mathbf{x}\|_2 < r_{\text{srf}}\}$ , where  $r_{\text{cmb}}$  and  $r_{\text{srf}}$  correspond to the inner and outer mantle boundary, and  $\|\cdot\|_2$  denotes the Euclidean norm of a vector. By taking the Earth radius as reference unit, we set  $r_{\text{cmb}} = 0.55$  and  $r_{\text{srf}} = 1$ . We discretize  $\Omega$  by an initial tetrahedral mesh  $\mathcal{T}_0$  using a standard icosahedral meshing approach for spherical shells, see e.g. [8]. From this we construct a family of semistructured meshes  $\mathcal{T} := \{\mathcal{T}_\ell, \ell = 0, \dots, L\}$  by uniform refinement up to level  $L \in \mathbb{N}_0$ . For the finite element discretization of the Stokes system (1), we employ standard conforming linear finite element spaces for velocity and pressure on  $\mathcal{T}$ . While this  $P^1$ - $P^1$  pairing is of computational interest, it is known to be unstable. We use the pressure stabilization Petrov-Galerkin (PSPG) method [6] as stabilization technique. Using standard nodal basis functions for the finite element spaces, we obtain on each level  $\ell$  of the hierarchy a linear system of algebraic equations

$$\mathcal{L}_\ell \begin{pmatrix} \mathbf{u}_\ell \\ \mathbf{p}_\ell \end{pmatrix} := \begin{pmatrix} A_\ell & G_\ell \\ D_\ell & -C_\ell \end{pmatrix} \begin{pmatrix} \mathbf{u}_\ell \\ \mathbf{p}_\ell \end{pmatrix} = \begin{pmatrix} \mathbf{f}_\ell \\ \mathbf{g}_\ell \end{pmatrix}, \quad \ell = 0, \dots, L, \quad (2)$$

where  $\mathbf{u}_\ell \in \mathbb{R}^{n_{\mathbf{u};\ell}}$  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  $n_{\mathbf{p};\ell}$ . For our considerations below, it is advantageous to re-write (2) by sorting the vector of unknowns with respect to the different types of degrees of freedom to expose the scalar building blocks of (2)

$$\mathcal{L}_\ell = \begin{pmatrix} A_\ell^{11} & A_\ell^{12} & A_\ell^{13} & G_\ell^1 \\ A_\ell^{21} & A_\ell^{22} & A_\ell^{23} & G_\ell^2 \\ A_\ell^{31} & A_\ell^{32} & A_\ell^{33} & G_\ell^3 \\ D_\ell^1 & D_\ell^2 & D_\ell^3 & -C_\ell \end{pmatrix}, \quad \begin{pmatrix} \mathbf{u}_\ell \\ \mathbf{p}_\ell \end{pmatrix} = \begin{pmatrix} \mathbf{u}_\ell^1 \\ \mathbf{u}_\ell^2 \\ \mathbf{u}_\ell^3 \\ \mathbf{p}_\ell \end{pmatrix}. \quad (3)$$

In this representation, the upper left  $3 \times 3$  substructure of blocks corresponds to  $A_\ell$  and is related to the divergence of the strain tensor in (1). The submatrix  $D_\ell$ , resulting from the discretization of the divergence operator in the continuity equation, has a  $1 \times 3$  block-structure, while  $G_\ell$ , coming from the pressure gradient in (1), has a  $3 \times 1$  block-structure and our discretization yields  $D_\ell = G_\ell^\top$ . The stabilization  $C_\ell$  term acts only on the pressure and, therefore, gives a  $1 \times 1$  block. It can be viewed as a discrete Laplacian operator acting on the pressure with Neumann boundary condition. Note that, while it is obvious that  $A_\ell$  depends on the viscosity  $\nu$ , it is also necessary to include  $\nu^{-1}$  in the stabilization  $C_\ell$ .

The mesh hierarchy  $\mathcal{T}$  allows to construct an efficient geometric all-at-once Uzawa multigrid method [12]. For solving the linear system (2), we apply multigrid V-cycles with three pre- and post-smoothing steps on level  $L$  and on each

coarser level two extra smoothing steps are added. Using a Uzawa type smoother then guarantees mesh-independent convergence, and we denote this type of multigrid as  $V_{\text{var}}(3, 3)$ . As the multigrid method acts both on velocity and pressure, the problem that needs to be solved on the bottom of the V-cycle is also of the form (2). For this, we employ the preconditioned minimal residual method (PMINRES). Our preconditioner has a block structure, where we apply a Jacobi preconditioned conjugate gradient method to the velocity part and perform a lumped mass matrix scaling on the pressure.

