Efficient Distributed Matrix-free Multigrid Methods on Locally Refined Meshes for FEM Computations

PETER MUNCH, Institute of Mathematics, University of Augsburg, Germany, Institute of Material Systems Modeling, Helmholtz-Zentrum Hereon, Germany, and Institute for Computational Mechanics, Technical University of Munich, Germany
TIMO HEISTER, Mathematical and Statistical Sciences, Clemson University, USA
LAURA PRIETO SAAVEDRA, Department of Chemical Engineering, Polytechnique Montréal, Canada
MARTIN KRONBICHLER, Institute of Mathematics, University of Augsburg and Department of Information Technology, Uppsala University, Sweden

This work studies three multigrid variants for matrix-free finite-element computations on locally refined meshes: geometric local smoothing, geometric global coarsening (both h-multigrid), and polynomial global coarsening (a variant of p-multigrid). We have integrated the algorithms into the same framework—the open source finite-element library `deal.II`—, which allows us to make fair comparisons regarding their implementation complexity, computational efficiency, and parallel scalability as well as to compare the measurements with theoretically derived performance metrics. Serial simulations and parallel weak and strong scaling on up to 147,456 CPU cores on 3,072 compute nodes are presented. The results obtained indicate that global-coarsening algorithms show a better parallel behavior for comparable smoothers due to the better load balance, particularly on the expensive fine levels. In the serial case, the costs of applying hanging-node

This work was supported by the Bayerisches Kompetenznetzwerk für Technisch-Wissenschaftliches Hoch- und Höchstleistungsrechnen (KONWIHR) through the projects "Performance tuning of high-order discontinuous Galerkin solvers for SuperMUC-NG" and "High-order matrix-free finite-element implementations with hybrid parallelization and improved data locality". The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC-NG at Leibniz Supercomputing Centre (LRZ, www.lrz.de) through project id pr83te.
Timo Heister was partially supported by the National Science Foundation (NSF) Award DMS-2028346, OAC-2015848, EAR-1925575, by the Computational Infrastructure in Geodynamics initiative (CIG), through the NSF under Award EAR-0949446 and EAR-1550901 and The University of California – Davis, and by Technical Data Analysis, Inc. through US Navy STTR Contract N68335-18-C-0011. Clemson University is acknowledged for generous allotment of compute time on Palmetto cluster.
This research is part of the Frontera computing project at the Texas Advanced Computing Center. Frontera is made possible by National Science Foundation award OAC-1818253.
Authors' addresses: P. Munch, Institute of Mathematics, University of Augsburg, Universitätsstraße 12a, 86159 Augsburg, Germany; Institute of Material Systems Modeling, Helmholtz-Zentrum Hereon, Max-Planck-Straße 1, 21502 Geesthacht, Germany; Institute for Computational Mechanics, Technical University of Munich, Boltzmannstraße 15, 85748 Garching b. München, Germany; email: peter.muench@uni-a.de; T. Heister, Mathematical and Statistical Sciences, Clemson University, O-110 Martin Hall, Clemson, South Carolina, USA; email: heister@clemson.edu; L. Prieto Saavedra, Department of Chemical Engineering, Polytechnique Montréal, P. O. Box 6079, Stn Centre-Ville, H3C 3A7, Montreal, QC, Canada; email: laura.prieto-saavedra@polymtl.ca; M. Kronbichler, Institute of Mathematics, University of Augsburg, Universitätsstraße 12a, 86159 Augsburg, Germany and Department of Information Technology, Uppsala University, Box 337, 75105 Uppsala, Sweden; email: martin.kronbichler@uni-a.de.

2329-4949/2023/03-ART3 $15.00
https://doi.org/10.1145/3580314

constraints might be significant, leading to advantages of local smoothing, even though the number of solver iterations needed is slightly higher. When using p- and h-multigrid in sequence (hp-multigrid), the results indicate that it makes sense to decrease the degree of the elements first from a performance point of view due to the cheaper transfer.

CCS Concepts: • **Mathematics of computing → Solvers**; **Mathematical software performance**; • **Computer systems organization → Multicore architectures;**

Additional Key Words and Phrases: Multigrid, finite-element computations, linear solvers, matrix-free methods

ACM Reference format:
Peter Munch, Timo Heister, Laura Prieto Saavedra, and Martin Kronbichler. 2023. Efficient Distributed Matrix-free Multigrid Methods on Locally Refined Meshes for FEM Computations. *ACM Trans. Parallel Comput.* 10, 1, Article 3 (March 2023), 38 pages.
https://doi.org/10.1145/3580314

1 INTRODUCTION

Many solvers for **finite-element methods (FEM)** rely on efficient solution methods for second-order **partial differential equations (PDEs)**, e.g., for the Poisson equation:

$$-\Delta u = f, \tag{1}$$

where u is the solution variable and f is the source term. Poisson-like problems also frequently occur as subproblems, e.g., in computational fluid dynamics [10, 30, 59] or in computational plasma physics [80]. Efficient realizations often rely on adaptively refined meshes to resolve geometries or features in the solution.

Multigrid methods are among the most competitive solvers for such problems [36]. The three basic steps of a two-level algorithm are (1) presmoothing, in which the high-frequency error components in the initial guess are removed with a *smoother*; (2) coarse-grid correction, in which a related problem on a coarser grid is solved, requiring *intergrid transfer operators* and a *coarse-grid solver*; and (3) postsmoothing, in which the high-frequency error components introduced during interpolation are removed. Nesting two-level algorithms recursively gives a multigrid algorithm. In library implementations, these steps are generally expressed as operators. The operations can be generally chosen and/or configured by the user and strongly depend on the multigrid variant selected. This publication discusses massively parallel multigrid variants for locally refined meshes, including the efficient implementation of their operators, compares them with each other, and gives recommendations regarding optimal configurations.

1.1 Multigrid Variants

Which multigrid approach to choose depends on the way the mesh is generated and on the underlying finite-element space. If the mesh is generated by globally refining each cell of a coarse grid recursively, then it is a natural choice to apply geometric multigrid (abbreviated here as h-multigrid), which uses the levels of the resulting mesh hierarchy as multigrid levels. Alternatively, in the context of high-order finite elements, it is possible to create levels by reducing the polynomial order of the shape functions p of the elements, while keeping the mesh the same, as done by polynomial multigrid (abbreviated as p-multigrid). For a very fine, unstructured mesh with low-order elements, it is not as trivial to explicitly construct enough multigrid levels and one might need to fall back to non-nested multilevel algorithms [1, 22, 23] or algebraic multigrid (AMG; see the review by Stüben [97]). These basic multigrid strategies can be nested in hybrid multigrid

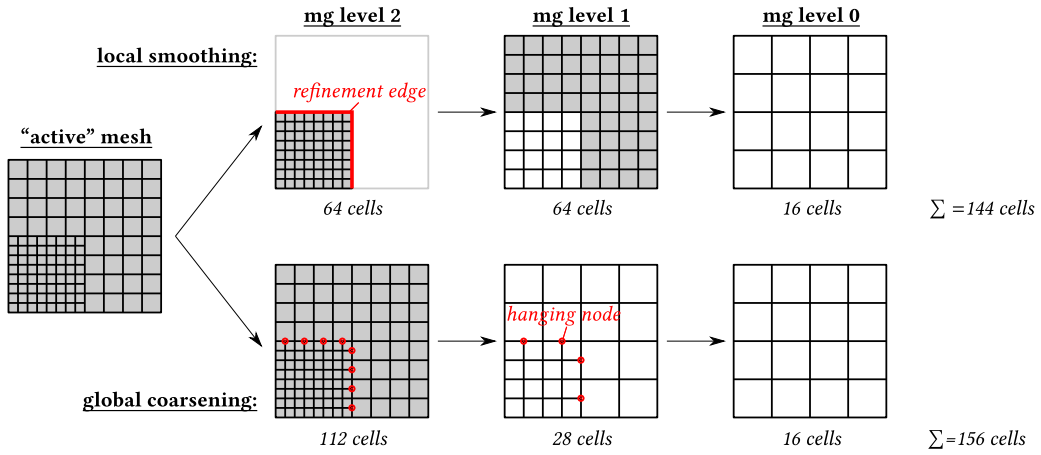


Fig. 1. Visual comparison of geometric multigrid methods for locally refined meshes. Top: (Geometric) local smoothing. Bottom: (Geometric) global coarsening. Local smoothing only considers cells strictly on the same refinement level. This typically introduces an internal boundary (at the refinement edge) when the cells do not cover the whole computational domain. Only if they do (here for level 1 and 0, not for level 2), one can switch to a coarse-grid solver. Instead, global coarsening considers the whole domain and typically introduces hanging nodes on the multigrid levels. Global coarsening tends to involve work on more cells in total compared to local-smoothing algorithms but often reduces the number of cells per multigrid level quicker on the finer levels. The gray shading indicates active cells.

solvers [32, 71, 84, 85, 89, 93, 98] and, in doing so, one can exploit the advantages of all of them regarding robustness. Most common are *hp*-multigrid, which combines *h*- and *p*-multigrid, and AMG as black-box coarse-grid solver of geometric or polynomial multigrid solvers.

All the above-mentioned multigrid variants are applicable to locally refined meshes. However, local refinement comes with additional options (local vs. global definition of the multigrid levels, resulting in local(-smoothing) and global(-coarsening) versions of geometric/polynomial multigrid) and with additional challenges, which are particularly connected with the presence of hanging-node constraints, as indicated in Figure 1.

1.2 Related Work

Some authors of this study have been involved in previous contributions to the research field of multigrid methods. Their implementations are used and extended in this work.

In References [27, 28, 64, 66], matrix-free and parallel implementations of geometric local-smoothing algorithms from the `deal.II` finite-element library were investigated and compared with AMG, demonstrating the benefits of using matrix-free implementations for both CPU and GPU but also non-optimal scalability due to load imbalance for simple partitioning strategies. Local-smoothing algorithms have been a core module of `deal.II` for many years; however, they gained much maturity, flexibility, and performance due to the above-mentioned thorough studies. In Reference [66], their performance was compared to a state-of-the-art implementation of a Poisson solver from the literature [36]: A speedup of 30% for quadratic elements and a speedup of a factor of 8 for fourth-degree elements on comparable hardware underline the high node-level performance of the code.

Independently of local smoothing, an efficient hybrid multigrid solver for discontinuous **Galerkin methods (DG)** for globally refined meshes was developed and presented in

Reference [32]. It is part of the `deal.II`-based incompressible Navier–Stokes solver `ExaDG` [10] and relies on auxiliary-space approximation [5], i.e., on the transfer into a continuous space, as well as on subsequent execution of p-multigrid, h-multigrid, and AMG. That solver was extended to leverage locally refined meshes, using a geometric/polynomial global-coarsening algorithm after the transfer into the continuous space, to simulate the flow through a lung geometry [60]. Its functionalities have been generalized and are the foundations of the new global-coarsening implementation in `deal.II` [7, 9], targeting both h- and p-multigrid, in addition to the established geometric local-smoothing implementation.

1.3 Our Contribution

In this publication, we consider three well-known multigrid algorithms for locally refined meshes for continuous higher-order matrix-free FEM: geometric local smoothing, geometric global coarsening (both h-multigrid), and polynomial global coarsening (a variant of p-multigrid). We have implemented them into the same framework, which allows us to compare their implementation complexity and performance for a large variety of problem sizes. This has not been done in an extensive way in the literature, often using only one of them [28, 99]. Furthermore, we rely on matrix-free operator evaluation, which provides optimal, state-of-the-art implementations in terms of node-level performance on modern hardware [66] and hence exercise the methods in a challenging context in terms of communication costs and workload differences.

The algorithms presented in this publication have been integrated into the open source finite-element library `deal.II` [8] and are part of its 9.4 release [9]. All experimental results have been obtained with small benchmark programs leveraging on the infrastructure of `deal.II`. The programs are available on GitHub under https://github.com/peterrum/dealii-multigrid.

The results obtained in this publication for continuous FEM are transferable to the DG case, where one does not have to consider hanging-node constraints but fluxes between differently refined cells. In the case of auxiliary-space approximation [60], this difference only involves the finest level and the rest of the multigrid algorithm could be as described in this publication.

The remainder of this work is organized as follows. In Section 2, we give a short overview of multigrid variants applicable to locally refined meshes. Section 3 presents implementation details of our solver, and Section 4 discusses relevant performance metrics. Sections 5 and 6 demonstrate performance results for geometric multigrid and polynomial multigrid, and, in Section 7, the solver is applied to a challenging Stokes problem. Finally, Section 8 summarizes our conclusions and points to further research directions.

2 MULTIGRID METHODS FOR LOCALLY REFINED MESHES

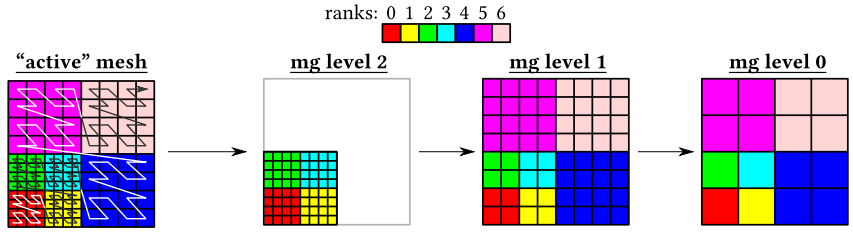
Algorithm 1 presents the basic multigrid algorithm to solve an equation system of the form $\boldsymbol{Ax} = \boldsymbol{b}$ arising from the FEM discretization of (1) ($\boldsymbol{A}$ is the system matrix, $\boldsymbol{b}$ is the right-hand-side vector containing the source term f and the boundary conditions, and $\boldsymbol{x}$ is the solution vector). In the first step, data are transferred to the multigrid levels, after which a multigrid cycle (in this study, a V-cycle with the steps presmoothing, computation of the residual, restriction, solution on the coarser grid, prolongation, and postsmoothing) is performed. Then, the result is copied back from the multigrid levels. The steps to copy data from and to the multigrid levels are not strictly needed in all cases; however, these steps are required by local smoothing and can be used to switch from double to single precision to reduce the costs of multigrid if it is used as a mixed-precision preconditioner [64]. The multigrid algorithm is complemented with the algorithms of a smoother (e.g., Chebyshev smoother [2]) and of a coarse-grid solver once the recursion is terminated. Instead of using multigrid as a solver, we choose to precondition a conjugate-gradient solver [45] with one multigrid cycle per iteration as this is often more robust. These algorithms are not presented here.

ALGORITHM 1: Multigrid V-cycle called recursively on each level l to solve $\boldsymbol{Ax} = \boldsymbol{b}$. It operates on vectors $[\boldsymbol{b}^{(0)}, \ldots, \boldsymbol{b}^{(L)}]$ and $[\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(L)}]$, which are filled/read on the finest level L, and uses the level operators $\boldsymbol{A}^{(l)}$, smoothers, intergrid operators, and coarse-grid solvers. In the case of local smoothing, we distinguish between interior DoFs (not labeled specially) and DoFs on the internal boundaries ($\boldsymbol{x}_E^{(l)}$) as well as decompose the level operator $\boldsymbol{A}^{(l)}$ into $\boldsymbol{A}_{SS}^{(l)}$, $\boldsymbol{A}_{SE}^{(l)}$, $\boldsymbol{A}_{ES}^{(l)}$, and $\boldsymbol{A}_{EE}^{(l)}$ (see the explanation in Section 2.1). For global coarsening, $\boldsymbol{A}^{(l)} = \boldsymbol{A}_{SS}^{(l)}$. The arrows in braces on the right indicate the type of communication at each step: (⇳) involves vertical communication between levels; (⇔) involves horizontal communication on the same level. Additional steps needed by local smoothing are highlighted in gray.

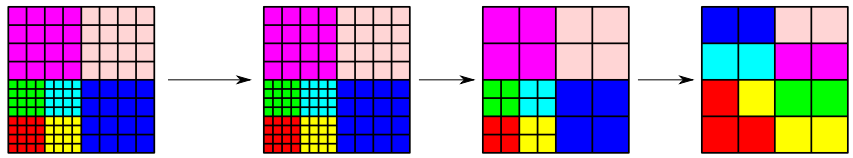
if $l = L$ **then**
  $[\boldsymbol{b}^{(0)}, \ldots, \boldsymbol{b}^{(L)}] \leftarrow \boldsymbol{b}$; /* copy to multigrid level(s) */
if $l = 0$ **then**
  $\boldsymbol{x}^{(0)} \leftarrow \texttt{CoarseGridSolver}(\boldsymbol{A}^{(0)}, \boldsymbol{b}^{(0)})$; /* coarse-grid solver: $\boldsymbol{A}^{(0)} \overset{!}{=} \boldsymbol{A}_{SS}^{(0)}$ (⇔/⇳) */
else
  $\boldsymbol{x}^{(l)} \leftarrow \texttt{Smoother}(\boldsymbol{A}_{SS}^{(l)}, \boldsymbol{0}, 0, \boldsymbol{b}^{(l)})$; /* presmoothing with $\boldsymbol{x}_E^{(l)} = \boldsymbol{0}$ (⇔) */
  $\left(\boldsymbol{r}^{(l)}, r_E^{(l)}\right) \leftarrow \left(\boldsymbol{b}^{(l)} - \boldsymbol{A}_{SS}^{(l)}\boldsymbol{x}^{(l)}, -\boldsymbol{A}_{ES}^{(l)}\boldsymbol{x}^{(l)}\right)$; /* compute residual (⇔) */
  $\boldsymbol{b}^{(l-1)} \leftarrow \boldsymbol{b}^{(l-1)} + \texttt{Restrictor}\left(\boldsymbol{r}^{(l)}, r_E^{(l)}\right)$; /* restrict residual (⇳) */
  $\texttt{VCycleLevel}(l-1)$; /* recursion */
  $\left(\boldsymbol{x}^{(l)}, x_E^{(l)}\right) \leftarrow \boldsymbol{x}^{(l)} + \texttt{Prolongator}\left(\boldsymbol{x}^{(l-1)}\right)$; /* prolongation (⇳) */
  $\boldsymbol{x}^{(l)} \leftarrow \texttt{Smoother}(\boldsymbol{A}_{SS}^{(l)}, \boldsymbol{x}^{(l)}, x_E^{(l)}, \boldsymbol{b}^{(l)})$; /* postsmoothing with $\boldsymbol{x}_E^{(l)} \neq \boldsymbol{0}$ (⇔) */
if $l = L$ **then**
  $\boldsymbol{x} \leftarrow [\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(L)}]$; /* copy from multigrid level(s) */

The parallelization of multigrid, e.g., based on domain decomposition (see Figure 2), involves the parallelization of each step of Algorithm 1. Here, two types of communication patterns arise: (1) *horizontal communication* on the same multigrid level during smoothing and residual evaluation, which involves point-to-point ghost-value data exchange between neighboring subdomains, and (2) *vertical communication* between multigrid levels during intergrid transfer. In the best case, vertical communication is also a point-to-point communication of some "lower-dimensional" (surface-induced) share of a vector, but it can imply the transfer of essentially the full vector if the levels are partitioned independently and/or the numbers of processes on the levels are reduced extremely (e.g., to one). The communication pattern within the coarse-grid solver depends on the solver chosen and often also involves vertical and horizontal communication. Global reductions are also common on the coarse grid. We note that the outer (conjugate-gradient) solver also involves global reductions to compute, e.g., the residual norm once per iteration, needed to determine termination. However, these global reductions are generally negligible due to the dominating costs of multigrid.