The HHG framework is a carefully designed and implemented high performance finite element multigrid software package [3, 12] which has already demonstrated its usability for geodynamical simulations [1, 22]. Conceptually, refinement of the input mesh  $\mathcal{T}_0$ , which we call *macro mesh*, generates new nodes on edges, faces and within the volume of the tetrahedra of the input mesh. In HHG, these nodal values are organized by their geometric classification into a system of container data-structures called *primitives*. The nodal values in the interior of each macro tetrahedron are stored in a volume primitive, and similarly the values on macro edges, faces and vertices in their respective primitives. In this way, each nodal value is uniquely assigned to one primitive. Note that, only starting with refinement level two, we get nodes to store in the volume primitives. We use  $\mathcal{T}_2$  as coarsest level in our multigrid solver. HHG's approach of splitting nodes between primitives of different geometric dimensionality naturally integrates with distributed-memory parallelism. Primitives are enriched by the nodal values of neighboring primitives in the form of ghost layer data-structures and kept up-to-date by MPI-communication in case of off-process dependencies, [3, 4].

The structured refinement of the input mesh, employed in HHG, results in the same types of tetrahedra being adjacent to each node within a certain primitive type and, thus, identical coupling patterns for these nodes. For constant  $\nu$  on each macro tetrahedron, the discretization results also in the weights of these coupling being constant when proceeding from one node of a primitive to the next. This allows to use a constant stencil for all nodes in each volume primitive in a matrix-free approach, resulting in a significantly improved performance of computationally-intensive matrix-vector multiplications. In view of the system matrix in (3), we can identify the non-zero entries of each row of each block by a stencil and denote it by

$$s_{ij}^{A;m,n} = (A_\ell^{mn})_{ij}, \quad s_{ij}^{D;m} = (D_\ell^m)_{ij}, \quad s_{ij}^{G;m} = (G_\ell^m)_{ij}, \quad s_{ij}^C = (C_\ell)_{ij},$$

for row index  $i$  and column index  $j$  and  $m, n \in \{1, 2, 3\}$ . Within each volume primitive each stencil reduces to 15 non-zero entries. In the following, we will denote a stencil weight by  $s_{ij}$ , if there is no ambiguity. The full 15pt stencil at node  $i$  will be written as  $s_{i,\cdot}$ .

### 3 Efficient On-the-Fly Stencil Assembly

While the hybrid approach of HHG exhibits superior performance, its geometry approximation on curved domains such as the spherical shell, is limited in

the sense that no refined nodes reside on the actual boundary. To account for this, in our implementation the fine grid nodes can be projected outwards onto the spherical surface. Also all interior nodes are projected to form concentric spherical layers. In a matrix-free framework, this comes at the cost that the FE stencils have to be repeatedly re-assembled on-the-fly.

We briefly describe the assembly procedure. For brevity, we show this only for  $A_\ell^{11}$  from (3); the other entries are computed analogously. For linear FE the stencil weight  $s_{ij}$  can be computed by

$$s_{ij} = \sum_{t \in \mathcal{N}(i,j)} J_t^{-\top} \nabla \hat{\phi}_{i_{\text{loc}}} \cdot J_t^{-\top} \nabla \hat{\phi}_{j_{\text{loc}}} |\det(J_t)| \int_t \nu \, d\mathbf{x} = \sum_{t \in \mathcal{N}(i,j)} E_{i_{\text{loc}}, j_{\text{loc}}}^t \bar{\nu}_t \quad (4)$$

where  $J_t$  is the Jacobian of the mapping from the reference element  $\hat{t}$ ,  $\mathcal{N}(i, j)$  the set of elements with common nodes  $i$  and  $j$ ,  $E^t \in \mathbb{R}^{4 \times 4}$  the local element matrix on  $t$ ,  $i_{\text{loc}}$  the element local index of the global node  $i$ , and  $\hat{\phi}_{i_{\text{loc}}}$  the associated shape function. We can use a vertex based quadrature rule for the integral over  $\nu$  by summing over the four vertices of  $t$  with weights  $1/4$ . This fits naturally to the HHG memory layout where the coefficients  $\nu_i$  are stored point-wise. Also techniques for elimination of common sub-expressions can be employed, see [14].

A traditional matrix-free implementation requires to repeatedly evaluate (4) on-the-fly. For the full 15pt stencil  $s_{i,:}$ , this involves the computation of  $E^t$  on each of the 24 elements adjacent to node  $i$ . Even though we use optimized code generated by the FEniCS Form Compiler [18] for this task, it constitutes the most expensive part in the stencil assembly procedure and severely reduces overall performance. We term this approach IFEM and it will serve as our baseline for comparison. We remark that our implementation is node- and not element-centric. A benefit of this is, e.g., that the central stencil weight, essential for point-smoothers, is directly available. A disadvantage is that it performs redundant operations as it does not take into account the fact that each element matrix is shared by four nodes. We could slightly reduce the operation count by computing only the  $i$ -th row of the matrix when dealing with node  $i$ . However, this still involves the Jacobian of the reference mapping which gives the largest contribution to the number of operations.

In order to recover the performance of the original HHG implementation also on curved domains we recently proposed an alternative approach in [2] for block-wise constant  $\nu$ . It replaces the expensive evaluation of (4) with approximating the values of  $s_{ij}$  by a low-order polynomial. The polynomial coefficients are computed via a least-squares fit in a setup phase and stored. Hence we denote the technique as LSQP. Later, whenever the stencil  $s_{i,:}$  is needed, one has to evaluate 15 polynomials at node  $i$ , one for each stencil weight. In [2] quadratic polynomials gave the best compromise between accuracy and runtime performance provided that the coarse scale mesh was fine enough. Furthermore, we showed that this approximation does not violate the optimal approximation order of the  $L^2$ -discretization error for linear finite elements, provided that the pairing of refinement depth  $L$  and macro mesh size  $H$  is selected carefully. Results for the Laplace operator [2, Table 4.1] indicated that for eight levels of refinement

the converted macro resolution of the spherical shell should be at least around 800 km. For the experiments carried out in Sect. 5, this is satisfied except for the smallest run, though even there we find good results, see Table 2.

For our PDE problem (2), we have to deal with two additional challenges. Firstly, instead of a scalar PDE operator as used in [2] we have a system of PDEs. Secondly, we have to incorporate the non-constant viscosity in the elliptic operators  $A_\ell$  and  $C_\ell$ . Conceptually, our discrete PDE system (3) consists of  $4 \times 4$  operator blocks coupling the three velocity components and the pressure. Our implementation allows to individually replace any of 16 suboperators by a LSQP approximation. Here, we only report on the most compute time saving approach, which is to replace all of the suboperators by the surrogates. We do this on all levels  $\mathcal{T}_\ell$ , apart from the coarsest one  $\ell = 2$ . We remark that the polynomials are evaluated at the nodal centers which leads to a small asymmetry in the operators. In [2] we found this relative asymmetry to be in  $\mathcal{O}(h)$ . This does not impact the algebraic convergence of the multigrid solver. However, it leads to a small issue on the coarsest level. There LSQP uses the same matrix  $\mathcal{L}_2$  as IFEM. That matrix is symmetric positive semi-definite with a trivial kernel. Due to the asymmetry in our LSQP approach the restricted residual can include contributions from that kernel, which we fix by a simple projection of the right-hand side onto  $\text{Im}(\mathcal{L}_2)$  to avoid problems with our PMINRES solver.

How to accommodate variable viscosity is a more intricate problem. In addition to the geometry variation, which can be approximated by quadratic polynomials as shown in [2], we also get variations due to the non-constant viscosity. If these are smooth enough, LSQP still yields good results. For more complex viscosity models, like in Sect. 5, with strong lateral variations a low order polynomial approximation may lead to poor results. Also in time-dependent and/or non-linear simulations where viscosity changes together with temperature and/or velocity, we would need to regularly recompute the polynomial coefficients. We, therefore, choose another approach. Recall that the most expensive part in (4) is the computation of the 24 element matrices. Instead of directly approximating  $s_{ij}$ , one can also approximate the contributions of  $E^t$  by quadratic polynomials. That is we substitute the expensive  $E_{i_{\text{loc}}, j_{\text{loc}}}^t$  by an inexpensive polynomial approximation  $\tilde{E}_{i_{\text{loc}}, j_{\text{loc}}}^t$  in (4). The polynomial approximation then solely depends on the geometry and is independent of the coefficients. Thus, it works for all kinds of coefficients. To distinguish between the two variants, we denote the original one as LSQP<sub>S</sub> and the new modified one as LSQP<sub>E</sub>. Note that due to the linearity of the least-squares fit w.r.t. the input data, LSQP<sub>E</sub> yields the same stencil weights as LSQP<sub>S</sub> in case of blockwise constant coefficients.

Each element matrix  $E^t$  contributes four values to one stencil  $s_{i,:}$ . Thus, in total the LSQP<sub>E</sub> version requires to define  $4 \cdot 24$  quadratic polynomials per macro element. For the full system (2) with general  $\nu$ , we approximate the stencils of  $A_\ell$  and  $C_\ell$  via LSQP<sub>E</sub>, while for  $G_\ell$  and  $G_\ell^\top$  the faster LSQP<sub>S</sub> version is used.

## 4 Towards a Rigorous Performance Analysis

The LSQP<sub>S</sub> approach was shown in [2] to be significantly faster than the traditional IFEM implementation. A more fundamental performance study must employ an absolute metric that does not rely on just quantifying the speed-up with respect to an arbitrary baseline implementation. To account for the real algorithmic efficiency and scalability of the implementation in relation to the relevant hardware limitations, we follow [14] where the notion of *textbook multigrid efficiency* [5] was extended to analyze massively parallel implementations. This metric is known as *parallel textbook multigrid efficiency* (parTME) and relies on detailed hardware performance models. While this goes beyond the scope of our current contribution, this section will provide first results and lay the foundation for further investigations.