The various multigrid algorithms for locally refined meshes differ in the construction of the levels and the concrete details in the implementation of the multigrid steps. We consider two types of geometric multigrid methods: geometric local smoothing in Section 2.1 and geometric global coarsening in Section 2.2. Figure 1 gives a visual comparison of them and points out the issues resulting from the local or global definition of the levels, which will be discussed extensively in the following. Furthermore, we will detail polynomial global coarsening in Section 2.3. In the case of these multigrid variants, the level operator $\boldsymbol{A}^{(c)}$ can be obtained either recursively via the Galerkin operator $\boldsymbol{A}^{(c)} := \boldsymbol{R}^{(c,f)}\boldsymbol{A}^{(f)}\boldsymbol{P}^{(f,c)}$ with restriction matrix $\boldsymbol{R}^{(c,f)}$ and prolongation matrix



(a) First-child policy in the context of (geometric) local smoothing.



(b) Repartitioning of each level in the context of (geometric) global coarsening.

Fig. 2. Visual comparison of two possible strategies for partitioning of the multigrid levels of the mesh shown in Figure 1 for seven processes. Colors indicate the ranks of the subdomains. The active level is partitioned uniformly along a space-filling curve, according to a sum of weights for each cell. If a color does not appear (e.g., ranks 4–6 on level 2 in (a)), then it means that a process does not have any work on that level, leading to load imbalance during smoothing. If subdomains with the same color do not overlap (e.g., ranks 1–5 on level 0 and 1 in (b)), then this leads to more expensive intergrid transfer operators.

$\boldsymbol{P}^{(f,c)}$ constructed geometrically or by rediscretization. While rediscretization is not optimal in all cases, it is sufficient for the present experiments, allowing us to benefit from the fact that level operators are independent of each other as, for matrix-free computations, one wants to construct neither $\boldsymbol{A}^{(c)}/\boldsymbol{A}^{(f)}$ nor $\boldsymbol{R}^{(c,f)}/\boldsymbol{P}^{(f,c)}$ explicitly.

AMG can be used for the solution on locally refined meshes as well. Since the levels are constructed recursively via the Galerkin operator with the restriction and prolongation matrices constructed algebraically, no distinction regarding the local or global definition of the levels is possible. Since we use AMG in the following only as a coarse-grid solver, we refer to the literature for more details: Clevenger et al. [28] present a scaling comparison between AMG and a matrix-free version of local smoothing for a Laplace problem with Q_2 basis functions, showing a clear advantage of matrix-free multigrid methods on modern computing systems.

2.1 Geometric Local Smoothing

Geometric local-smoothing algorithms [19, 24, 28, 51, 53, 54, 70, 77, 90, 96] use the refinement hierarchy also for multigrid levels and perform smoothing refinement level by refinement level: Cells of less refined parts of the mesh are skipped (see the top part of Figure 1) so that hanging-node constraints do not need to be considered during smoothing. The authors of References [3, 20, 50, 77, 100, 102] investigate a version of local smoothing in which smoothing is also performed on a halo of a single coarse cell, which includes hanging-node constraints. We will not consider this form of local smoothing in the following.

The fact that domains on each level might not cover the whole computational domain results in multiple issues. Data need to be transferred in lines 2 and 13 in Algorithm 1 to and from *all* multigrid levels that are active, i.e., have cells that are not refined further. Moreover, internal interfaces (also known as "refinement edges") can appear with the need for special treatment. For details, interested readers are referred to References [51, 64]. In the following, we summarize key aspects relevant for our investigations.

For the purpose of explanation, let us split the **degrees of freedom (DoFs)** associated with the cells on an arbitrary level l into the interior ones $\boldsymbol{x}_S^{(l)}$ and the ones at the refinement edges $\boldsymbol{x}_E^{(l)}$, leading to the following block structure in the associated matrix system $\boldsymbol{A}^{(l)}\boldsymbol{x}^{(l)} = \boldsymbol{b}^{(l)}$:

$$\begin{pmatrix} \boldsymbol{A}_{SS}^{(l)} & \boldsymbol{A}_{SE}^{(l)} \\ \boldsymbol{A}_{ES}^{(l)} & \boldsymbol{A}_{EE}^{(l)} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_S^{(l)} \\ \boldsymbol{x}_E^{(l)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}_S^{(l)} \\ \boldsymbol{b}_E^{(l)} \end{pmatrix}.$$

For presmoothing on level l, homogeneous Dirichlet boundary conditions are applied at the refinement edges ($\boldsymbol{x}_E^{(l)} = \boldsymbol{0}$), allowing us to skip the coupling matrices. However, when switching to a coarser or finer level, the coupling matrices need to be considered. The residual to be restricted becomes the following:

$$\begin{pmatrix} \boldsymbol{r}_S^{(l)} \\ \boldsymbol{r}_E^{(l)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}_S^{(l)} \\ \boldsymbol{b}_E^{(l)} \end{pmatrix} - \begin{pmatrix} \boldsymbol{A}_{SS}^{(l)} & \boldsymbol{A}_{SE}^{(l)} \\ \boldsymbol{A}_{ES}^{(l)} & \boldsymbol{A}_{EE}^{(l)} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_S^{(l)} \\ \boldsymbol{x}_E^{(l)} \end{pmatrix} \overset{\boldsymbol{x}_E^{(l)}=\boldsymbol{0}}{=} \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{b}_E^{(l)} \end{pmatrix} + \underbrace{\begin{pmatrix} \boldsymbol{b}_S^{(l)} \\ \boldsymbol{0} \end{pmatrix} - \begin{pmatrix} \boldsymbol{A}_{SS}^{(l)} \\ \boldsymbol{A}_{ES}^{(l)} \end{pmatrix} \boldsymbol{x}_S^{(l)}}_{*}. \quad (2)$$

Since $\boldsymbol{b}_E^{(l)}$ has already been transferred to the coarser level by line 2 in Algorithm 1, one only has to restrict and add the result of term $*$ to the coarser level. During postsmoothing, an inhomogeneous Dirichlet boundary condition is applied with boundary values prescribed by the coarser level. This can be achieved, e.g., by modifying the right-hand side ($\boldsymbol{b}_S^{(l)} \leftarrow \boldsymbol{b}_S^{(l)} - \boldsymbol{A}_{SE}^{(l)}\boldsymbol{x}_E^{(l)}$).

A natural choice to partition the multigrid levels for local-smoothing algorithms is to partition the active level and let cells on lower refinement levels inherit the rank of their children. A simple variant of it is the "first-child policy" (see Figure 2(a) or Reference [28]): It recursively assigns the parent cell the rank of its first child cell. Since in adaptive FEM codes the parents of locally owned and ghost cells are generally already available on processes due to treelike data-structure storage of adaptively refined meshes [15, 26], no additional data structures need to be constructed and saved, but the storage of an additional flag (multigrid rank of the cell) is enough, leading to low memory consumption. Furthermore, intergrid transfer operations are potentially cheap as data are mainly transferred locally. A disadvantage—besides having to consider the edge constraints—is the potential load imbalance on the levels, as discussed in Reference [28]. This load imbalance could be alleviated by partitioning each level for itself. Such an alternative partitioning algorithm would lead to similar problems as in the case of global-coarsening algorithms (discussed next) and, as a result, the needed data structures would become more complex. This would counter the claimed simplicity of the data structures of this method; hence, we will consider geometric local smoothing only with "first-child policy" in this publication.

Furthermore, the fact that the transfer to and from the multigrid levels involves all active levels prevents an early switch to a coarse-grid solver on levels finer than those that are indeed not locally refined anymore, i.e., $\boldsymbol{A}_{SS}^{(l)} = \boldsymbol{A}^{(l)}$. However, the absence of hanging nodes allows the usage of smoothers that have been developed for uniformly refined meshes, e.g., patch smoothers [11, 55], which are superior for anisotropic meshes.

Since geometric local smoothing is the only local-smoothing approach we will consider here, we will call it simply—as common in the literature—*local smoothing* in the following.

2.2 Geometric Global Coarsening

Geometric global-coarsening algorithms [20, 21] coarsen all cells simultaneously, translating to meshes with hanging nodes also on coarser levels of the multigrid hierarchy (see the bottom part of Figure 1). The computational complexity—i.e., the total number of cells to be processed—is slightly

higher than in the case of local smoothing and might be non-optimal for some extreme examples of meshes [19, 54].

The fact that all levels cover the whole computational domain has the advantage that no internal interfaces need to be considered and the transfer to/from the multigrid levels becomes a simple copy operation to/from the finest level. However, hanging nodes have to be considered during the application of the smoothers on the levels. While this typically does not require new algorithms for basic smoothers as the infrastructure to support adaptive meshes can be simply applied to coarser representations, the operator evaluation and the applicable smoothers might become more expensive per cell by the need to resolve hanging-node constraints [81]. However, global-coarsening approaches show—for comparable smoothers—a better convergence behavior, which improves with the number of smoothing iterations [13, 14].

As the work on the levels generally increases compared to local smoothing, it is a valid option to repartition each level separately (see Figure 2(b)). On the one hand, this implies a higher pressure on the transfer operators, since they need to transfer data between independent meshes,[1] requiring potentially complex internal data structures, which describe the connectivities, and involved setup routines.[2] On the other hand, it opens the possibility to control the load balance between processes and the minimal granularity of work per process (by removing processes on the coarse level in a controlled way, allowing us to switch to subcommunicators [99]) as well as to apply a coarse-grid solver on any level. For the same reason, the construction of **full multigrid solvers (FMG)**, which visit the finest level only a few times, is easier, since any level can be used as a starting point for the recursion. Not selecting the actual coarsest grid but some more refined level allows us to relax the unfavorable complexity of FMG in terms of $O(L^2)$ by reducing the time spent on the coarse levels.

2.3 Polynomial Global Coarsening

Polynomial global-coarsening algorithms [12, 17, 18, 25, 29, 33, 35, 37, 38, 40, 41, 43, 44, 46, 48, 49, 52, 67, 72–76, 83, 85, 87, 88, 91, 94, 95, 99] are based on keeping the mesh size h constant on all levels but reducing the polynomial degree p of shape functions, e.g., to $p = 1$. Hence, the multigrid levels in this case have the same mesh but different polynomial orders. There are various strategies to reduce the order of the polynomial degree [32, 42]: The most common is the bisection strategy, which repeatedly halves the degree $p^{(c)} = \lfloor p^{(f)}/2 \rfloor$. This strategy reduces the number of DoFs in the case of a globally refined mesh similarly to the geometric multigrid strategies and provides a compromise between the number of V-cycles necessary to reduce the residual norm below a desired threshold and the cost of a single V-cycle [32].

The statements made in Section 2.2 about geometric global coarsening are also valid for polynomial global coarsening. However, the number of unknowns is reduced uniformly on each subdomain in contrast to geometric global coarsening, which obviates repartitioning of the level grids. This leads to a transfer operation that mainly works on locally owned DoFs.

In the following, we call geometric global coarsening simply *global coarsening* and polynomial global coarsening *polynomial coarsening*. As a reference, polynomial local smoothing arises, e.g., in the context of p- or hp-refinement, where only cells with the finest degree are smoothened [79].

[1] In `deal.II`, one needs to create a sequence of grids for global coarsening (each with its own hierarchical description). This is generally acceptable, since repartitioning of each level often leads to non-overlapping trees so that a single data structure containing all geometric multigrid levels would have little benefit for reducing memory consumption.

[2] Sundar et al. [99] present a two-step setup routine: The original fine mesh is coarsened and the resulting "surrogate mesh" is repartitioned. For space-filling-curve-based partitioning, this approach turns out to be highly efficient.

3 IMPLEMENTATION DETAILS

In this section, we detail efficient implementations of the multigrid ingredients for locally refined meshes used by Algorithm 1. We start with the handling of constraints. Then, we proceed with the matrix-free evaluation of operator $\boldsymbol{A}$, which is needed on the active and the multigrid levels, as well as with smoothers and coarse-grid solvers. This section is concluded with an algorithmic framework to efficiently realize all considered kinds of matrix-free transfer operators.

3.1 Handling Constraints

Constraints need to be considered—with slight differences—in the case of both local-smoothing and global-coarsening algorithms. First, we impose Dirichlet boundary conditions in a strong form and express them as constraints. Second, hanging-node constraints, which force the solution of the refined side to match the polynomial expansion on the coarse side, need to be considered to maintain H^1 regularity of the tentative solution [92]. In a general way, these constraints can be expressed as $x_i = \sum_j c_{ij} x_j + b_i$, where x_i is a constrained DoF, x_j a constraining DoF, c_{ij} the coefficient relating the DoFs, and b_i a real value, which can be used to consider inhomogeneities. We do not eliminate constraints but use a condensation approach [16, 101].

3.2 Matrix-free Operator Evaluation

Instead of assembling the sparse system matrix $\boldsymbol{A}$ and performing matrix-vector multiplications of the form $\boldsymbol{Ax}$, the matrix-free operator evaluation computes the underlying finite-element integrals to represent $\mathcal{A}(\boldsymbol{x})$. This is motivated by the fact that many iterative solvers and smoothers do not need the matrix $\boldsymbol{A}$ explicitly but only its action on a vector. On a high level, matrix-free operator evaluation can be derived in two steps: (1) loop switching $\boldsymbol{v} = \boldsymbol{Ax} = (\sum_e (\mathcal{G}_e^T \circ C_e^T) \boldsymbol{A}_e (C_e \circ \mathcal{G}_e)) \boldsymbol{x} = \sum_e \mathcal{G}_e^T \circ C_e^T (\boldsymbol{A}_e (C_e \circ \mathcal{G}_e \boldsymbol{x})) = \sum_e \mathcal{G}_e^T \circ C_e^T (\boldsymbol{A}_e \boldsymbol{x}_e)$, i.e., replacing the sequence "assembly–mat-vec" by a loop over cells with the three steps "gathering–application of the element stiffness matrix–scattering," and (2) exploitation of the structure of the element stiffness matrix $\boldsymbol{A}_e$. In this formula, the operator $C_e \circ \mathcal{G}_e$ represents the assembly of element-wise quantities into the global matrix and vectors, respectively [92], including the constraints C_e as discussed in Section 3.1. The element stiffness matrix $\boldsymbol{A}_e$ is constructed from three main ingredients in a finite element method, the shape functions and derivatives of the tentative solution tabulated at quadrature points, geometric factors and coefficients of the underlying differential operator at quadrature points, and the multiplication by the test functions and their derivatives as well as subsequent summation over quadrature points. The final structure of a matrix-free operator evaluation in the context of continuous finite elements in operator form is as follows:

$$\boldsymbol{v} = \mathcal{A}(\boldsymbol{x}) = \sum_e \mathcal{G}_e^T \circ C_e^T \circ \underbrace{\tilde{\mathcal{S}}_e^T \circ \mathcal{Q}_e \circ \mathcal{S}_e}_{\boldsymbol{A}_e} \circ C_e \circ \mathcal{G}_e \boldsymbol{x}. \tag{3}$$

This structure is depicted in Figure 3. For each cell e, cell-relevant values are gathered with operator $\mathcal{G}_e$, constraints are applied with C_e, and values, gradients, or Hessians associated to the vector $\boldsymbol{x}$ are computed at the quadrature points with $\mathcal{S}_e$. These quantities are processed by a quadrature-point operation $\mathcal{Q}_e$; the result is integrated and summed into the result vector $\boldsymbol{v}$ by applying $\tilde{\mathcal{S}}_e^T$, C_e^T, and $\mathcal{G}_e^T$. In this publication, we consider symmetric (self-adjoint) PDE operators with $\tilde{\mathcal{S}}_e = \mathcal{S}_e$.