The parTME metric is based on an architecture-aware characterization of a work unit (WU), where one WU is defined as one operator application of the full system. Here, we restrict ourselves to one scalar suboperator of (3). Conceptually, the extension to the full system is straightforward. The operator application can be expressed in terms of stencil based nodal updates  $u_i \leftarrow \sum_{j=1}^{15} s_{ij}u_j$ . The number of such updates performed per unit time is measured as *lattice updates per second* (Lup/s). This quantifies the primary performance capability of a given computer system with respect to a discretized system. A careful quantification of the Lup/s with an analytic white box performance model will often exhibit significant code optimization potential, as shown in [14]. Equally important, it provides absolute numbers of what performance can be expected from given hardware. This is crucial for a systematic performance engineering methodology. Our target micro-architecture is the eight-core Intel Sandy Bridge (SNB) Xeon E5-2680 processor with clock frequency 2.7 GHz as used in SuperMUC Phase 1. This processor delivers a peak performance of 21.6 double precision GFlops per core, and 172.8 GFlops per chip. However, this is under the assumptions that the code vectorizes perfectly for the Sandy Bridge AVX architecture, that the multiply-add instructions can be exploited optimally, and that no delays occur due to slow access to data in the different layers of the memory hierarchy.

We start with a classic cost count per update to derive an upper bound for the maximal achievable Lup/s. Here, we will compare the versions IFEM, LSQP<sub>S</sub> and LSQP<sub>E</sub> that are extensions of (CC) and (VC) for domains with curved boundaries.

First, we briefly recapitulate the cost for (CC) and (VC) and refer to [14] for details. On a blockwise regular mesh with constant coefficients, also the stencils are blockwise constant. Thus, for (CC) only one single 15pt stencil is required per block. This can be easily stored and loaded without overhead. Therefore, the cost for one stencil based update is 14 add/15 mult. For variable coefficients, the stencils have to be assembled on-the-fly. This requires the additional evaluation of (4). In the (VC) implementation, one can exploit the fact that on a polyhedral domain there exist only six different congruency classes of local elements. Thus, again per block its contributions to (4) can be pre-computed.

**Table 1.** Maximal and measured performance on one Intel SNB core

Kernel	Domain	Coefficients	Add/Mult	$p_{\text{core}}^{\text{max}}$	Measured
CC	Polyhedral	Blockwise constant	14/15	720 MLup/s	176 MLup/s
VC	Polyhedral	Variable	136/111	79.4 MLup/s	39.5 MLup/s
IFEM	Curved	Variable	1480/1911	5.7 MLup/s	0.7 MLup/s
LSQP <sub>S</sub>	Curved	Moderately variable	44/45	245 MLup/s	71.7 MLup/s
LSQP <sub>E</sub>	Curved	Variable	328/303	33.0 MLup/s	11.3 MLup/s

Now, we turn to curved domains. The LSQP<sub>S</sub> approach is the extension of (CC) with the additional cost of 15 evaluations of a quadratic polynomial, one for each stencil component. For the evaluation, we use the scheme described in [2] that allows to evaluate a quadratic polynomial with 2 multiply-add operations. We note that LSQP<sub>S</sub> can also be seen as an extension of (VC) for moderately variable coefficients. For problems with strongly variable coefficients, we propose either to use IFEM or the LSQP<sub>E</sub> approach. Different from (VC), the contributions of the 24 neighboring element matrices must be re-computed on-the-fly. For IFEM, we count 56 additions and 75 multiplications per element matrix. The advantage of LSQP<sub>E</sub> is obvious, since only 4 polynomial evaluations, one for each of the four contributions are required per element matrix. Again, this can be achieved with 8 multiply-add operations. In Table 1, we report the total number of operations for the different algorithms. Based on the operation count, the processor peak performance provides an upper limit on the achievable performance. In Table 1 we show these upper bounds as well as the measured values. For (CC) and (VC) the values are taken from [14]. For the measurements, we employed the Intel C/C++ Compiler 17.0 with flags `-O3 -march = native -xHost`.

Table 1 clearly shows that the peak rates are far from being obtained. For the simpler kernels (CC) and (VC), we carefully analyzed the performance discrepancy using the roofline and Execution-Cache-Memory models, see [14] and the references therein. Reasons why the peak rates are not achieved, are the limitations in bandwidth, but also bottlenecks that occur in the instruction stream and CPU-internal memory transfers between the cache layers. A full analysis for the advanced kernels is outside the scope of this contribution, but will be essential in the future to exhibit the possible optimization potential. But even the simple Flop count and the measured throughput values indicate the success of LSQP<sub>S</sub> and LSQP<sub>E</sub> in terms of reducing operation count as compared to a conventional implementation, such as IFEM. Similarly, the MLup/s show a substantial improvement. Both together, and the comparison with (CC) and (VC) indicate that there may be further room for improvement.

## 5 Accuracy and Weak Scaling Results

In this section, we analyze the accuracy and scaling behavior of our implementation for a geophysical application. Our largest simulation run will be with a global resolution of the Earth’s mantle of  $\sim 1.7$  km.



**System:** We run our simulations on SuperMUC Phase1, a TOP500 machine at the LRZ, Garching, Germany. It is an IBM iDataPlex DX360M4 system equipped with eight-core SNB processors, cf. Sect. 4. Per core around 1.5 GB of memory are available to applications. Two sockets or 16 cores form one compute node, and 512 nodes are grouped into one island. The nodes are connected via an Infiniband FDR10 network. In total, there are 147 456 cores distributed on 18 islands with a total peak performance of 3.2 PFlop/s. We used the Intel compiler with options as in Sect. 4 and the Intel 2017.0 MPI library.

**Setup:** The icosahedral meshing approach for the spherical shell does not allow for an arbitrary number of macro elements in the initial mesh and the smallest feasible number of macros would be 60 already. Also we are interested in the scaling behavior from typical to large scale scenarios. Thus, we perform experiments starting on one island and scaling up to eight islands. We try to get as close as possible to using the full number of nodes on each island, while keeping the tangential to radial aspect ratio of the macro elements close to 1:1.

Inside a node, we assign two macro elements to each MPI process running on a single core. As the memory consumption of our application is on average about 1.7 GB per core, we utilize only 12 of the 16 available cores per node. These 12 cores are equally distributed on the two sockets by setting `LMPI_PIN_PROCESSOR_LIST = 0-5,8-13`. A deep hierarchy with 8 levels of refinement is used. This yields problem sizes with  $1.3 \cdot 10^{11}$  DoFs on 5 580 cores (one island),  $2.7 \cdot 10^{11}$  DoFs on 12 000 cores (two islands),  $4.8 \cdot 10^{11}$  DoFs on 21 600 cores (four islands) and  $1.1 \cdot 10^{12}$  DoFs on 47 250 cores (eight islands).

**Geophysical Model:** In order to have a realistic Stokes-type problem (1) as it appears in applications, we consider the following model. On the top of the mantle we prescribe non-homogeneous Dirichlet boundary conditions, composed of a no-outflow component and tangential components given by present day plate velocity data from [20]. On the core-mantle boundary vanishing tangential shear stress resulting in a free-slip condition is enforced.

In terms of viscosity, we employ a similar model as used in [9]. The viscosity is the product of a smooth function depending on the temperature and the radial position and a discontinuous function reflecting a viscosity jump in radial direction due to an asthenospheric layer, a mechanically weak zone where the viscosity is several orders of magnitude smaller than in the lower mantle. The concrete thickness of the asthenosphere is unknown and subject to active research, see e.g. [22]. Here, we choose the model from [22] with a thickness of 660 km as this depth is one of two transition zones of seismic wave velocities. The viscosity model in non-dimensional form is given by

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

where  $d_a = 660/R$  with the Earth radius  $R = 6371$  (km). Finally, we used present day temperature and density fields to compute the buoyancy term  $\mathbf{f}$  and the viscosity, see [7].

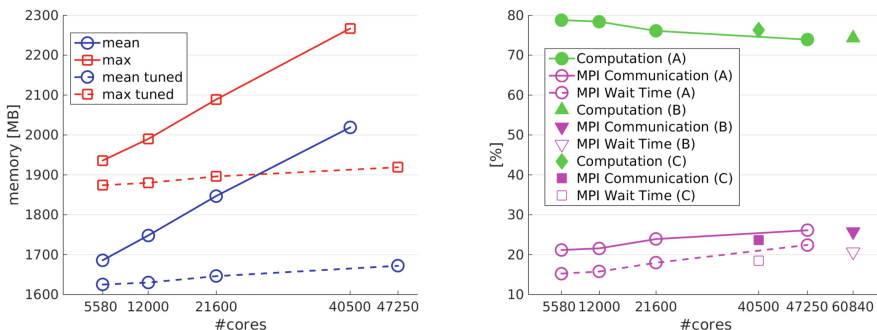
**Table 2.** Results for one island scenario with  $1.3 \cdot 10^{11}$  degrees of freedom: differences in the velocities inside the mantle obtained with IFEM and LSQP for different refinement levels (left); characteristic velocities in cm/a for level 8 (right).