In the literature, specialized implementations for GPUs [4, 57, 64, 68, 69] and CPUs [4, 62, 63, 80, 82] for operations as expressed in Equation (3) have been presented. For tensor-product (quadrilateral and hexahedral) elements, a technique known as sum factorization [78, 86] is often applied, which allows us to replace full interpolations from the local solution values to the quadrature points by a sequence of one-dimensional (1D) steps. In the context of CPUs, it is an option to

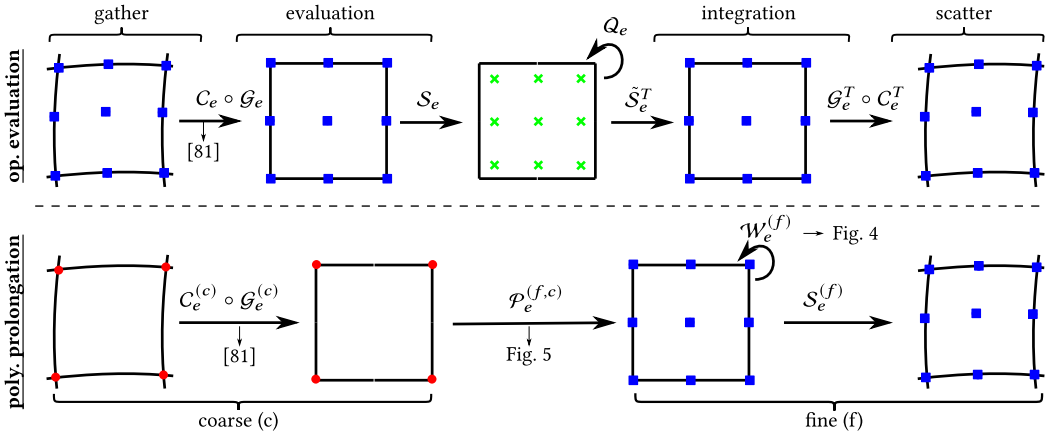


Fig. 3. Basic steps of a matrix-free operator evaluation according to Equation (3) and of a matrix-free polynomial prolongation according to Equation (8) for a single cell e.

vectorize across elements [62, 63], i.e., perform $(\mathcal{S}^T \circ \mathcal{Q} \circ \mathcal{S})_e$ for multiple cells in different lanes of the same instructions. This necessitates the data to be laid out in a struct-of-arrays fashion. The reshuffling of the data from array-of-structs format to struct-of-arrays format and back can be done, e.g., by $\mathcal{G}_e$, while looping through all elements [62]. Note that our implementation performs $C_e \circ \mathcal{G}_e \boldsymbol{x}$ for a single SIMD batch of cells at a time to keep intermediate results in caches and reduce global memory traffic. For the application of hanging-node constraints, we use the special-purpose algorithm presented in Reference [69, 81], which is based on updating the DoF map $\mathcal{G}_e$ and applying in-place sum factorization for the interpolation during the application of edge and face constraints. Even though this algorithm is highly optimized and comes with only a small overhead for high-order finite elements (<20% per cell with hanging nodes), the additional steps are not for free particularly for linear elements. This might lead to load imbalances in a parallel setting when some processes have a disproportionately high number of cells with hanging nodes, see the quantitative analysis in Reference [81].

Despite relying on matrix-free algorithms overall, some of the multigrid ingredients (e.g., smoother and coarse-grid solver) need an explicit representation of the linear operator in the form of a matrix or a part of it (e.g., its diagonal). Based on the matrix-free algorithm (3) applied to the local unit vector e_i, the local matrix can be computed column by column:

$$\boldsymbol{A}_e(:,i) = \mathcal{S}_e^T \circ \mathcal{Q}_e \circ \mathcal{S}_e \boldsymbol{e}_i. \tag{4}$$

The resulting element matrix can be assembled as usual, including the resolution of the constraints. Computing the diagonal is slightly more complex when attempting to immediately write into a vector for the diagonal without complete matrix assembly. In our code, we choose to compute the jth entry of the locally relevant diagonal contribution via

$$d_e(j) = \sum_i \left[C_e^T \boldsymbol{A}_e(:,i) \right] C_e(i,j) \quad \forall j \in \{ j \mid C_{e,ji} \neq 0 \}, \tag{5}$$

i.e., we apply the local constraint matrix $\boldsymbol{C}_e$ from the left to the ith column of the element matrix (computed via Equation (4)) and apply $\boldsymbol{C}_e$ again from the right. This approach needs as many basis vector applications as there are shape functions per cell. The local result can be simply added to the global diagonal via $d = \sum_e \mathcal{G}_e^T d_e$. For cells without constrained DoFs, Equation (5) simplifies to $d_e(i) = \boldsymbol{A}_e(i,i)$.

3.3 Smoother

As the derivation of good matrix-free smoothers is an active research topic, we use Chebyshev iterations around a point-Jacobi method for smoothing [2]. This setup needs an operator evaluation and its diagonal representation according to Section 3.2. It is run on the levels defined by either the global-coarsening or the local-smoothing multigrid algorithm. In the case of local smoothing, refinement edges can be treated as homogeneous Dirichlet boundaries under the assumption that the right-hand-side vector is suitably modified.

3.4 Coarse-grid Solver

The algorithms described in Section 3.2 allow us to set up traditional coarse-grid solvers, e.g., a Jacobi solver, a Chebyshev solver, and direct solvers, but also AMG. In this publication, we mostly apply AMG as a coarse-grid solver, since we use it either on very coarse meshes (in this case, it falls back to a direct solver) or for problems discretized with linear elements, for which AMG solvers are competitive [66].

3.5 Transfer Operator

The prolongation operator $\mathcal{P}^{(f,c)}$ prolongates the result $\boldsymbol{x}$ from a coarse space c to a fine space f (between either different geometric grids or different polynomial degrees),

$$\boldsymbol{x}^{(f)} = \mathcal{P}^{(f,c)}\boldsymbol{x}^{(c)}.$$

According to the literature [14, 99], this can be done in three steps:

$$\boldsymbol{x}^{(f)} = \mathcal{W}^{(f)} \circ \tilde{\mathcal{P}}^{(f,c)} \circ C^{(c)}\boldsymbol{x}^{(c)} \tag{6}$$

with $C^{(c)}$ setting the values of constrained DoFs on the coarse mesh, particularly resolving the hanging-node constraints, $\tilde{\mathcal{P}}^{(f,c)}$ performing the prolongation on the discontinuous space as if no hanging nodes were present, and the weighting operator $\mathcal{W}^{(f)}$ zeroing out the DoFs constrained on the fine mesh.

To derive a matrix-free implementation, one can express Equation (6) for nested meshes as loops over all (coarse) cells (see also Figure 3):

$$\boldsymbol{x}^{(f)} = \sum_{e\in\{cells\}} \mathcal{S}_e^{(f)} \circ \mathcal{W}_e^{(f)} \circ \mathcal{P}_e^{(f,c)} \circ C_e^{(c)} \circ \mathcal{G}_e^{(c)}\boldsymbol{x}^{(c)}. \tag{7}$$

Here, $C_e^{(c)} \circ \mathcal{G}_e^{(c)}$ gathers the cell-relevant coarse DoFs and applies the constraints just as in the case of the matrix-free operator evaluation (3). The operator $\mathcal{P}_e^{(f,c)}$ embeds the coarse-space solution into the fine space, whereas $\mathcal{S}_e^{(f)}$ sums the result back to a global vector. Since multiple cells could add to the same global entry of the vector $\boldsymbol{x}^{(f)}$ during the cell loop, the values to be added have to be weighted with the inverse of the valence of the corresponding DoF. This is done by $\mathcal{W}_e^{(f)}$, which also ignores constrained DoFs (zero valence) to be consistent with Equation (6). Figure 4 shows, as an example, the values of $\mathcal{W}^{(f)}$ for a simple mesh for a scalar Lagrange element of degree $1 \le p \le 3$.

We construct the element prolongation matrices as

$$\left(\boldsymbol{P}_e^{(f,c)}\right)_{ij} = \left(\boldsymbol{M}_e^{(f)}\right)^{-1} \left(\phi_i^{(f)}, \phi_j^{(c)}\right)_{\Omega_e} \quad \text{with} \quad \left(\boldsymbol{M}_e^{(f)}\right)_{ij} = \left(\phi_i^{(f)}, \phi_j^{(f)}\right)_{\Omega_e},$$

where $\phi^{(c)}$ are the shape functions on the coarse cell and $\phi^{(f)}$ on the fine cell. Instead of treating each "fine cell" on its own, we group direct children of the coarse cells together and define the fine shape functions on the appropriate subregions. As a consequence, the "finite element" on the fine mesh depends both on the actual finite-element type (like on the polynomial degree and

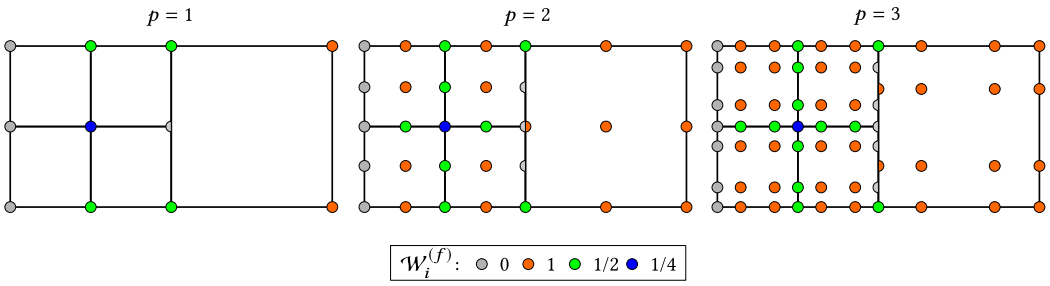


Fig. 4. Example for entries i of $\mathcal{W}^{(f)}$ for a simple mesh configuration with Dirichlet boundary at the left face for a scalar continuous Lagrange element of degree $1 \leq p \leq 3$. In our implementation, constrained DoFs do not contribute to the valence of constraining DoFs, which results in valences of one for DoFs inside constraining (coarse) edges/faces. To reduce overhead by $\mathcal{W}^{(f)}$, our implementation utilizes that, for $p > 2$, all DoFs of a geometric entity (vertex, edge, ...) have the same valence (stored per cell as 9 integers in 2D and 27 in 3D).

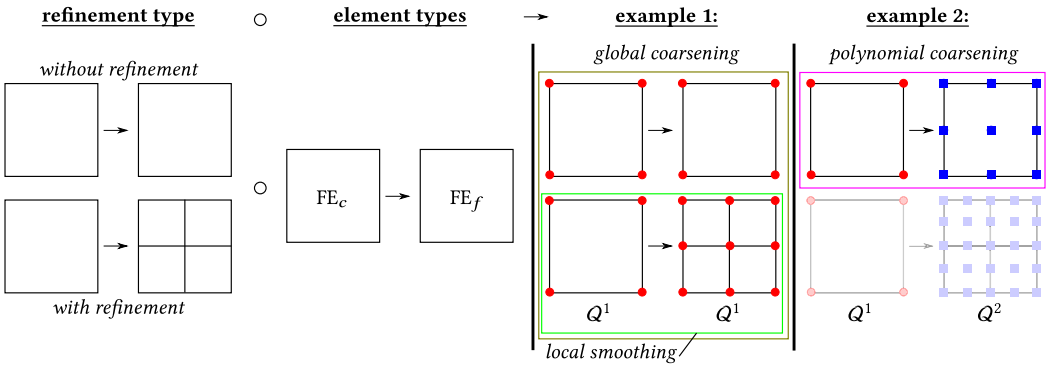


Fig. 5. Left: Construction of the element prolongation $\mathcal{P}_e^{(f,c)}$, based on the *refinement-type* (with/without refinement) and *element-types pair* (coarse and fine FE). Right: Examples for prolongation with and without refinement for equal-degree and different-degree finite elements ($p^{(c)} = p^{(f)} = 1$ vs. $p^{(c)} = 1, p^{(f)} = 2$). Relevant prolongation types for local smoothing, global coarsening, and polynomial coarsening are highlighted. Note that, in the case of global coarsening, two types of prolongation (categories) are needed.

continuity) and on the refinement type, as indicated in Figure 5. For local smoothing, there is only one refinement configuration, since all cells on a finer level are children of cells on the coarser level. In the case of polynomial global coarsening, the mesh stays the same and only the polynomial degree is uniformly changed for all cells. In the case of geometric global coarsening, cells on a finer level are either identical to the cells or direct children of cells on the coarser level, but the element and its polynomial degree stay the same. This necessitates two prolongation cases, an identity matrix for non-refined cells and the coarse-to-fine embedding. We define the set of all coarse–fine cell pairs connected via the same element prolongation matrix as category C.

Since $\mathcal{P}_e^{(f,c)} = \mathcal{P}_{C(e)}^{(f,c)}$, i.e., all cells of the same category $C(e)$ have the same element prolongation matrix, and to be able to apply them for multiple elements in one go in a vectorization-over-elements fashion [62] as in the case of matrix-free loops (3), we loop over the cells type by type so that Equation (7) becomes

$$\boldsymbol{x}^{(f)} = \sum_c \sum_{e \in \{e \mid C(e)=c\}} \mathcal{S}_e^{(f)} \circ \mathcal{W}_e^{(f)} \circ \mathcal{P}_c^{(f,c)} \circ \mathcal{C}_e^{(c)} \circ \mathcal{G}_e^{(c)} \boldsymbol{x}^{(c)}. \tag{8}$$

We choose the restriction operator as the transpose of the prolongation operator,

$$\mathcal{R}^{(c,f)} = (\mathcal{P}^{(f,c)})^T \quad \text{using locally} \quad \mathcal{R}_e^{(c,f)} = \left(\mathcal{P}_e^{(f,c)}\right)^T.$$

We conclude this subsection with discussing the appropriate data structures for transfer operators, beginning with the ones needed for both global geometric and polynomial coarsening. Since global-coarsening algorithms smoothen on the complete computational domain, data structures only need to be able to perform two-level transfers (8) independently between arbitrary fine (f) and coarse (c) grids. $\mathcal{C}_e^{(c)} \circ \mathcal{G}_e^{(c)}$ is identical to $\mathcal{C}_e \circ \mathcal{G}_e$ in the matrix-free loop (3) so that specialized algorithms and data structures [81] can be applied and reused. $\mathcal{S}_e^{(f)}$ needs the indices of DoFs for the given element e to be able to scatter the values, and $\mathcal{W}_e^{(f)}$ stores the weights of DoFs (or of geometric entities—see also Figure 4) for the given element e. $\mathcal{P}_{C(e)}^{(f,c)}$ (and $\mathcal{R}_{C(e)}^{(c,f)}$) need to be available for each category. We regard them as general operators and choose the most efficient way of evaluation based on the element types: simple dense-matrix representation vs. some substructure with sum factorization based on smaller 1D prolongation matrices. The category of each cell has to be known for each cell.

Alongside these process-local data structures, one needs access to all constraining DoFs on the coarse level required during $\mathcal{C}_e^{(c)} \circ \mathcal{G}_e^{(c)}$ and to the DoFs of all child cells on the fine level during the process-local part of $\mathcal{S}_e^{(f)}$, which is concluded by a data exchange. If external vectors do not allow access to the required DoFs, then we copy data to/from internal temporal global vectors with appropriate ghosting [62]. We choose the coarse-side identification of cells due to its implementation simplicity and structured data access at the price of more ghost transfer.[3] For setting up the communication pattern, we use consensus-based sparse dynamic algorithms [6, 47].

For the sake of separation of concerns, one might create three classes to implement a global-coarsening transfer operator as we have done in `deal.II`. The relation of these classes is shown in Figure 6: The multigrid transfer class (`MGTransferGlobalCoarsening`) delegates the actual transfer tasks to the right two-level implementation (`MGTwoLevelTransfer`), which performs communications needed as well as evaluates (8) for each category and cell by using category-specific information from the third class (`MGTransferSchemes`).

The discussed data structures are applicable also to local smoothing. The fact that only a single category is available in this case allows us to simplify many code paths. However, one needs data structures that match DoF indices on the active level and on *all* multigrid levels that share a part of the mesh with the active level in addition to the two-level data structures. For further details on implementation aspects of transfer operators for local smoothing, see References [28, 64] and the documentation of `MGTransferMatrixFree` class in `deal.II`.

4 PERFORMANCE METRICS

In Section 3, we have presented efficient implementations of the operators in Algorithm 1 for local smoothing, global coarsening, and polynomial coarsening. Most of the discussion was independent of the multigrid variant chosen, highlighting the similarities from the implementation

[3] Sundar et al. [99] showed that, by assigning all children of a cell to the same process, one can easily derive an algorithm that allows us to perform the cell-local prolongation/restriction on the fine side, potentially reducing the amount of data to be communicated during the transfer. Since we allow levels to be partitioned arbitrarily in our implementation, we do not use this approach. Furthermore, one should note that the algorithm proposed there does not allow us to apply the constraints $\mathcal{C}_e^{(c)}$ during a single cell loop as in Equation (8) but needs a global preprocessing step as in Equation (6), potentially requiring additional sweeps through the whole data with access to the slow main memory.

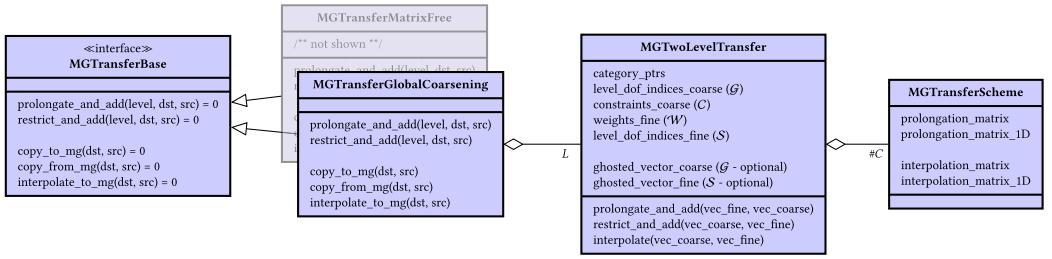


Fig. 6. UML diagram of the global-coarsening transfer operator `MGTransferGlobalCoarsening` in `deal.II`. It implements the base class `MGTransferBase`, which is also the base class of `MGTransferMatrixFree` (local smoothing) and delegates its prolongation/restriction/interpolation tasks to the right `MGTwoLevelTransfer` instance. Each of these instances is defined between two levels and loops over categories/cells to evaluate Equation (8) by using the prolongation matrices from the correct `MGTransferScheme` object. Furthermore, it is responsible for the communication, for which it has two optional internal vectors with appropriate ghosting. `MGTwoLevelTransfer` objects can be initialized for geometric or polynomial global coarsening, enabling support for (global) *h*-, *p*-, and *hp*-multigrid in `MGTransferGlobalCoarsening`.

point of view. The main differences arise naturally from the local or global definition of levels: e.g., one might need to consider hanging-node constraints during matrix-free loops when doing global coarsening or polynomial coarsening. Local smoothing, however, has the disadvantage of performing additional steps: (1) global transfer to/from multigrid levels and (2) special treatment of refinement edges. In the following sections, we quantify the influence of the costs of the potentially more expensive operator evaluations and of the additional operator evaluations, related to the choice of the multigrid level definition.

Our primary goal is to minimize the *time to solution*. It consists of setup costs and the actual solve time, which is the product of the *number of iterations* and *the time per iteration*. We will disregard the setup costs, since they normally amortize in time-dependent simulations, where one does not remesh every time step, and ways to optimize the setup of global-coarsening algorithms are known in the literature [99]. The time of the solution process strongly depends on the choice of the smoother, which influences the number of iterations and, as a result, also the time to solution. Since different iteration numbers might distort the view on the performance of the actual multigrid algorithm, we will also consider the value of the time per iteration as an important indicator of the computational performance of multigrid algorithms particularly to quantify the additional costs in an iteration.