level	discr. $L_2$	max-norm	charac. velocities	IFEM	LSQP	difference
4	$2.81 \cdot 10^{-4}$	$2.58 \cdot 10^{-2}$	avg. (whole mantle)	5.92	5.92	$5.60 \cdot 10^{-5}$
5	$4.05 \cdot 10^{-4}$	$4.84 \cdot 10^{-2}$	avg. (asthenosphere)	10.23	10.23	$1.10 \cdot 10^{-4}$
6	$5.19 \cdot 10^{-4}$	$6.70 \cdot 10^{-2}$	avg. (lower mantle)	4.48	4.48	$1.12 \cdot 10^{-4}$
7	$5.75 \cdot 10^{-4}$	$7.89 \cdot 10^{-2}$	max. (asthenosphere)	55.49	55.49	$2.61 \cdot 10^{-4}$
8	$6.83 \cdot 10^{-4}$	$8.58 \cdot 10^{-2}$	max. (lower mantle)	27.46	27.46	$6.33 \cdot 10^{-4}$

**Accuracy:** Before considering the run-time and scaling behavior of our new LSQP approach, we demonstrate its applicability by providing in Table 2 a comparison to results obtained with IFEM. We observe that the differences are sufficiently small in relation to typical mantle velocities and the uncertainties in the parameters that enter the model. The fact that the differences slightly grow with level reflects the two-scale nature of LSQP, as the finite element error decreases with mesh size  $h$  of the finest level, while the matrix approximation error is fixed by the mesh size  $H$  of the coarsest level, see also [2].

**Memory Consumption:** One important aspect in large scale simulations is memory consumption. Ideally, it should stay constant in weak scaling runs, as the number of DoFs per process remains the same. However, this is not always the case, especially in large scale simulations, due to buffer sizes that scale with the number of MPI ranks, see [10] for some examples.

To determine how strongly this affects our application, we measure the memory consumption per MPI process using the Intel MPI Performance Snapshot (mps) tool [16]. In Fig. 1 (left), we report the mean and maximum memory usage over all MPI processes. For each process, we assigned two volume primitives. The difference between the mean and maximum value comes from the different numbers of lower dimensional primitives attached to one process.



**Fig. 1.** Left: mean and max memory usage over all MPI processes. Right: percentage of computation versus communication (non-overlapping).

**Table 3.** Default and tuned Intel MPI DAPL settings ( $p$  = total no. of MPI processes.)

Environment variable	Default	Tuned
<code>I_MPI_DAPL_UD_SEND_BUFFER_NUM</code>	$16 + 4p$	8208
<code>I_MPI_DAPL_UD_RECV_BUFFER_NUM</code>	$16 + 4p$	8208
<code>I_MPI_DAPL_UD_ACK_SEND_POOL_SIZE</code>	256	8704
<code>I_MPI_DAPL_UD_ACK_RECV_POOL_SIZE</code>	$512 + 4p$	8704
<code>I_MPI_DAPL_UD_RNDV_EP_NUM</code>	4	2

For the default MPI buffer settings, we observe a significant linear increase in the memory usage caused by MPI. As a result the eight islands case runs out of memory. We therefore reduced the number of cores per node for this run to 10 resulting in configuration (B) (Table 4). Alternatively, one could decrease the number of MPI ranks for the same problem size and core count by using hybrid MPI/OpenMPI parallelism as done in [11]. This does, however, also not attack the root of the problem. For this, we need to deal with the MPI library instead.

On an Infiniband cluster the Intel MPI library uses the Shared Memory (SHM) transport mechanism for intra-node communication, while for inter-node communication it uses the Direct Access Programming Library (DAPL). While the UD (User Datagramm) version of DAPL is already much more memory conservative than the RC (Reliable Connection) version, the default buffer pool sizes still scale with the number of MPI processes, [10]. This can be seen from the default configuration values in Table 3. As suggested in [10], we set the internal DAPL UD buffer sizes to the fixed values given in Table 3, leading to a significant decrease of the memory consumption. The latter, now, shows almost perfect weak scalability and allows to go to extreme scales. Compared to the all-to-all communication scenarios shown in [10], we even see a much better scaling behavior up to 47 250 MPI ranks. We also do not notice any performance loss.

**Computation vs. Communication:** Current supercomputers provide tremendous computing capacities. This makes computations relatively cheap compared to communication that gets more expensive, the more processes are used. So, often communication is the bottleneck in high-performance codes.

To investigate the ratio of both, we again employ the Intel mps tool to measure the time for computation, i.e., mean time per process spent in the application code versus time for MPI communication. The latter is the time spent

**Table 4.** Configurations used in our experiments; default is to use configuration (A).

Configuration	Macro elements per core	Cores per node	# Cores (8 islands)	# DoFs (8 islands)
A	2	12	47 250	$1.1 \cdot 10^{12}$
B	2	10	40 500	$9.1 \cdot 10^{11}$
C	1	16	60 840	$6.8 \cdot 10^{11}$

inside the MPI library. This tool also reports the MPI imbalance, i.e., the mean unproductive wait time per process spent in the MPI library calls, when a process is waiting for data. This time is part of the reported MPI communication time. Here, a high percentage of computation is favorable, while the MPI imbalance should be small. Note that we do not overlap computation and communication. Using overlapping communication does not improve the performance significantly [13].