To get a first estimate of the benefits of an algorithm compared to another, one can derive following metrics purely from geometrical information:

- The *serial workload* can be estimated as the sum of the number of cells on all levels $W_s = \sum_l C_l$, with C_l being the number of cells on level l. This metric is based on the assumption that all cells have the same costs, which is not necessarily true in the context of hanging nodes [81].
- The *parallel workload* can be estimated as the sum of the maximum number of cells owned by any process p on each level: $W_p = \sum_l \max_p C_l^p$, i.e., the critical path of the cells. In the ideal case, one would expect that $C_l^p = C_l/P$ with P being the number of processes and therefore $W_p = W_s/P$. However, due to potential load imbalances, the work might not be well distributed on the levels, i.e., $\max_p (C_l^p) > C_s/P$. Since one can theoretically only proceed to the next level once all processes have finished processing a level, load imbalances will result in some processes waiting at imaginary barriers. We say "imaginary barriers" as level

operators only involve point-to-point communication between neighboring processes in the grid for our implementation. Nevertheless, this simplified point of view is acceptable, since multigrid algorithms are multiplicative Schwarz methods between levels, inherently leading to a sequential execution of the levels. We define *parallel workload efficiency* as $W_s/(W_p \cdot P)$, as has been also done in Reference [28].

- We define *horizontal communication efficiency* as 50% of the number of ghost cells accumulated over all ranks and divided by the total number of cells. The division by two is necessary to take into account that only one neighbor is updating the ghost values. As such, this ratio can be seen as a proxy of how much information needs to be communicated when computing residuals and updating ghost values. As this number counts cells, it is independent of the polynomial degree of the element chosen. The element degree used determines the absolute amount of communication necessary. Note that, in reality, only degrees of freedom located at the interface have to be exchanged such that the fraction of the solution that needs to be communicated is less than the fraction of those cells.
- *Vertical communication efficiency* is the share of fine cells that have the same owning process as their corresponding coarse cell (parent). This quantity gives an indication on the efficiency of the transfer operator and on how much data has to be exchanged. A small number indicates that most of the data has to be completely permuted, involving a large volume of communication. This metric has been considered in Reference [28] as well.
- Increasing the *number of (multigrid) levels* leads to additional synchronization points and communication steps and, as a result, might lead to increased latency. The absolute value of the minimum time per V-cycle can be roughly approximated by

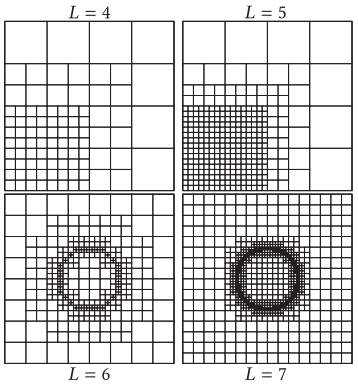
$$\Delta t^{\texttt{V-Cycle}}_{\text{Latency}} = L(\underbrace{(k-1)}_{\mathcal{S}_{\text{pre}}} + \underbrace{1}_{\text{residual}} + \underbrace{1}_{\mathcal{R}} + \underbrace{1}_{\mathcal{P}} + \underbrace{k}_{\mathcal{S}_{\text{post}}})\Delta t^{\text{vmult}}_{\text{latency}} = 2L(k+1)\Delta t^{\text{vmult}}_{\text{latency}} \tag{9}$$

as the product of $t^{\text{vmult}}_{\text{latency}}$ (the latency of an operator evaluation consisting of one update to the ghost values in the source vector and one step to send integral contributions in the destination vector back to the owner) and the sum of all communication steps accumulated over presmoothing, residuum evaluation, restriction, prolongation, and postsmoothing on all levels. In this approximation, we assume that the coarse-grid solver does not perform any communication and has negligible latency. Furthermore, the smoother is assumed to have the same communication cost as one matrix-vector product in each of the k iterations, which is the case for simple Chebyshev iterations around point-Jacobi preconditioners [58, 65].

The metrics "parallel workload efficiency," "horizontal communication efficiency," and "vertical communication efficiency," as well as "latency" (at the scaling limit) all influence the experimentally measurable "parallel (strong-scaling) efficiency" of a multigrid solver.

Furthermore, *memory consumption* of the grid class is a secondary metric.[4] A common argument supporting the usage of local-smoothing algorithms is that no space is needed for potentially differently partitioned meshes and complex data structures providing the connectivity between them, since the multigrid algorithm can simply reuse the already existing mesh hierarchy also for the multigrid levels [28].

[4]We use the memory-consumption output provided by `deal.II`. No particular efforts have been put in reducing the memory consumption of the triangulations in the case of global coarsening.



	octant				shell			
			#dofs				#dofs	
L	#cells	%HN	$p=1$	$p=4$	#cells	%HN	$p=1$	$p=4$
3	1.2e+2	31%	2.2e+2	9.3e+3	-	-	-	-
4	7.0e+2	37%	1.0e+3	5.1e+4	-	-	-	-
5	4.7e+3	23%	5.7e+3	3.2e+5	1.2e+3	69%	2.0e+3	9.3e+4
6	3.5e+4	12%	3.8e+4	2.3e+6	6.8e+3	78%	9.8e+3	5.1e+5
7	2.7e+5	6.2%	2.8e+5	1.8e+7	3.7e+4	70%	4.8e+4	2.6e+6
8	2.1e+6	3.1%	2.2e+6	1.4e+8	2.7e+5	38%	3.2e+5	1.9e+7
9	1.7e+7	1.6%	1.7e+7	1.1e+9	2.2e+6	19%	2.3e+6	1.4e+8
10	1.3e+8	0.8%	1.4e+8	8.6e+9	1.7e+7	10%	1.8e+7	1.1e+9
11	1.1e+9	0.4%	1.1e+9	6.9e+10	1.4e+8	4.8%	1.4e+8	8.9e+9
12	8.6e+9	0.2%	8.9e+9	-	1.1e+9	2.4%	1.1e+9	7.1e+10
13	-	-	-	-	8.9e+9	1.2%	8.9e+9	5.7e+11

Fig. 7. Cross section at the center of the geometries of the `octant` (top) and the `shell` (bottom) simulation. Additionally, the number of cells, the share of cells with hanging-node constraints, and the number of DoFs (for a scalar Lagrange element with $p = 1$ and $p = 4$) are given for each refinement case.

Examples

In the experimental Sections 5 and 6, we consider two types of static 3D meshes, as has been also done in Reference [28]. They are obtained by refining a coarse mesh consisting of a single cell defined by $[-1, 1]^3$ according to one of the following two solution criteria:

- `octant`: refine all mesh cells in the first octant $[-1, 0]^3$ L times and
- `shell`: after $L - 3$ uniform refinements, perform three local refinement steps with all cells whose center $\boldsymbol{c}$ is $|\boldsymbol{c}| \leq 0.55$, $0.3 \leq |\boldsymbol{c}| \leq 0.43$, and $0.335 \leq |\boldsymbol{c}| \leq 0.39$.

These two meshes are relevant in practice, since similar meshes occur in simulations of flows with far fields and of multi-phase flows with bubbles [59] or any kind of interfaces. All the refinement procedures are completed by a closure after each step, ensuring one-irregularity in the sense that two leaf cells may only differ by one level if they share a vertex. Figure 7 shows the considered meshes and provides numbers regarding the cell count for $3 \leq L \leq 13$. All meshes are partitioned along space-filling curves [15, 26] with the option to assign cells weights.

Tables 1 and 2 give—as examples—evaluated numbers for geometrical metrics of the two considered meshes for a single process and for 192 processes with cells constrained by hanging nodes weighted with the factor of 2 for partitioning, compared to the rest of the cells. For a single process, only workload and memory consumption are shown. In the tables, bold indicates the most beneficial behavior among the listed variants.

Starting with the `octant` case, one can see that the serial workload in the case of local smoothing and global coarsening is similar, with local smoothing having consistently less work. The workload of each level is depicted in Figure 8. The behavior of the memory consumption is similar to the one of the workload: Global coarsening has a slightly higher memory consumption, since it explicitly needs to store the coarser meshes as well; however, the second finest mesh has already approximately one eighth of the size of the coarsest triangulation so that the overhead is small. In the parallel case, the memory consumption differences are higher: This is related to the fact that—in the case of global coarsening—overlapping ghosted forests of trees need to be saved. In contrast, the workload in the parallel case is much lower in the case of global coarsening: While global coarsening is able to reach efficiencies higher than 90%, the efficiency is only approximately 50–60% in the case of local smoothing. The high value in the global-coarsening case was to be expected, since we repartition each level during construction. The low value in the case of local

Table 1. Geometrical Multigrid Statistics for the octant Test Case for Different Numbers of Refinements

	1 process				192 processes									
	local smoothing		global coarsening		local smoothing					global coarsening				
L	wl	mem	wl	mem	wl	wl-eff	v-eff	h-eff	mem	wl	wl-eff	v-eff	h-eff	mem
3	**1.4e+2**	**6.6e+4**	1.4e+2	8.7e+4	**1.8e+1**	**3%**	**89%**	60%	**1.3e+6**	2.5e+1	3%	17%	1.0e+5	2.7e+6
4	**8.0e+2**	**3.3e+5**	8.4e+2	4.2e+5	**2.2e+1**	**18%**	**85%**	56%	**1.2e+7**	3.3e+1	13%	4%	1.0e+5	1.5e+7
5	**5.4e+3**	**2.0e+6**	5.6e+3	2.5e+6	7.2e+1	38%	**88%**	58%	**5.3e+7**	**6.7e+1**	**43%**	1%	**59%**	6.8e+7
6	4.0e+4	**1.4e+7**	4.0e+4	1.7e+7	4.0e+2	51%	**96%**	65%	**1.3e+8**	**2.8e+2**	**76%**	6%	**66%**	2.0e+8
7	3.1e+5	**1.1e+8**	3.1e+5	1.3e+8	2.7e+3	58%	**99%**	75%	**4.2e+8**	**1.7e+3**	**92%**	13%	75%	6.2e+8
8	2.4e+6	**8.6e+8**	2.4e+6	9.9e+8	2.0e+4	62%	**99%**	84%	**1.8e+9**	**1.3e+4**	**96%**	23%	84%	2.4e+9
9	1.9e+7	**6.8e+9**	1.9e+7	7.8e+9	1.6e+5	64%	**99%**	91%	**1.0e+10**	**1.0e+5**	**98%**	38%	91%	1.3e+10

wl: Serial/Parallel Workload; wl-eff: Parallel Workload Efficiency; v-eff: Vertical Communication Efficiency; h-eff: Horizontal Communication Efficiency; mem: Memory Consumption in Bytes. Figures 8 (serial) and 9 (parallel) give a detailed breakdown of the workload of each multigrid level for $L = 8$.

Table 2. Geometrical Multigrid Statistics for the shell Test Case for Different Numbers of Refinements

	1 process				192 processes									
	local smoothing		global coarsening		local smoothing					global coarsening				
L	wl	mem	wl	mem	wl	wl-eff	v-eff	h-eff	mem	wl	wl-eff	v-eff	h-eff	mem
5	**1.4e+3**	**6.0e+5**	1.8e+3	9.5e+5	3.1e+1	**22%**	**80%**	55%	**3.2e+7**	4.5e+1	21%	2%	**56%**	4.7e+7
6	**7.8e+3**	**3.2e+6**	9.2e+3	4.4e+6	1.6e+2	26%	**89%**	59%	**7.7e+7**	**1.0e+2**	**47%**	12%	59%	1.3e+8
7	**4.2e+4**	**1.7e+7**	4.9e+4	2.2e+7	7.6e+2	28%	**96%**	**66%**	**1.5e+8**	**4.3e+2**	**58%**	36%	65%	3.0e+8
8	**3.1e+5**	**1.2e+8**	3.4e+5	1.4e+8	4.7e+3	34%	**99%**	**76%**	**4.4e+8**	**2.3e+3**	**75%**	78%	75%	7.7e+8
9	2.5e+6	**8.9e+8**	2.5e+6	1.1e+9	3.5e+4	36%	**99%**	**85%**	**1.8e+9**	**1.5e+4**	**86%**	93%	84%	2.7e+9
10	—	—	—	—	2.7e+5	38%	**99%**	91%	**1.0e+10**	**1.1e+5**	**92%**	97%	91%	1.3e+10

The label "-" indicates that the simulation ran out of memory.

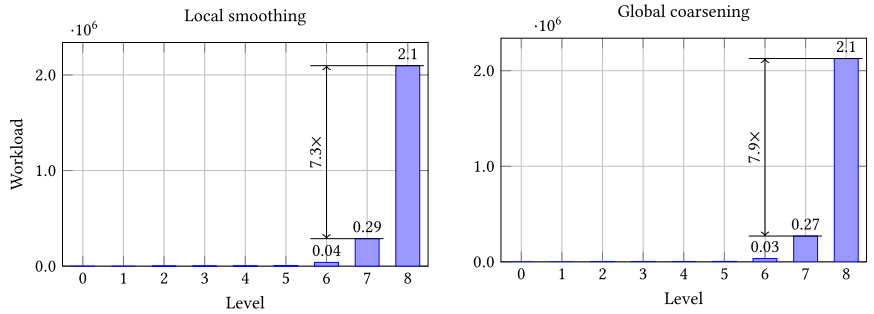


Fig. 8. Workload of each multigrid level of an octant simulation with a single process for $L = 8$.

smoothing can be seen in Figure 9, where the minimum, maximum, and average workload are shown for each level. The workload on the finest level is optimally distributed between processes with cells; however, there are some processes without any cells, i.e., any work, on that level. On the second level, most processes can reduce the number of cells nearly optimally by a factor of 8, but processes idle on the finest level start to participate in the smoothing process with a much higher number of cells, similarly to what other processes process on the finest level. This pattern of discrepancy between the minimum and the maximum number of cells on the lower levels continues and, as a consequence, the maximum workload per level is higher, increasing the overall parallel workload. Figure 9 might also give the impression of a load imbalance in the case of global coarsening, since the minimum workload on the finest level is the half of the maximum value. This is related to the way we partition—penalizing cells that have hanging nodes with a weight of

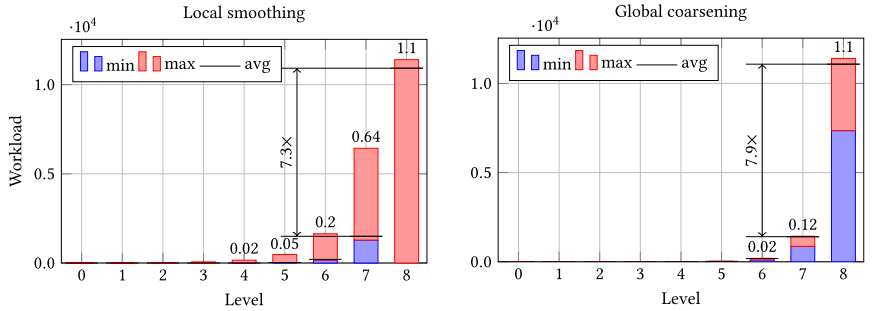


Fig. 9. Workload of each multigrid level of an octant simulation with 192 processes for $L = 8$.

2—and to the fact that a lot of cells with hanging nodes are clustered locally. The actual resulting load imbalance is small, as shown in Figure 12. The difference between local smoothing and global coarsening in the parallel workload comes at a price. While the vertical communication efficiency is—by construction—high in the local-smoothing case, this is not true in the case of global coarsening: Twenty percent or less is not uncommon, requiring the permutation of data during transfer. The horizontal communication costs are similar in the case of local smoothing and global coarsening. Assuming that (1) pre- and postsmoothing are the most time-consuming steps, (2) the parallel workload is the relevant metric, and (3) the transfer between levels is not dominant in this case, the values indicate that global coarsening might be twice as fast as local smoothing. However, if the transfer is the bottleneck, then the picture might look differently so that a conclusive statement only based on geometrical metrics is not possible in this case.

The observations for the `octant` case are also made, in a more pronounced form, in the `shell` case. Here, the vertical communication is more favorable in the case of global coarsening, and the workload efficiency is significantly worse in the local-smoothing case so that one can expect a noticeable speedup when using global coarsening.

In summary, one can state the following: In the serial case, global coarsening has to process at least as many cells as local smoothing so that one can expect that the latter has—with the assumption that the number of iterations is the same—an advantage regarding throughput, particularly since no hanging-node constraints have to be applied. With an increasing number of processes, the workload might not be well distributed in the case of local smoothing, leading to a drop in parallel efficiency. In contrast, this is—by construction—not an issue in the global-coarsening case, since the work is explicitly redistributed between the levels. The price is that one might need to send around a lot of data during restriction and prolongation. The number of levels in the case of local smoothing and global coarsening is the same, leading to potentially same scaling limits $O(L)$. However, with appropriate partitioning of the levels, one could decrease the number of participating processes on the coarser levels and switch to a coarse-grid solver at an earlier stage in the global-coarsening case, leading to a better scaling limit due to lower latency. In the following, we show experimentally that the above statements can be verified and take a more detailed look at the influence of the mutually contradicting requirements “good workload balance” and “cheap transfer” on the parallel efficiency.

Making definite general conclusions is difficult as they depend on the number of processes as well as on the type of the coarse mesh and of the refinement. While the two meshes considered here are prototypical for many problems we have encountered in practice, it is clear that the statements made in this publication cannot hold in all cases, since it is easy to construct examples that favor one multigrid variant over the other. However, the two meshes clearly demonstrate the

dominating aspects at various problem sizes; the actual crossover point, however, might be problem- and hardware-specific.

5 PERFORMANCE ANALYSIS: H-MULTIGRID

In the following, we solve a 3D Poisson problem with homogeneous Dirichlet boundary conditions and a constant right-hand side to compare the performance of local-smoothing and global-coarsening algorithms for the `octant` and `shell` test cases, as introduced in Section 4. The results for a more complex setup with non-homogeneous Dirichlet boundary conditions are shown in Table 7 in Appendix A.

We will use a continuous Lagrange finite element, whose shape functions are defined as the tensor products of 1D shape functions with degree p. For quadrature, we consider the consistent Gauss–Legendre quadrature rule with $(p+1)^3$ points.