Besides our default configuration (A) and configuration (B), we consider a third case (C) for the eight islands run. Here, we increase the number of cores per node to the maximum of 16. This increases the total number of MPI processes to 60 840. To make this feasible, we assign one single macro element per rank. This can be seen as the most critical run in terms of communication as it involves the largest number of MPI processes.

The results are shown in Fig. 1 (right), where all initialization times are excluded. We find only a slight increase of communication during weak scaling. And even for the extreme cases the amount of communication is only about 25%. However, we also observe a relatively high MPI imbalance of around 20%. This is partly due to the imbalance of lower dimensional primitives and could be improved by a load balancing scheme that takes the cost of face primitives into account. Changing the number of macro elements per MPI process (C), or varying the number of cores per node (A, B) does hardly affect the results.

**Parallel Efficiency:** Finally, we report in Table 5 the time-to-solution. For these runs, we switch off any profiling. The iteration is stopped when the residual is reduced by  $10^5$  starting with a zero initial guess. For our geophysical application such a stopping criterion is more than sufficient. The high viscosity jump in our application makes the problem particularly difficult for the coarse grid (c.g.) solver. Choosing the right stopping criterion is essential for the Uzawa multigrid (UMG) convergence rate, while tuning it becomes quite tricky. It turned out that a criterion based on a maximal iteration count is favorable compared to a tolerance based criterion. In Table 5, we also report the best values we came up with. We remark that for the two islands case we could not find an acceptable number of c.g. iterations that reduced the UMG V-cycles below 10. For this run,

**Table 5.** Weak scaling results for geophysical application: Runtime w/ and w/o coarse grid solver (c.g.) and no. of UMG iterations. Values in brackets show no. of c.g. iterations (preconditioner/Minres). Parallel efficiency is shown for timings w/ and w/o c.g. \*Timings and parallel efficiency are scaled to 7 UMG iterations.

Islands	Cores	DoFs	Global resolution	UMG V-cycles	Time-to-solution	Time-to-sol. w/o c.g	Parallel efficiency
1	5 580	$1.3 \cdot 10^{11}$	3.4 km	7 (50/150)	1347 s	1151 s	1.00/1.00
2	12 000	$2.7 \cdot 10^{11}$	2.8 km	10* (100/150)	1493 s	1183 s	0.90/0.97
4	21 600	$4.8 \cdot 10^{11}$	2.3 km	7 (50/250)	1468 s	1201 s	0.92/0.96
8	47 250	$1.1 \cdot 10^{12}$	1.7 km	8* (50/350)	1609 s	1209 s	0.83/0.95

the element aspect ratio deviates most from 1:1. For all other simulations, the UMG iterations are stable around 7. Note that for the largest simulation the residual reduction was  $9.9 \cdot 10^4$  after 7 iterations, so the stopping criterion was only slightly missed. For a fair comparison of runtimes, we scaled all timings to 7 iterations. On up to eight islands, we find a parallel efficiency of 83%. Taking into account that it includes the c.g. solver with its non-optimal complexity, this is an excellent value. Examining the time-to-solution with the c.g. solver excluded, we find an almost perfect parallel efficiency on up to 47 250 cores of 95%. Compared to the IFEM reference implementation, we observe for the smallest run a speed-up of a factor larger than 20. In order to save core-h, and thus energy, we did not perform such a comparison for the larger scenarios.

## 6 Outlook

We extended our LSQP approach to systems of PDEs with variable coefficients and demonstrated that it is suitable for large scale geophysical applications. A systematic performance analysis demonstrates the new matrix-free techniques lead to substantial improvements compared to conventional implementations and they indicate that there is potential for further improvement. In future work, we will expand our study by detailed performance models for a rigorous performance classification and optimization.

**Acknowledgments.** 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 (GCS) for providing computing time on the supercomputer SuperMUC at LRZ. Special thanks go to the members of LRZ for the organization and their assistance at the “LRZ scaling workshop: Emergent applications”. Most scaling results were obtained during this workshop.