We start with investigating the serial performance, proceed with parallel execution with moderate numbers of processes, and, finally, analyze the parallel behavior on the large scale (150k processes). We conclude this section with investigation of an alternative partitioning scheme for global coarsening.

To obtain the best performance, the experiments are configured in the following way:

- Cells with hanging-node constraints are weighted by the factor of 2.
- The conjugate-gradient solver is run until a reduction of the l_2-norm of the unpreconditioned residual by 10^4 is obtained. We choose a rather coarse tolerance, since this is a common value for the solution of time-dependent problems, such as the Navier–Stokes equations, where good initial guesses can be obtained by projection and extrapolation without the need to converge multigrid to many digits. Similarly, coarse tolerances also indicate the costs of solving in a full-multigrid scenario with the finest level only correcting against the next coarser one.
- The conjugate-gradient solver is preconditioned by a single V-cycle of either a local-smoothing or a global-coarsening multigrid algorithm.
- To increase the throughput, all operations in the multigrid V-cycle are run with single-precision floating-point numbers, while the conjugate-gradient solver is run in double precision [64].
- We use a Chebyshev/point-Jacobi smoother of degree 3 on all levels.
- As coarse-grid solver, we use two V-cycles of AMG (double-precision, ML [34] with parameters shown in Appendix C).

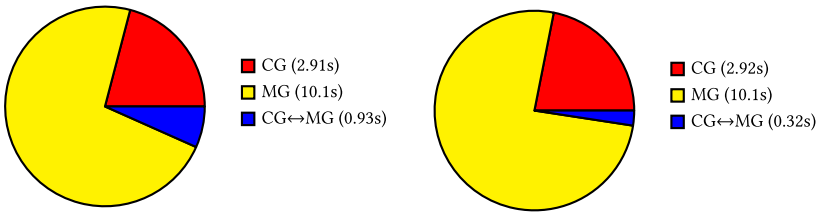
The results of performance studies leading to the decision on the configuration described above are presented in Tables 8–12. All experiments have been conducted on the SuperMUC-NG supercomputer. Its compute nodes have two sockets (each with 24 cores of Intel Xeon Skylake) and the AVX-512 ISA extension so that 8 doubles or 16 floats can be processed per instruction. A detailed specification of the hardware is given in Table 3. The parallel network is organized into islands of 792 compute nodes each. The maximum network bandwidth per node within an island is 100 GBit/s = 12.5GB/s[5] via a fat-tree network topology. Islands are connected via a pruned-tree network architecture (pruning factor 1:4). We have run all experiments six times and report the best timings of the last five runs. We note that executions on SuperMUC-NG are relatively noisy even for small-scale simulations with hundreds of processes, which results in performance degeneration by up to 50%. Our investigations revealed that this has only a minor effect on the execution times of the multigrid variants in comparison. Nevertheless, to rule out any source of inconsistency in

[5]https://doku.lrz.de/display/PUBLIC/SuperMUC-NG, retrieved on February 26, 2022.

Table 3. Specification of the Hardware System Used for Evaluation

	Intel Skylake Xeon Platinum 8174
cores	2×24
frequency base (max AVX-512 frequency)	2.7 GHz
SIMD width	512 bit
arithmetic peak (dgemm performance)	4147 GFLOP/s (3318 GFLOP/s)
memory interface	DDR4-2666, 12 channels
STREAM memory bandwidth	205 GB/s
empirical machine balance	14.3 FLOP/Byte
L1-/L2-/L3-/MEM size	32kB (per core)/1MB (per core)/66MB (shared)/96GB(shared)
compiler + compiler flags	g++, version 9.1.0, `-O3 -funroll-loops -march=skylake-avx512`

Memory bandwidth is according to the STREAM triad benchmark (optimized variant without read for ownership transfer involving two reads and one write), and GFLOP/s are based on the theoretical maximum at the AVX-512 frequency. The dgemm performance is measured for $m = n = k = 12{,}000$ with Intel MKL 18.0.2. We measured a frequency of 2.5 GHz with AVX-512 dense code for the current experiments. The empirical machine balance is computed as the ratio of measured dgemm performance and STREAM bandwidth from RAM memory.



(a) Local smoothing (per it.: 13.94s). (b) Global coarsening (per it.: 13.34s).

Fig. 10. Time per iteration spent for conjugate-gradient solver (CG), multigrid preconditioner (MG), as well as transfer between solver and preconditioner (CG↔MG) in a serial `shell` simulation with $k = 4$ and $L = 9$.

the results, we report timings for all multigrid variants from the same job execution in each table and figure. Apart from these machine-caused outliers, the statistical distribution is within a few percentages of the minimum and not explicitly reported in the following figures.

For local smoothing and global coarsening, we use different implementations of the transfer operator from `deal.II` (see Section 3.5). To demonstrate that they are equivalent and results shown in the following are indeed related to the definition of multigrid levels and the resulting different algorithms, Table 13 presents a performance comparison of these implementations for uniformly refined meshes of a cube, for which both algorithms are equivalent.

For global coarsening, we have also investigated the possibility to decrease the number of participating processes and to switch to the coarse-grid solver earlier. For our test problems, we could not see any obvious benefits for the time to solution so that we do not use these features of global coarsening in this publication but defer their investigation to future work.

5.1 Serial Runs: Overview

Figure 10 gives an overview of the time shares during the solution process in a serial `shell` simulation for local smoothing and global coarsening. Without going into details of the actual numbers, one can see that most of the time is spent in the multigrid preconditioner in the case of both local smoothing and global coarsening (72%/76%). It is followed by the other operations in the outer conjugate-gradient solver (21%/22%). The least time is spent for transferring data between preconditioner and solver (7%/2%). The higher time of the transfer in the local-smoothing case is not surprising, since the transfer involves all multigrid levels sharing cells with the active level.

Table 4. Number of Iterations and Time to Solution for Local Smoothing (LS) and Global Coarsening (GC) with 1 Process and 192 Processes for the octant Simulation Case

	1 process								192 processes (4 nodes)							
	$p=1$				$p=4$				$p=1$				$p=4$			
	LS		GC		LS		GC		LS		GC		LS		GC	
L	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]
3	**4**	**5.0e-4**	4	6.0e-4	**4**	**3.8e-3**	4	4.3e-3	4	1.2e-3	**4**	**1.0e-3**	4	2.5e-3	**4**	**2.5e-3**
4	**4**	**1.8e-3**	4	2.5e-3	**4**	**1.8e-2**	4	2.0e-2	4	2.6e-3	**4**	**1.7e-3**	4	4.3e-3	**4**	**4.1e-3**
5	**4**	**8.6e-3**	4	1.2e-2	4	1.1e-1	**3**	**8.7e-2**	4	5.3e-3	**4**	**2.6e-3**	4	6.9e-3	**3**	**5.0e-3**
6	**4**	**5.1e-2**	4	6.5e-2	4	8.8e-1	**3**	**6.5e-1**	4	5.6e-3	**4**	**4.1e-3**	4	1.7e-2	**3**	**1.0e-2**
7	4	3.6e-1	**3**	**3.2e-1**	4	6.9e+0	**3**	**5.0e+0**	4	1.3e-2	**3**	**5.7e-3**	4	8.4e-2	**3**	**5.0e-2**
8	4	2.8e+0	**3**	**2.3e+0**	4	5.3e+1	**3**	**3.8e+1**	4	3.9e-2	**3**	**2.1e-2**	4	7.2e-1	**3**	**4.3e-1**
9	4	2.2e+1	**3**	**1.8e+1**	—	—	—	—	4	2.3e-1	**3**	**1.3e-1**	—	—	—	—

Figures 11 (serial) and 12 (parallel) give a detailed breakdown of the times of one V-cycle for the case shaded in gray, $L = 8$ and $p = 4$.

Table 5. Number of Iterations and Time to Solution for Local Smoothing (LS) and Global Coarsening (GC) with 1 Process and 192 Processes for the shell Simulation Case

	1 process								192 processes (4 nodes)							
	$p=1$				$p=4$				$p=1$				$p=4$			
	LS		GC		LS		GC		LS		GC		LS		GC	
L	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]
5	**5**	**3.6e-3**	4	7.1e-3	**4**	**3.1e-2**	4	4.5e-2	5	6.6e-3	**4**	**3.1e-3**	4	6.4e-3	**4**	**6.3e-3**
6	**5**	**1.5e-2**	4	2.8e-2	**4**	**1.9e-1**	4	2.2e-1	5	6.8e-3	**4**	**4.1e-3**	4	1.0e-2	**4**	**9.2e-3**
7	**5**	**7.3e-2**	4	1.3e-1	**4**	**1.0e+0**	4	1.2e+0	5	9.2e-3	**4**	**5.6e-3**	4	2.3e-2	**4**	**1.8e-2**
8	**5**	**5.1e-1**	4	7.3e-1	**4**	**7.8e+0**	4	7.9e+0	5	1.7e-2	**4**	**1.2e-2**	4	1.1e-1	**4**	**6.9e-2**
9	**5**	**3.7e+0**	4	4.3e+0	4	5.8e+1	**4**	**5.6e+1**	5	6.3e-2	**4**	**3.5e-2**	4	8.5e-1	**4**	**5.2e-1**
10	—	—	—	—	—	—	—	—	5	3.9e-1	**4**	**1.9e-1**	—	—	—	—

Since the overwhelming share of solution time is taken by the multigrid preconditioner, all detailed analysis in the remainder of this work concentrates on the multigrid V-cycle.

One should note that spending 72%/76% of the solution time within the multigrid preconditioner is already low, particularly taking into account that the outer conjugate-gradient solver also performs its operator evaluations (1 matrix-vector multiplication per iteration) in an efficient matrix-free fashion. The low costs are related to the usage of single-precision floating-point numbers and to the low number of pre-/postsmoothing steps, which results in a total of six to seven matrix-vector multiplications per level and iteration.

5.2 Serial Run

Tables 4 and 5 show the number of iterations and the time to solution for the octant, and the shell test cases run serially with local smoothing and global coarsening as well as with the polynomial degrees $p = 1$ and $p = 4$.

It is well visible that local smoothing has to perform at least as many iterations as global coarsening, with the difference in iterations limited to 1 in the examples considered. This difference is due to the additional smoothening applied to certain cells in the course of global coarsening. Note that global coarsening also benefits from the simple setup of a smooth solution with artificial refinement, see Table 7 for a somewhat more realistic test case. This comes with a higher serial workload for global coarsening, which is also visible in the times of a single V-cycle (not shown). Nevertheless, the lower number of iterations leads to a smaller time to solution in the case of global coarsening in some instances. Generally, the global-coarsening V-cycle is relatively more expensive in the case of the shell simulation with linear elements: This is not surprising due to the higher number of cells with hanging-node constraints (see also Figure 7) in the shell case and

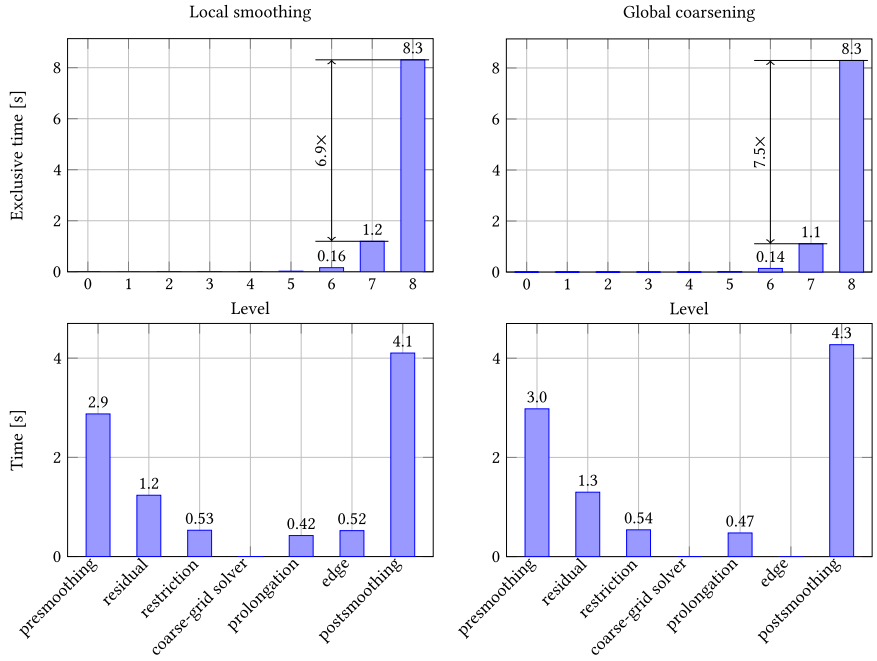


Fig. 11. Profile of one V-cycle of an octant simulation with a single process for $L = 8$ and $p = 4$.

the higher overhead of linear elements for application of hanging-node constraints, as analyzed in Reference [81].

Figure 11 shows the distribution of times spent on each multigrid level and in each multigrid stage for the `octant` case with $L = 8$ and $p = 4$. While the overall runtimes are similar for local smoothing and global coarsening, distinct (and expected) differences are visible for the individual ingredients: The `restriction` and `prolongation` steps take about the same time in both cases. The `presmoothing`, `residual`, and `postsmoothing` steps are slightly more expensive in the global-coarsening case, which is related to the observation that the evaluation of the level operator $\boldsymbol{A}^{(l)}$ (2/1/3 times) is the dominating factor. For local smoothing, the computation of $\boldsymbol{A}_{ES}^{(l)}\boldsymbol{x}_{S}^{(l)}$ is realized as a side product of the application of $\boldsymbol{A}^{(l)}$, and hence limiting its impact on the residual step. The only visible additional cost of local smoothing is $\boldsymbol{A}_{SE}^{(l)}\boldsymbol{x}_{E}^{(l)}$ for the modification of the right-hand-side vector for postsmoothing to incorporate inhomogenous Dirichlet boundary conditions (`edge` step). However, its evaluation is less expensive than the application of $\boldsymbol{A}^{(l)}$, since only DoFs in proximity to the interface are updated.

5.3 Moderately Parallel Runs

We report our findings of simulations with 192 processes on four compute nodes. Tables 4 and 5 confirm the significance of a good workload balance for the time to solution, as indicated by Tables 1 and 2. The number of processes does not influence the number of iterations due to the chosen smoother. Speedups—compared to local smoothing—are reached: up to 2.3/1.7 for the `octant` ($p = 1/p = 4$) and up to 2.1/1.6 for the `shell` case if the different iteration numbers are considered. Normalized per solver iteration, the advantage of global coarsening in these four cases is 2.0, 1.3, 1.7, and 1.6, respectively. Figure 12 illustrates this behavior by showing the distribution of the minimum/maximum/average times spent on each multigrid level and each multigrid

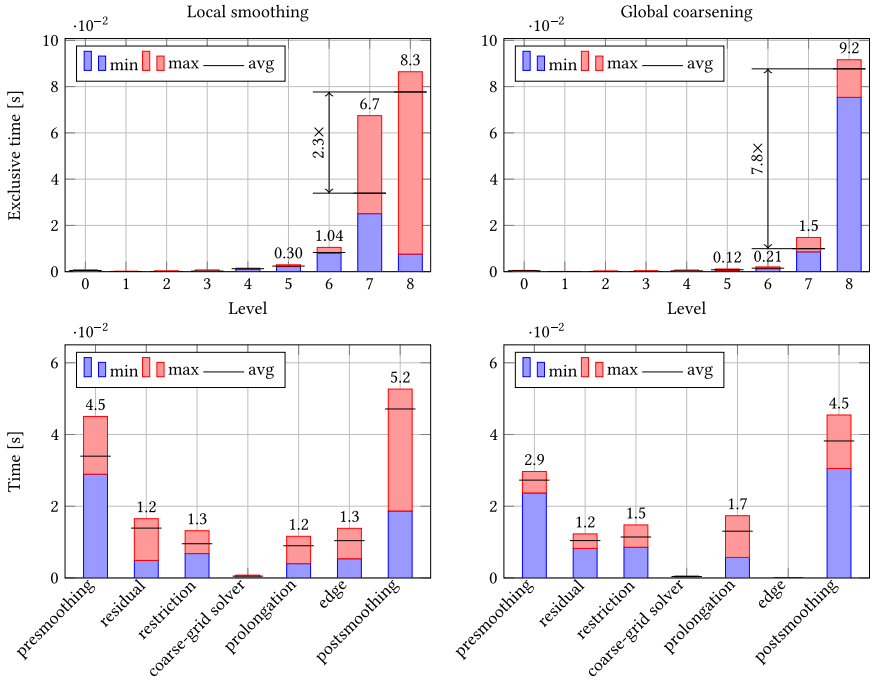


Fig. 12. Profile of one V-cycle of an octant simulation with 192 processes for $L = 8$ and $p = 4$.

stage for the `octant` case with $L = 8$. In the case of global coarsening, it is well visible that the load is equally distributed and the time spent on the levels is significantly reduced level by level for the higher levels. For the finest ones, local smoothing shows a completely different picture. On the finest level, there are processes with hardly any work, but nevertheless the average work is close to the maximum value, indicating that the load is well balanced among the processes with work. However, on the second finest level, a significant workload imbalance—by a factor of 2.8—is visible also among the processes with work. This leads to the situation that the maximum time spent on the second finest level is just slightly less than the one on the finest level, contradicting our expectation of a geometric series and leading to the observed increase in the total runtime.

5.4 Large-scale Parallel Run

Figures 13 and 14 show results of scaling experiments starting with 1 compute node (48 processes) up to 3,072 nodes (147,456 processes). Besides the times of a single V-cycle, we plot the *normalized throughput* (DoFs per process and time per iteration) against the *time per iteration*. The plot can be read in the following way: Far from the scaling limit (right top corner of the plot), simulations take longer but are more efficient (parallel efficiency of one); at the scaling limit (left bottom corner of the plots), simulations reach shorter times at the cost of lower efficiencies. Here, a horizontal behavior from the right to the left corresponds to ideal strong scaling. If different lines coincide, then we observe ideal weak scaling. Based on this plot, one can judge the resource utilization and parallel efficiency of a simulation if a certain time to solution should be obtained.

The throughput for the polynomial degree $p = 4$ is higher by a factor of 3 than for $p = 1$. This is expected and related to the used matrix-free algorithms and their node-level performance, which improves with the polynomial order [66]. Just as in the moderately parallel case (see Section 5.3),

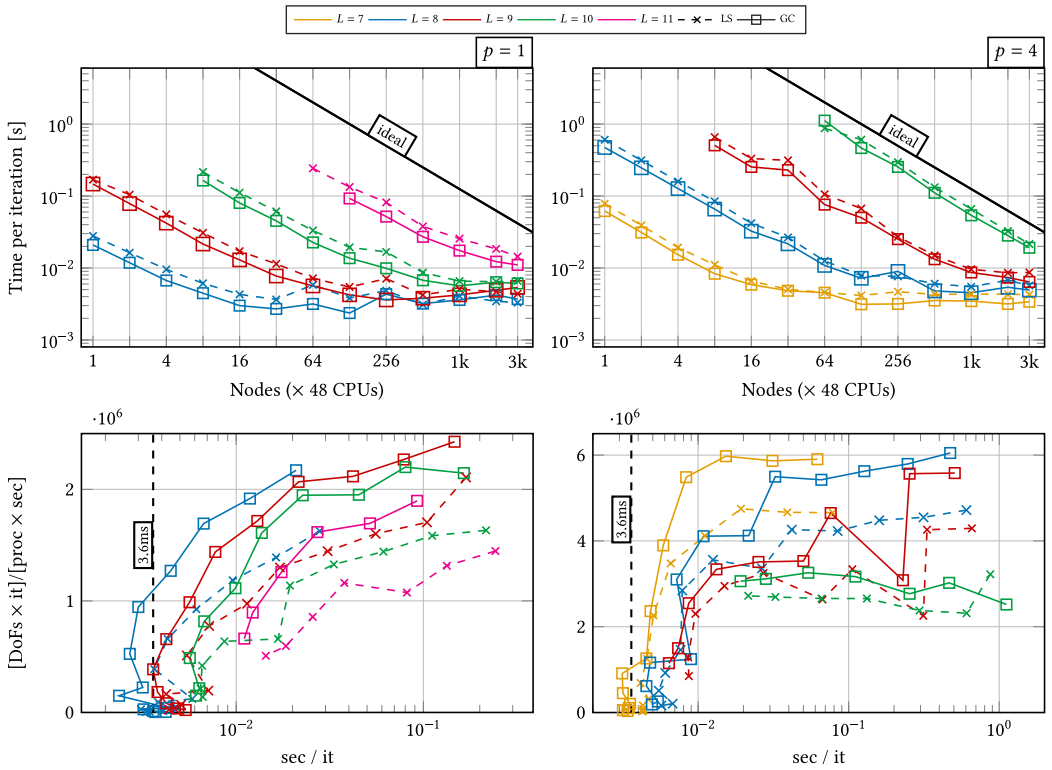


Fig. 13. Strong-scaling comparison of local smoothing (LS) and global coarsening (GC) for `octant` for $p = 1$ and $p = 4$. The label "3.6ms" indicates the theoretical scaling limit for $L = 9$.

we can observe better timings in the case of global coarsening for a large range of configurations (maximum speedup: `octant` 1.9/1.4 for $p = 1/p = 4$, `shell`: 2.4/2.4). The number of iterations of local smoothing is 4 for both cases and all refinement numbers. For global coarsening, it is 4 for the `sphere` case and decreases from 4 to 2 with increasing number of refinements in the case of `quadrant` so that the actual speedups reported above are even higher. The high speedup numbers of global coarsening in the `shell` simulation case are particularly related to its high workload and vertical efficiency, as shown in Table 5.

The normalized plots give additional insights. Apart from the obvious observation that the minimal time to solution increases with increasing number of refinements (left bottom corner of the plots in Figures 13 and 14) and global coarsening starts with higher throughputs, one can also see that the decrease in parallel efficiency is more moderate in the case of global coarsening. For the example of the `octant` case with $p = 1/L = 10$, one can increase the number of processes by a factor of 16 and still obtain a throughput per process that is higher than the one in the case of single-node computations of local smoothing, which normally shows a kink in efficiency at early stages (particularly visible in the `shell` case, which matches the findings made in Reference [28]). A further observation is that the lines for global coarsening overlap far from the scaling limit, i.e., the throughput is independent of the number of processes and the number of refinements. This is not the case for local smoothing, where the throughput deteriorates with the number of levels, indicating load-balance problems. The simulations with $p = 4$ show similar trends, but the lines are not as smooth, possibly due to the decreased granularity for higher orders.

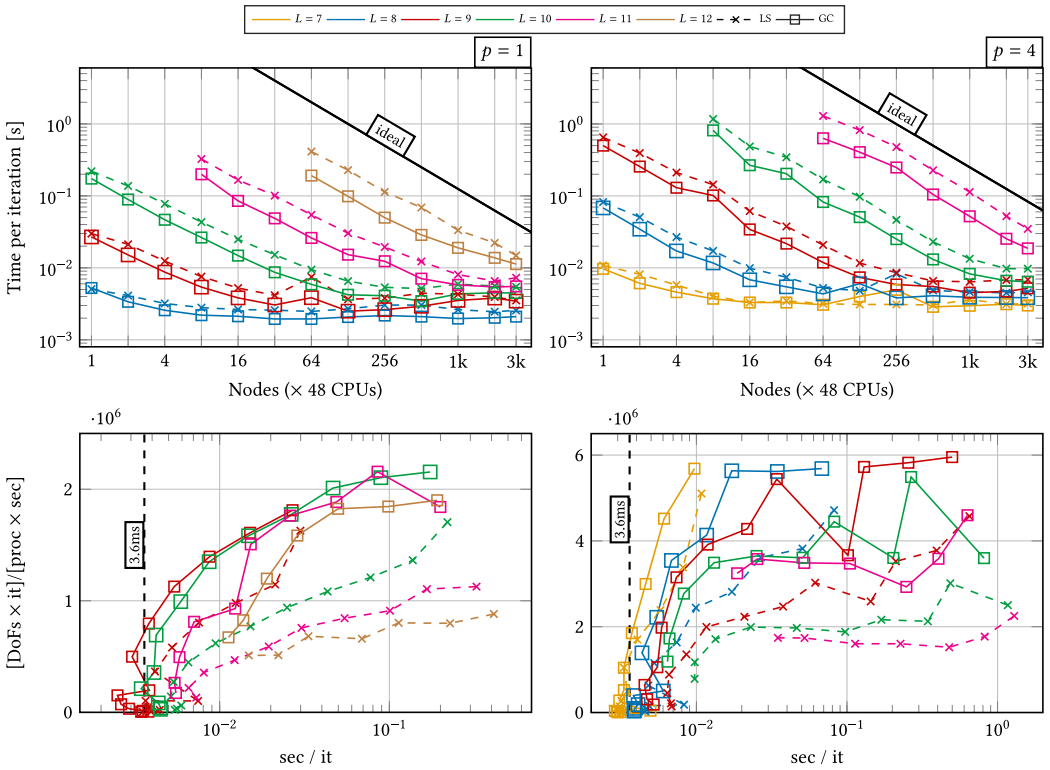


Fig. 14. Strong-scaling comparison of local smoothing (LS) and global coarsening (GC) for `shell` for $p = 1$ and $p = 4$. The label "3.6ms" indicates the theoretical scaling limit for $L = 9$.

With Equation (9), one can approximate a latency $\Delta t^{\texttt{V-Cycle}}_{\text{Latency}} = 3.6$ ms for $L = 9$ and $\Delta t^{\texttt{vmult}}_{\text{Latency}} = 50\ \mu$s. This value is visualized with vertical lines in the diagrams. One can see that the approximation indeed matches well for $p = 1$, indicating that the solvers are limited by the latency in these ranges. Due to the similarity between the numbers of communication steps in the case of local smoothing and global coarsening, the minimal times reached are similar. For $p = 4$, the discrepancy between the approximated limit and the measurements is higher; the minimal times reached by global coarsening are smaller, indicating that the latency limit has not been reached yet and load imbalances still have a noticeable effect. For less refinements, e.g., $L = 7$, better correspondence can be observed. Apart from these observations, we emphasize that, in practice, one would run at higher efficiencies (left-top corner in the normalized plot), for which both $p = 1$ and $p = 4$ are dominated by load-imbalance effects.

5.5 First-child Policy as Alternative Partitioning Strategy for Global Coarsening

Section 5.3 indicates that local smoothing with first-child policy might suffer from deteriorated reduction rates of the maximum number of cells on each level; in particular, there might be processes without any cells, i.e., any work, increasing the critical path, although the vertical efficiency is optimal. In this section, we consider the first-child policy as an alternative for partitioning of the levels for global coarsening.

Figure 15 presents the timings of large-scale `octant` simulations for (1) local smoothing, (2) global coarsening with default partitioning, and (3) global coarsening with first-child policy for

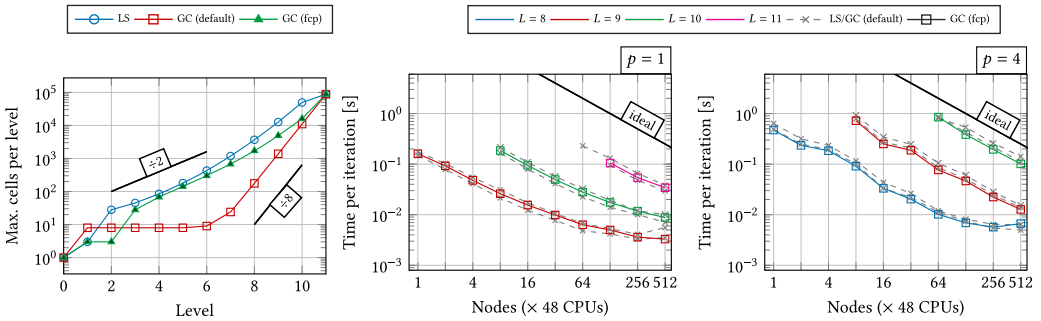


Fig. 15. Left: Maximum number of cells per level ($L = 11$; 256 nodes). Right: Strong scaling of global coarsening with first-child policy (fcp) in comparison to local smoothing (LS) and geometric global coarsening (GC) with default policy for `octant`.

$p = 1/p = 4$. The timings of the latter show similar trends as global coarsening with default partitioning and are lower than the ones of local smoothing. To explain this counterintuitive observation, Figure 15 provides the maximum number of cells on each multigrid level of the three approaches for $L = 11$ on 256 nodes. Since global coarsening with first-child policy does not perform any repartitioning, better reduction trends as in the local-smoothing case on all levels cannot be expected; however, one can observe that, for the local section of the refinement tree, the number of cells on the finest levels is reduced nearly as well as in the case of the default partitioner, i.e., the parallel workload and the parallel efficiency are not much worse; nonetheless, with higher vertical efficiency. This is not possible for the lower levels, but the behavior of the finest levels dominates the overall trends, since they take the largest time of the computation. One should note that local smoothing has access to the same cells but simply skips them during smoothing of a given level, missing the opportunity to reduce the problem size locally. In our experiments, it is not clear whether an optimal reduction of cells is crucial for all configurations: For $p = 1$, global coarsening with default partitioning is faster for most configurations (up to 20%) and, for $p = 4$, global coarsening with first-child policy is faster (up to 10%). We could trace this difference back to the different costs of the transfer, where, for $p = 4$, a reduced data transfer (both within the compute node and across the network) and, for $p = 1$, a better load balance is beneficial.

In the `shell` case (Figure 16), we observe that both global-coarsening partitioning strategies result in very similar reduction rates, which can be traced back to the fact that both strategies lead to comparable partitionings of the levels (see also Table 5, which shows a very high vertical efficiency of the default partitioning) and—as a consequence—to very similar solution times (±10%). Local smoothing only reduces the maximum number of cells per level optimally once all locally refined cells have been processed and the levels that had been constructed via global refinement have been reached. This case stresses the issue of load imbalances related to reduction rates significantly differing between processes.

6 PERFORMANCE ANALYSIS: P-MULTIGRID

In this section, we consider p-multigrid. The settings are as described in Section 5. The degrees on the levels are obtained by a bisection strategy. On the coarsest level ($p = 1$, fine mesh with hanging nodes), we run a single V-cycle of either a local-smoothing or a global-coarsening geometric multigrid solver in an *hp* context. In Appendix B, we present a comparison with state-of-the-art AMG solvers [31, 34] as coarse-grid solvers of p-multigrid on 16 nodes. The results show better timings in favor of geometric multigrid, which also turned out to be more robust with a single V-cycle.

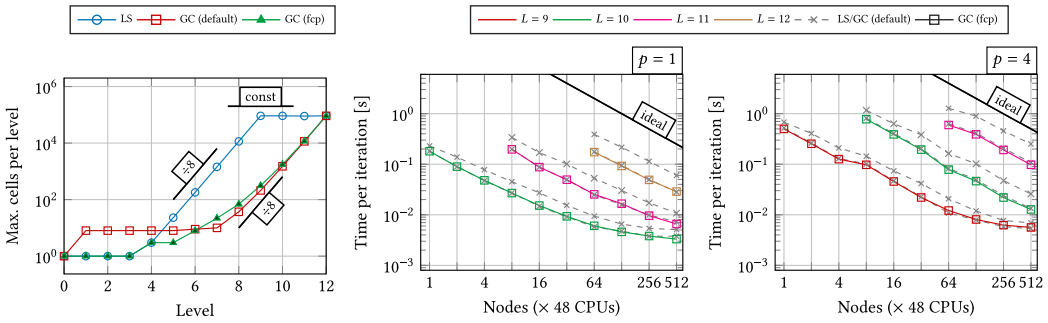


Fig. 16. Left: Maximum number of cells per level ($L = 12$; 256 nodes). Right: Strong scaling of global coarsening with first-child policy (fcp) in comparison to local smoothing (LS) and geometric global coarsening (GC) with default policy for `shell`.

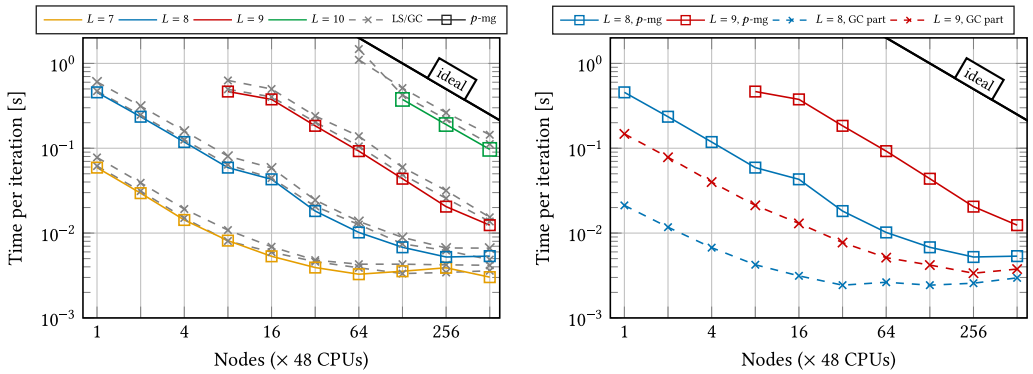


Fig. 17. Strong scaling of p-multigrid for `octant` for $p = 4$. p-multigrid switches to a global-coarsening coarse-grid solver immediately once linear elements have been reached. Left: Comparison with local smoothing (LS) and geometric global coarsening (GC) for $p = 4$. Right: Comparison with geometric global coarsening (GC) for $p = 1$ (coarse-grid problem of p-multigrid).

Figure 17 presents a strong-scaling comparison of the p-multigrid version of the global-coarsening algorithm with local-smoothing and global-coarsening variants of h-multigrid for the `octant` case for $p = 4$.[6] As we use a bisection strategy in the context of p-multigrid, the overall (hp-)multigrid algorithm has two additional levels compared to pure h-multigrid, given the same fine mesh, but with additional levels for $p = 1$ and $p = 2$. In our experiments, we observe that p-multigrid needs at least as many iterations as the pure (global-coarsening) h-multigrid algorithm (with small differences of at most 1). Since this might be related to the chosen problem and might be different on unstructured meshes as well as for problems with high-contrast coefficients and boundary layers, we consider "time per iteration" in the following.

One can see that one cycle of p-multigrid is 10–15% faster than h-multigrid for moderate numbers of processes. The reason for this is that the smoother application on the finest level is equally expensive; however, the transfer between the two finest levels is cheaper: Due to the same partitioning of these levels, the data are mainly transferred between cells that are locally owned on both

[6]The raw data for $L = 9$ is provided in Appendix B.

the coarse and the fine level. At the scaling limit (not shown), p-multigrid falls behind h-multigrid regarding performance. This is related to the increased latency due to a higher number of levels.

For the sake of completeness, Figure 17 includes the times spent in the geometric global-coarsening part, i.e., the timings for the V-cycle for $p = 1$ for $L = 8/L = 9$ as the coarse-grid problem of the p-multigrid solver. Due to the ~64× smaller size, its share is negligible for a wide range of nodes, but it becomes noticeable at the scaling limit.

7 APPLICATION: VARIABLE-VISCOSITY STOKES FLOW

We conclude this publication by presenting preliminary results of a practical application from geosciences by integrating the global-coarsening framework into the mantle convection code `ASPECT` [39, 61] and comparing it against the existing local-smoothing implementation [27].

We consider the variable-viscosity Stokes problem

$$\begin{aligned} -\nabla \cdot (2\eta\varepsilon(\mathbf{u})) + \nabla p &= f \\ \nabla \cdot \mathbf{u} &= 0 \end{aligned}$$

with a Q2-Q1 Taylor–Hood discretization of velocity $\vec{u}$, pressure p, and a given viscosity $\eta(x) > 0$, the symmetric gradient $\varepsilon(\mathbf{u}) = 1/2\nabla\mathbf{u} + 1/2(\nabla\mathbf{u})^T$, and forcing f. The resulting linear system

$$\begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} U \\ P \end{pmatrix} = \begin{pmatrix} F \\ 0 \end{pmatrix}$$

is solved with a Krylov method (in our tests using IDR(2), see Reference [27]) preconditioned using a block preconditioner,

$$P^{-1} = \begin{pmatrix} A & B^T \\ 0 & -S \end{pmatrix}^{-1},$$

where the Schur complement $S = BA^{-1}B^T$ is approximated using a mass matrix weighted by the viscosity. The inverses of the diagonal blocks of A and the Schur complement approximation $\hat{S}$ are each approximated by applying a single V-cycle of geometric multigrid using a Chebyshev smoother of degree 4 and implemented in a matrix-free fashion. Each IDR(2) iteration consists of three matrix-vector products and three preconditioner applications. Operator and preconditioner are set up to act on the complement of the nullspace containing the constant pressures. For a detailed description of the preconditioner see Reference [27].