## References

1. Bauer, S., et al.: Hybrid parallel multigrid methods for geodynamical simulations. In: Bungartz, H.-J., Neumann, P., Nagel, W.E. (eds.) *Software for Exascale Computing - SPPEXA 2013–2015*. LNCSE, vol. 113, pp. 211–235. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-40528-5\\_10](https://doi.org/10.1007/978-3-319-40528-5_10)
2. Bauer, S., Mohr, M., Rüde, U., Weismüller, J., Wittmann, M., Wohlmuth, B.: 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)
3. Bergen, B., Gradl, T., Rüde, U., Hülsemann, F.: A massively parallel multigrid method for finite elements. *Comput. Sci. Eng.* **8**(6), 56–62 (2006)
4. Bergen, B., Hülsemann, F.: Hierarchical hybrid grids: data structures and core algorithms for multigrid. *Numer. Linear Algebra Appl.* **11**, 279–291 (2004)
5. Brandt, A.: *Barriers to achieving textbook multigrid efficiency (TME) in CFD*. Institute for Computer Applications in Science and Engineering, NASA Langley Research Center (1998)

6. Brezzi, F., Douglas, J.: Stabilized mixed methods for the Stokes problem. *Numer. Math.* **53**(1), 225–235 (1988)
7. Colli, L., Ghelichkhan, S., Bunge, H.P., Oeser, J.: Retrodictions of Mid Paleogene mantle flow and dynamic topography in the Atlantic region from compressible high resolution adjoint mantle convection models: sensitivity to deep mantle viscosity and tomographic input model. *Gondwana Res.* **53**, 252–272 (2018)
8. Davies, D.R., Davies, J.H., Bollada, P.C., Hassan, O., Morgan, K., Nithiarasu, P.: A hierarchical mesh refinement technique for global 3-D spherical mantle convection modelling. *Geosci. Model Dev.* **6**(4), 1095–1107 (2013)
9. Davies, D.R., Goes, S., Davies, J., Schuberth, B., Bunge, H.P., Ritsema, J.: Reconciling dynamic and seismic models of earth’s lower mantle: the dominant role of thermal heterogeneity. *Earth Planet. Sci. Lett.* **353–354**, 253–269 (2012)
10. Durnov, D., Steyer, M.: Intel MPI Memory Consumption. *The Parallel Universe* 21 (2015)
11. Gmeiner, B., Rüde, U., Stengel, H., Waluga, C., Wohlmuth, B.: Performance and scalability of hierarchical hybrid multigrid solvers for Stokes systems. *SIAM J. Sci. Comput.* **37**(2), C143–C168 (2015)
12. Gmeiner, B., Huber, M., John, L., Rüde, U., Wohlmuth, B.: A quantitative performance study for Stokes solvers at the extreme scale. *J. Comput. Sci.* **17**(Part 3), 509–521 (2016)
13. Gmeiner, B., Köstler, H., Stürmer, M., Rüde, U.: Parallel multigrid on hierarchical hybrid grids: a performance study on current high performance computing clusters. *Concurr. Comput.: Pract. Exp.* **26**(1), 217–240 (2014)
14. Gmeiner, B., Rüde, U., Stengel, H., Waluga, C., Wohlmuth, B.: Towards textbook efficiency for parallel multigrid. *Numer. Math. Theor. Meth. Appl.* **8**(01), 22–46 (2015)
15. Heister, T., Dannberg, J., Gassmüller, R., Bangerth, W.: High accuracy mantle convection simulation through modern numerical methods - II: realistic models and problems. *Geophys. J. Int.* **210**(2), 833–851 (2017)
16. Intel Corp.: MPI Performance Snapshot, version: 2017.0.4 (2017). <https://software.intel.com/en-us/node/701419>
17. Kronbichler, M., Kormann, K.: A generic interface for parallel cell-based finite element operator application. *Comput. Fluids* **63**, 135–147 (2012)
18. Logg, A., Ølgaard, K.B., Rognes, M.E., Wells, G.N.: FFC: the FEniCS form compiler. In: Logg, A., Mardal, K.A., Wells, G. (eds.) *Automated solution of differential equations by the finite element method*. LNCSE, vol. 84, pp. 227–238. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-23099-8\\_11](https://doi.org/10.1007/978-3-642-23099-8_11)
19. May, D.A., Brown, J., Pourhiet, L.L.: A scalable, matrix-free multigrid preconditioner for finite element discretizations of heterogeneous Stokes flow. *Comput. Methods Appl. Mech. Eng.* **290**, 496–523 (2015)
20. Müller, R.D., Sdrolias, M., Gaina, C., Roest, W.R.: Age, spreading rates, and spreading asymmetry of the world’s ocean crust. *Geochem. Geophys. Geosyst.* **9**(4), 1525–2027 (2008)
21. Rudi, J., Malossi, A.C.I., Isaac, T., Stadler, G., Gurnis, M., Staar, P.W.J., Ineichen, Y., Bekas, C., Curioni, A., Ghattas, O.: 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 2015*, pp. 5:1–5:12. ACM (2015)
22. Weismüller, J., Gmeiner, B., Ghelichkhan, S., Huber, M., John, L., Wohlmuth, B., Rüde, U., Bunge, H.P.: Fast asthenosphere motion in high-resolution global mantle flow models. *Geophys. Res. Lett.* **42**(18), 7429–7435 (2015). <https://doi.org/10.1002/2015GL063727>