We consider a 3D spherical shell benchmark problem called "nsinker_spherical_shell" that is part of `ASPECT`. A set of seven heavy sinkers is placed in a spherical shell with inner radius 0.54, outer radius 1.0, and no-flow boundary conditions. The flow is driven by the density difference of the sinkers ($\beta = 10$) and the gravity of magnitude 1 as follows, where c_i are the centers of the n sinkers[7]:

$$\begin{aligned} \Phi(x) &= \prod_{i=1}^{n} \left[1 - \exp\left(-\delta \max\left[0, |c_i - x| - \frac{\omega}{2} \right]^2 \right) \right], \\ \eta(x) &= \Phi(x)\eta_{\min} + (1 - \Phi(x))\eta_{\max}, \\ f(x) &= (0, 0, \beta(\Phi(x) - 1)), \end{aligned}$$

with $\delta = 200$, $\omega = 0.1$, $\eta_{\min} = 0.1$, and $\eta_{\max} = 10$. The viscosity is evaluated at the quadrature points of each cell on the finest level, averaged using harmonic averaging on each cell, and then interpolated to the coarser multigrid levels using the multigrid transfer operators (see also the function `interpolate_to_mg()` in Figure 6) for use in the multigrid preconditioner.

[7] Their locations are given in https://github.com/peterrum/dealii-multigrid/mantle-convection.

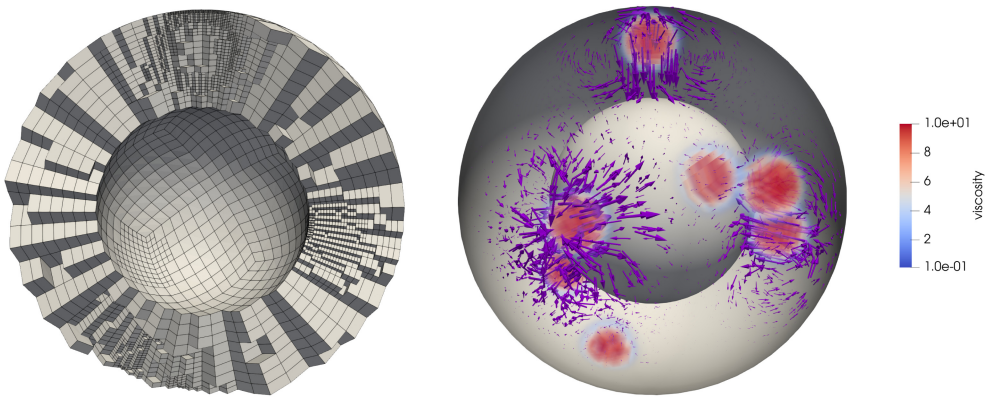


Fig. 18. Visualization of Stokes flow in a spherical shell: Adaptively refined mesh with two global and four additional local refinements (left); solution shown together with the high-viscosity sinkers in red and velocity vector field in purple (right).

Table 6. Performance for the Stokes-Flow Problem with 7,168 Processes with Number of Iterations (#it), Time of a Single Multigrid V-cycle of the Velocity Block, Total Solution Time, and Efficiency Metrics

		local smoothing						global coarsening					
L	#DoFs [1e6]	wl-eff	v-eff	h-eff	#it	solve [s]	V-cycle [s]	wl-eff	v-eff	h-eff	#it	solve [s]	V-cycle [s]
5	10.0	95%	93%	60%	24	1.11	0.011	77%	67%	61%	25	0.50	0.002
6	20.7	52%	96%	62%	25	1.54	0.006	77%	4%	63%	25	1.47	0.004
7	43.2	35%	97%	66%	27	3.20	0.010	86%	1%	66%	25	1.55	0.006
8	88.0	27%	98%	69%	27	4.94	0.021	91%	1%	69%	27	3.22	0.015
9	178.0	22%	99%	73%	28	9.81	0.046	95%	3%	73%	24	3.60	0.019
10	355.0	20%	100%	77%	29	15.36	0.104	97%	1%	76%	26	7.84	0.035
11	715.7	21%	100%	80%	27	26.38	0.217	99%	2%	80%	25	11.97	0.068
12	1441.4	20%	100%	83%	27	49.37	0.447	99%	2%	83%	26	21.77	0.140
13	2896.7	19%	100%	86%	27	107.24	0.980	100%	4%	86%	25	35.49	0.262
14	5861.9	21%	100%	89%	25	181.42	1.821	100%	5%	88%	30	78.46	0.518

The initial mesh (consisting of 96 coarse cells, 4 initial refinement steps, and a high-order manifold description) is refined adaptively using a gradient jump error estimator (see Reference [56]) of the velocity field roughly doubling the number of unknowns in each step, see Figure 18.

In the following, we compare local smoothing and global coarsening regarding number of iterations, time to solution, and time for a single V-cycle of the A block in Table 6. Computations are done on TACC Frontera[8] on 7,168 processes (128 nodes with 56 cores each). While iteration numbers are very similar, the simulation with global coarsening is about twice as fast in total. Each V-cycle is up to four times faster, which is not surprising, since the mesh has—for a high number of refinements—a workload efficiency of approximately 20%. The results suggest that the findings regarding iteration numbers and performance obtained in Section 5 for a simple Poisson problem are also applicable for this non-trivial vector-valued problem.

8 SUMMARY AND OUTLOOK

We have compared geometric local-smoothing and geometric global-coarsening multigrid implementations for locally refined meshes. They are based on optimal node-level performance through matrix-free operator evaluation and have been integrated into the same finite-element library (`deal.II`) to enable a fair comparison regarding implementation complexity and performance.

[8]Using deal.II v9.4.0 with 64bit indices, Intel 19.1.0.20200306 with -O3 and -march=native.

From the implementation point of view, the two multigrid versions are—except for some subtle differences—similar, requiring special treatment of either refinement edges or hanging-node constraints during the application of the smoother and the transfer operator. For the latter, one can usually rely on existing infrastructure for the application of constraints [62, 81]. During transfer, global coarsening needs to transfer between cells that might be refined or not. To be able to vectorize the cell-local transfer, we categorize cells within the transfer operator and process categories in one go. In the case of local smoothing, it is possible but not common to repartition the multigrid levels; instead, one uses local strategies for partitioning. For global coarsening, we investigated two partitioning strategies: one that optimally balances workload during smoothing via repartitioning each level and one that minimizes the data to be communicated during the transfer phase.

In a large number of experiments, we have made the observation that, for serial simulations, geometric local smoothing is faster than geometric global coarsening (if the number of iterations is the same), since the total number of cells to perform smoothing on is smaller and the need for evaluating hanging-node constraints on each level might be noticeable. For parallel simulations, an equally distributed reduction of cells is beneficial. If this is not given, then load imbalance is introduced and the critical path of cells, i.e., the time to solution, is increased. In the case of local smoothing, there might be a non-negligible number of processes without cells, naturally introducing load imbalance already on the finest—computationally most expensive—levels. Global coarsening with repartitioning alleviates this problem and reaches optimal parallel workload. It comes, however, with the disadvantage of expensive transfer steps due to permutation of the data. We made the observation that global coarsening with non-optimal partitions for smoothing but with local transfer allows us—for the examples considered—to reduce the number of cells on the finest levels surprisingly well, introducing only a small load imbalance. We could not make a definite statement on the choice of partitioning strategies for global coarsening, since it very much depends on whether the transfer or the load imbalance is the bottleneck for the given problem. Regardless of the type of partitioning of the levels, we have shown that global coarsening outperforms local smoothing by up a factor of 2.4 in terms of time to solution for both moderately large and large-scale simulations.

We have also considered polynomial global coarsening (p-multigrid). Its implementation is conceptually analogous to the one of geometric global coarsening with similar involved algorithms. Far from the scaling limit, p-multigrid shows better timings per iteration, which can be explained by the cheaper intergrid transfer. At the scaling limit, the introduction of additional multigrid levels (when combining h and p multigrid) is noticeable and leads to slower times due to latency. Generally, when p- and h-multigrid are used in sequence (hp-multigrid), reducing the degree p first makes sense from a performance point of view due to cheaper transfer as long as the number of iterations does not increase.

Global-coarsening algorithms also give the possibility to easily remove ranks from MPI communicators, allowing us to increase the granularity of the problem on each level. Once the problem size can no longer be reduced by a sufficient factor, one can switch to the coarse-grid solver even on a level with hanging nodes. Furthermore, h and p global coarsening could be done in one go: While this might lead to an overly aggressive coarsening with a consequential deterioration of the convergence rate, the reduced number of levels could result in an improved strong-scaling behavior due to a reduced latency in some cases. The investigation of these topics is deferred to future work.

In this publication, we focused on geometrically refined meshes consisting only of hexahedral shaped cells, where all cells have the same polynomial degree p. However, the presented algorithms also work for p- and hp-adaptive problems, where the polynomial degree varies for each cell, as well as for simplex and mixed meshes as the implementation in `deal.II` [7] shows. Moreover, they are—as demonstrated in Reference [60]—applicable for auxiliary-space approximation for DG, too.

APPENDICES

A APPENDIX TO SECTION 5

Table 7. Number of Iterations and Time to Solution for Local Smoothing (LS) and Global Coarsening (GC) for the octant Simulation case with 768 Processes (16 Nodes) with Given Analytical Solution: $u(\vec{x}) = (\frac{1}{\alpha\sqrt{2\pi}})^3 \exp(-||\vec{x} - \vec{x}_0||/\alpha^2)$ with $\vec{x}_0 = (-0.5, -0.5, -0.5)^T$ and $\alpha = 0.1$

	$p = 1$				$p = 4$			
	LS		GC		LS		GC	
L	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]
3	4	1.4e-3	4	1.0e-3	4	2.9e-3	4	2.9e-3
4	4	2.8e-3	4	1.7e-3	4	4.7e-3	4	4.2e-3
5	4	4.0e-3	4	2.7e-3	4	6.9e-3	4	6.5e-3
6	4	5.7e-3	4	4.2e-3	4	1.2e-2	4	1.0e-2
7	4	8.2e-3	4	6.1e-3	4	2.8e-2	4	2.4e-2
8	5	2.2e-2	4	1.2e-2	5	2.8e-1	4	1.7e-1
9	5	8.8e-2	4	5.2e-2	5	2.5e+0	4	1.6e+0
10	5	6.1e-1	4	3.8e-1	—	—	—	—

Right-hand-side function $f(\vec{x})$ and inhomogeneous Dirichlet boundary conditions have been selected appropriately.

Table 8. Number of Iterations and Time to Solution for Local Smoothing, Global Coarsening, and AMG (ML [34]) for the Octant Simulation Case with 768 Processes (16 Nodes)

	$p = 1$										$p = 4$							
	local smoothing				global coarsening				AMG (ML)		local smoothing				global coarsening			
	$k = 3$		$k = 6$		$k = 3$		$k = 6$				$k = 3$		$k = 6$		$k = 3$		$k = 6$	
L	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]
3	4	1.3e-3	3	1.4e-3	4	1.0e-3	2	7.0e-4	1	6.0e-3	4	2.6e-3	3	2.5e-3	4	2.5e-3	2	1.9e-3
4	4	2.8e-3	3	2.9e-3	4	1.7e-3	2	1.3e-3	1	2.5e-2	4	4.3e-3	3	4.6e-3	4	4.0e-3	2	3.0e-3
5	4	4.3e-3	3	4.6e-3	4	2.6e-3	2	2.3e-3	5	4.2e-3	4	6.1e-3	3	6.6e-3	3	4.9e-3	2	5.1e-3
6	4	5.8e-3	3	6.4e-3	4	4.3e-3	2	3.4e-3	8	1.5e-2	4	9.9e-3	3	1.1e-2	3	7.2e-3	2	7.1e-3
7	4	8.2e-3	3	9.3e-3	3	4.5e-3	2	4.9e-3	6	1.2e-2	4	2.7e-2	3	3.0e-2	3	1.8e-2	2	1.7e-2
8	4	1.7e-2	3	2.1e-2	3	9.1e-3	2	1.0e-2	6	3.0e-2	4	2.3e-1	3	2.6e-1	3	1.4e-1	2	1.4e-1
9	4	7.1e-2	3	8.6e-2	3	3.8e-2	2	4.3e-2	7	2.1e-1	4	2.0e+0	3	2.3e+0	3	1.2e+0	2	1.3e+0
10	4	5.0e-1	3	6.1e-1	2	1.9e-1	2	3.0e-1	7	1.6e+0	—	—	—	—	—	—	—	—

For local smoothing and global coarsening, results are shown for *Chebyshev smoothing degrees* of $k = 3$ and $k = 6$.

Table 9. Number of Iterations and Time to Solution for Local Smoothing (LS) and Global Coarsening (GC) for the Octant Simulation Case with 768 Processes (16 Nodes)

	double								float							
	$p = 1$				$p = 4$				$p = 1$				$p = 4$			
	LS		GC		LS		GC		LS		GC		LS		GC	
L	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]
3	4	1.3e-3	4	9.0e-4	4	2.6e-3	4	2.7e-3	4	1.3e-3	4	1.0e-3	4	2.4e-3	4	2.4e-3
4	4	2.8e-3	4	1.6e-3	4	4.5e-3	4	4.1e-3	4	2.7e-3	4	1.7e-3	4	4.1e-3	4	4.1e-3
5	4	4.0e-3	4	2.6e-3	4	6.6e-3	3	5.1e-3	4	4.0e-3	4	2.6e-3	4	6.1e-3	3	4.8e-3
6	4	7.8e-3	4	4.0e-3	4	1.1e-2	3	9.6e-3	4	6.0e-3	4	4.1e-3	4	1.1e-2	3	7.3e-3
7	4	8.9e-3	3	4.9e-3	4	3.8e-2	3	2.7e-2	4	8.3e-3	3	4.8e-3	4	2.7e-2	3	1.8e-2
8	4	2.1e-2	3	1.0e-2	4	4.0e-1	3	2.4e-1	4	1.7e-2	3	9.2e-3	4	2.3e-1	3	1.4e-1
9	4	9.9e-2	3	5.1e-2	4	3.2e+0	3	1.9e+0	4	7.1e-2	3	3.9e-2	4	2.0e+0	3	1.2e+0
10	4	7.7e-1	2	2.8e-1	—	—	—	—	4	5.0e-1	2	1.9e-1	—	—	—	—

All operations in the outer CG solver are run with double-precision floating-point numbers and in the multigrid V-cycle with the following *multigrid number types*: single- or double-precision floating-point numbers.

Table 10. Number of Iterations for Local Smoothing (LS) and Global Coarsening (GC) for the octant Simulation with 768 Processes (16 Nodes) and for Different *Global Relative Solver Tolerances*

	10^{-4}				10^{-6}				10^{-8}				10^{-10}			
	$p=1$		$p=4$		$p=1$		$p=4$		$p=1$		$p=4$		$p=1$		$p=4$	
L	LS	GC	LS	GC	LS	GC	LS	GC	LS	GC	LS	GC	LS	GC	LS	GC
3	4	4	4	4	6	5	6	6	7	6	8	7	9	8	10	9
4	4	4	4	4	6	6	6	5	8	7	8	7	10	9	10	9
5	4	4	4	3	6	5	6	5	8	7	8	7	10	8	10	8
6	4	4	4	3	6	5	6	5	8	7	8	6	10	8	10	8
7	4	3	4	3	6	5	6	4	8	6	8	6	10	8	10	8
8	4	3	4	3	6	4	6	4	8	6	8	6	9	8	10	7
9	4	3	4	3	6	4	6	4	8	6	8	6	9	7	10	7
10	4	2	—	—	6	4	—	—	8	6	—	—	9	7	—	—

Table 11. Time to Solution for Global Coarsening for the Octant Simulation *with 768 Processes* (16 Nodes) and *Different Cell Weights* for Cells Near Hanging Nodes Compared to Regular Cells

	$p=1$					$p=4$				
L/w	1.0	1.5	2.0	2.5	3.0	1.0	1.5	2.0	2.5	3.0
3	1.1e-3	1.0e-3	1.0e-3	1.0e-3	1.0e-3	2.7e-3	2.6e-3	2.6e-3	2.5e-3	2.5e-3
4	1.8e-3	1.8e-3	1.8e-3	1.7e-3	1.7e-3	3.9e-3	3.9e-3	3.9e-3	3.9e-3	3.9e-3
5	2.5e-3	2.7e-3	2.7e-3	2.6e-3	2.8e-3	4.4e-3	4.7e-3	4.6e-3	4.5e-3	4.6e-3
6	4.0e-3	4.1e-3	4.1e-3	4.2e-3	4.1e-3	7.3e-3	7.1e-3	7.0e-3	7.1e-3	7.0e-3
7	4.8e-3	4.8e-3	4.7e-3	4.4e-3	5.4e-3	1.8e-2	1.7e-2	1.6e-2	1.7e-2	1.8e-2
8	1.1e-2	1.0e-2	9.1e-3	8.5e-3	8.2e-3	1.1e-1	1.0e-1	1.0e-1	1.0e-1	1.0e-1
9	5.3e-2	4.4e-2	3.8e-2	3.5e-2	3.5e-2	9.0e-1	8.2e-1	7.9e-1	7.8e-1	7.9e-1
10	2.3e-1	1.9e-1	1.7e-1	1.5e-1	1.5e-1	-	-	-	-	-

Table 12. Time to Solution for Global Coarsening for the Octant Simulation with *24,576 Processes* (512 Nodes) and *Different Cell Weights* for Cells Near Hanging Nodes Compared to Regular Cells

	$p=1$					$p=4$				
L/w	1.0	1.5	2.0	2.5	3.0	1.0	1.5	2.0	2.5	3.0
3	1.4e-3	1.3e-3	1.4e-3	1.6e-3	1.4e-3	3.1e-3	3.0e-3	3.0e-3	2.9e-3	2.9e-3
4	2.1e-3	2.2e-3	2.1e-3	2.0e-3	2.1e-3	4.3e-3	4.4e-3	4.3e-3	4.3e-3	4.3e-3
5	2.9e-3	2.9e-3	2.9e-3	2.9e-3	2.9e-3	4.6e-3	5.0e-3	4.6e-3	5.1e-3	4.7e-3
6	4.5e-3	4.1e-3	4.1e-3	4.2e-3	4.1e-3	7.0e-3	7.0e-3	7.1e-3	6.9e-3	6.6e-3
7	5.9e-3	4.8e-3	5.7e-3	5.1e-3	5.6e-3	1.1e-2	1.1e-2	1.1e-2	1.1e-2	1.0e-2
8	7.6e-3	9.0e-3	7.9e-3	9.2e-3	8.9e-3	1.5e-2	1.5e-2	1.6e-2	1.5e-2	1.4e-2
9	1.2e-2	1.1e-2	1.1e-2	1.1e-2	1.0e-2	4.9e-2	4.2e-2	4.3e-2	4.2e-2	4.2e-2
10	1.8e-2	1.5e-2	1.4e-2	1.3e-2	1.3e-2	3.7e-1	3.4e-1	3.4e-1	3.4e-1	3.4e-1
11	7.6e-2	6.8e-2	5.7e-2	5.4e-2	5.3e-2	-	-	-	-	-

Table 13. Number of Iterations and Time to Solution for Local Smoothing (LS) and Global Coarsening (GC) for a Uniformly Refined Mesh of a Cube (*without Hanging Nodes*) with 768 Processes (16 Nodes)

	$p=1$				$p=4$			
	LS		GC		LS		GC	
L	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]
3	4	1.80e-3	4	1.80e-3	4	2.90e-3	4	3.20e-3
4	4	2.50e-3	4	2.40e-3	4	3.90e-3	4	4.00e-3
5	4	3.30e-3	4	3.20e-3	4	5.90e-3	4	6.40e-3
6	4	4.60e-3	4	5.40e-3	4	1.61e-2	4	1.69e-2
7	4	9.00e-3	4	9.00e-3	4	1.29e-1	4	1.29e-1
8	4	4.18e-2	4	4.21e-2	4	1.11e+0	4	1.10e+0
9	4	3.00e-1	4	2.98e-1	—	—	—	—

B APPENDIX TO SECTION 6

Table 14. Number of Iterations and Time to Solution for Local Smoothing (LS), Global Coarsening (GC), and AMG (ML [34] and BoomerAMG [31]) as *Coarse-grid Solver* of p-multigrid for the `octant` Simulation with 768 Processes (16 Nodes) for $p = 4$

	LS		GC		AMG (ML)								AMG (BoomerAMG)	
					$\#v$=1		$\#v$=2		$\#v$=3		$\#v$=4		$\#v$=1	
L	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]
3	4	2.60e-3	4	2.20e-3	4	1.70e-3	4	2.00e-3	4	2.20e-3	4	2.50e-3	4	1.70e-3
4	4	5.10e-3	4	3.90e-3	4	5.40e-3	4	8.30e-3	4	1.11e-2	4	1.39e-2	4	4.90e-3
5	4	7.60e-3	4	6.10e-3	4	1.17e-2	4	1.95e-2	4	2.74e-2	4	3.52e-2	4	1.17e-2
6	4	1.11e-2	4	9.20e-3	5	2.37e-2	4	3.24e-2	4	4.45e-2	4	5.96e-2	5	3.60e-2
7	4	2.35e-2	4	2.11e-2	6	4.70e-2	5	5.47e-2	4	5.59e-2	4	6.72e-2	6	1.04e-1
8	4	1.81e-1	4	1.72e-1	7	3.26e-1	5	2.65e-1	4	2.37e-1	4	2.63e-1	7	4.76e-1
9	4	1.53e+0	4	1.51e+0	9	3.54e+0	7	2.97e+0	6	2.75e+0	5	2.46e+0	8	3.83e+0

For AMG, different numbers of V-cycles $\#v$ are investigated. AMG parameters used are shown in Appendix C.

Table 15. Number of Iterations and Time of a Single Iteration for h-multigrid (Local Smoothing (LS), Global Coarsening (GC)), and p-multigrid (Local Smoothing or Global Coarsening as Coarse-grid Solver) for $L = 9$ and $p = 4$

	h-mg				p-mg			
	LS		GC		LS		GC	
#nodes	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]
8	4	6.30e-1	3	4.95e-1	4	4.75e-1	4	4.66e-1
16	4	4.99e-1	3	4.03e-1	4	3.83e-1	4	3.77e-1
32	4	2.40e-1	3	2.03e-1	4	1.87e-1	4	1.84e-1
64	4	1.39e-1	3	1.05e-1	4	9.57e-2	4	9.25e-2
128	4	5.97e-2	3	4.90e-2	4	4.49e-2	4	4.37e-2
256	4	3.17e-2	3	2.57e-2	4	2.35e-2	4	2.05e-2
512	4	1.54e-2	3	1.37e-2	4	1.38e-2	4	1.24e-2

C AMG PARAMETERS

```
Teuchos::ParameterList parameter_list;
ML_Epetra::SetDefaults("SA", parameter_list);

parameter_list.set("smoother: type", "ILU");
parameter_list.set("coarse: type", coarse_type);
parameter_list.set("initialize random seed", true);
parameter_list.set("smoother: sweeps", 1);
parameter_list.set("cycle applications", 2);
parameter_list.set("prec type", "MGV");
parameter_list.set("smoother: Chebyshev alpha", 10.);
parameter_list.set("smoother: ifpack overlap", 0);
parameter_list.set("aggregation: threshold", 1e-4);
parameter_list.set("coarse: max size", 2000);
```

Listing 1. ML [34] (Trilinos 12.12.1).

```
PCHYPRESetType(pc, "boomeramg");

set_option_value("-pc_hypre_boomeramg_agg_nl", "2");
set_option_value("-pc_hypre_boomeramg_max_row_sum", "0.9");
set_option_value("-pc_hypre_boomeramg_strong_threshold", "0.5");
set_option_value("-pc_hypre_boomeramg_relax_type_up", "SOR/Jacobi");
set_option_value("-pc_hypre_boomeramg_relax_type_down", "SOR/Jacobi");
set_option_value("-pc_hypre_boomeramg_relax_type_coarse", "Gaussian-elimination");
set_option_value("-pc_hypre_boomeramg_grid_sweeps_coarse", "1");
set_option_value("-pc_hypre_boomeramg_tol", "0.0");
set_option_value("-pc_hypre_boomeramg_max_iter", "2"));
```

Listing 2. BoomerAMG [31] (PETSc 3.14.5).

ACKNOWLEDGMENTS

The authors acknowledge collaboration with Maximilian Bergbauer, Thomas C. Clevenger, Ivo Dravins, Niklas Fehn, Marc Fehling, and Magdalena Schreter as well as the `deal.II` community.

REFERENCES

[1] Mark Adams. 2002. Evaluation of three unstructured multigrid methods on 3D finite element problems in solid mechanics. *Int. J. Numer. Methods Eng.* 55, 5 (2002), 519–534.

[2] Mark Adams, Marian Brezina, Jonathan Hu, and Ray Tuminaro. 2003. Parallel multigrid smoothing: Polynomial versus Gauss–Seidel. *J. Comput. Phys.* 188, 2 (2003), 593–610.

[3] Mark Adams, Phillip Colella, Daniel T. Graves, Jeff N. Johnson, Noel D. Keen, Terry J. Ligocki, Daniel F. Martin, Peter W. McCorquodale, David Modiano, Peter O. Schwartz, T.D. Sternberg, and Brian Van Straalen. 2014. *Chombo Software Package for AMR Applications Design Document.* Lawrence Berkeley National Laboratory Technical Report LBNL-6616E (2014).

[4] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, Will Pazner, Mark Stowell, Vladimir Tomov, Ido Akkerman, Johann Dahm, David Medina, and Stefano Zampini. 2021. MFEM: A modular finite element methods library. *Comput. Math. Appl.* 81, 1 January (2021), 42–74. DOI: http://dx.doi.org/10.1016/j.camwa.2020.06.009

[5] Paola F. Antonietti, Marco Sarti, Marco Verani, and Ludmil T. Zikatanov. 2017. A uniform additive Schwarz preconditioner for high-order discontinuous Galerkin approximations of elliptic problems. *J. Sci. Comput.* 70, 2 (2017), 608–630.

[6] Daniel Arndt, Wolfgang Bangerth, Bruno Blais, Thomas C. Clevenger, Marc Fehling, Alexander V. Grayver, Timo Heister, Luca Heltai, Martin Kronbichler, Matthias Maier, Peter Munch, Jean-Paul Pelteret, Reza Rastak, Ignacio Tomas, Bruno Turcksin, Zhuoran Wang, and David Wells. 2020. The deal.II library, version 9.2. *J. Numer. Math.* 28, 3 (2020), 131–146. https://doi.org/10.1515/jnma-2020-0043

[7] Daniel Arndt, Wolfgang Bangerth, Bruno Blais, Marc Fehling, Rene Gassmöller, Timo Heister, Luca Heltai, Uwe Köcher, Martin Kronbichler, Matthias Maier, Peter Munch, Jean-Paul Pelteret, Sebastian Proell, Konrad Simon, Bruno Turcksin, David Wells, and Jiaqi Zhang. 2021. The deal.II library, version 9.3. *J. Numer. Math.* 29, 3 (2021), 171–186. https://doi.org/10.1515/jnma-2021-0081

[8] Daniel Arndt, Wolfgang Bangerth, Denis Davydov, Timo Heister, Luca Heltai, Martin Kronbichler, Matthias Maier, Jean-Paul Pelteret, Bruno Turcksin, and David Wells. 2021. The deal.II finite element library: Design, features, and insights. *Comput. Math. Appl.* 81, 1 (2021), 407–422. https://doi.org/10.1016/j.camwa.2020.02.022

[9] Daniel Arndt, Wolfgang Bangerth, Marco Feder, Marc Fehling, Rene Gassmöller, Timo Heister, Luca Heltai, Martin Kronbichler, Matthias Maier, Peter Munch, Jean-Paul Pelteret, Simon Sticko, Bruno Turcksin, and David Wells. 2022. The `deal.II` Library, Version 9.4. *J. Numer. Math.* 30, 3 (2022), 231–246. https://doi.org/10.1515/jnma-2022-0054

[10] Daniel Arndt, Niklas Fehn, Guido Kanschat, Katharina Kormann, Martin Kronbichler, Peter Munch, Wolfgang A. Wall, and Julius Witte. 2020. ExaDG: High-order discontinuous Galerkin for the exa-scale. In *Software for Exascale Computing—SPPEXA 2016–2019*, Hans-Joachim Bungartz, Severin Reiz, Benjamin Uekermann, Philipp Neumann, and Wolfgang E. Nagel (Eds.). Springer International Publishing, Cham, 189–224.

[11] Douglas Arnold, Richard Falk, and Ragnar Winther. 1997. Preconditioning in H(div) and applications. *Math. Comput.* 66, 219 (1997), 957–984.

[12] Harold L. Atkins and Brian T. Helenbrook. 2005. Numerical evaluation of p-multigrid method for the solution of discontinuous Galerkin discretizations of diffusive equations. In *Proceedings of the 17th AIAA Computational Fluid Dynamics Conference.* American Institute of Aeronautics and Astronautics, 1–11. https://doi.org/10.2514/6.2005-5110

[13] Eugenio Aulisa, Sara Calandrini, and Giacomo Capodaglio. 2018. An improved multigrid algorithm for n-irregular meshes with subspace correction smoother. *Comput. Math. Appl.* 76, 3 (2018), 620–632.

[14] Eugenio Aulisa, Giacomo Capodaglio, and Guoyi Ke. 2019. Construction of h-refined continuous finite element spaces with arbitrary hanging node configurations and applications to multigrid algorithms. *SIAM J. Sci. Comput.* 41, 1 (2019), A480–A507.

[15] Wolfgang Bangerth, Carsten Burstedde, Timo Heister, and Martin Kronbichler. 2011. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Trans. Math. Softw.* 38, 2 (2011), 14:1–28. https://doi.org/10.1145/2049673.2049678

[16] Wolfgang Bangerth and Oliver Kayser-Herold. 2009. Data structures and requirements for hp finite element software. *ACM Trans. Math. Softw.* 36, 1 (2009), 4:1–31.

[17] Francesco Bassi, Antonio Ghidoni, Stefano Rebay, and Pietro Tesini. 2009. High-order accurate p-multigrid discontinuous Galerkin solution of the Euler equations. *Int. J. Num. Methods iFluids* 60, 8 (2009), 847–865.

[18] Francesco Bassi and Stefano Rebay. 2003. Numerical solution of the Euler equations with a multiorder discontinuous finite element method. In *Computational Fluid Dynamics 2002*, S. W. Armfield (Ed.). Springer, Berlin, 199–204. https://doi.org/10.1007/978-3-642-59334-5_27

[19] Peter Bastian and Christian Wieners. 2006. Multigrid methods on adaptively refined grids. *Comput. Sci. Eng.* 8, 6 (2006), 44–54.
[20] Roland Becker and Malte Braack. 2000. Multigrid techniques for finite elements on locally refined meshes. *Numer. Lin. Algebr. Appl.* 7, 6 (2000), 363–379.
[21] Roland Becker, Malte Braack, and Thomas Richter. 2007. Parallel multigrid on locally refined meshes. In *Reactive Flows, Diffusion and Transport*, Willi Jäger, Rolf Rannacher, and Jürgen Warnatz (Eds.). Springer, Berlin, 77–92.
[22] Marco L. Bittencourt, Craig C. Douglas, and Raúl A. Feijóo. 2001. Nonnested multigrid methods for linear problems. *Numer. Methods Partial Differ. Eq.: Int. J.* 17, 4 (2001), 313–331.
[23] James H. Bramble, Joseph E. Pasciak, and Jinchao Xu. 1991. The analysis of multigrid algorithms with nonnested spaces or noninherited quadratic forms. *Math. Comp.* 56, 193 (1991), 1–34.
[24] Achi Brandt. 1977. Multi-level adaptive solutions to boundary-value problems. *Math. Comput.* 31, 138 (1977), 333–390.
[25] Jed Brown. 2010. Efficient nonlinear solvers for nodal high-order finite elements in 3D. *J. Sci. Comput.* 45, 1-3 (2010), 48–63. https://doi.org/10.1007/s10915-010-9396-8
[26] Carsten Burstedde, Lucas C. Wilcox, and Omar Ghattas. 2011. p4est : Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM J. Sci. Comput.* 33, 3 (2011), 1103–1133. https://doi.org/10.1137/100791634
[27] Thomas C. Clevenger and Timo Heister. 2021. Comparison between algebraic and matrix-free geometric multigrid for a stokes problem on adaptive meshes with variable viscosity. *Numer. Lin. Algebr. Appl.* 28, 5 (2021), e2375. https://doi.org/10.1002/nla.2375
[28] Thomas C. Clevenger and Timo Heister. 2021. Comparison between algebraic and matrix-free geometric multigrid for a Stokes problem on adaptive meshes with variable viscosity. *Numerical Linear Algebra with Applications* 28, 5 (2021), e2375:1–13. DOI: http://dx.doi.org/10.1002/nla.2375
[29] David Darmofal and Krzysztof Fidkowski. 2004. Development of a higher-order solver for aerodynamic applications. In *Proceedings of the 42nd AIAA Aerospace Sciences Meeting and Exhibit*. American Institute of Aeronautics and Astronautics, 1–12. https://doi.org/10.2514/6.2004-436
[30] Michel O. Deville, Paul F. Fischer, and Ernest H. Mund. 2002. *High-order Methods for Incompressible Fluid Flow*. Vol. 9. Cambridge University Press, Cambridge.
[31] Robert D. Falgout, Jim E. Jones, and Ulrike Meier Yang. 2006. The design and implementation of hypre, a library of parallel high performance preconditioners. In *Numerical Solution of Partial Differential Equations on Parallel Computers*, Are M. Bruaset and Aslak Tveito (Eds.). Springer, Berlin, 267–294.
[32] Niklas Fehn, Peter Munch, Wolfgang A. Wall, and Martin Kronbichler. 2020. Hybrid multigrid methods for high-order discontinuous Galerkin discretizations. *J. Comput. Phys.* 415 (2020), 109538. https://doi.org/10.1016/j.jcp.2020.109538
[33] Krzysztof J. Fidkowski, Todd A. Oliver, James Lu, and David L. Darmofal. 2005. p-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible navier-stokes equations. *J. Comput. Phys.* 207, 1 (2005), 92–113. https://doi.org/10.1016/j.jcp.2005.01.005
[34] Michael W. Gee, Christopher M. Siefert, Jonathan J. Hu, Ray S. Tuminaro, and Marzio G. Sala. 2006. *ML 5.0 Smoothed Aggregation User's Guide*. Technical Report SAND2006-2649, Sandia National Laboratories, Albuquerque.
[35] Antonio Ghidoni, Alessandro Colombo, Francesco Bassi, and Stefano Rebay. 2014. Efficient p-multigrid discontinuous Galerkin solver for complex viscous flows on stretched grids. *International Journal for Numerical Methods in Fluids* 75, 2 (2014), 134–154. DOI: http://dx.doi.org/10.1002/fld.3888
[36] Amir Gholami, Dhairya Malhotra, Hari Sundar, and George Biros. 2016. FFT, FMM, or Multigrid? A comparative study of state-of-the-art poisson solvers for uniform and nonuniform grids in the unit cube. *SIAM J. Sci. Comput.* 38, 3 (2016), C280–C306. https://doi.org/10.1137/15M1010798
[37] Xian-Zhong Guo and I. Norman Katz. 1998. Performance enhancement of the multi-p preconditioner. *Comput. Math. Appl.* 36, 4 (1998), 1–8.
[38] Xian-Zhong Guo and I. Norman Katz. 2000. A parallel multi-p method. *Comput. Math. Appl.* 39, 9-10 (2000), 115–123.
[39] Timo Heister, Juliane Dannberg, Rene Gassmöller, and Wolfgang Bangerth. 2017. High accuracy mantle convection simulation through modern numerical methods. II: Realistic models and problems. *Geophys. J. Int.* 210, 2 (2017), 833–851. https://doi.org/10.1093/gji/ggx195
[40] Brian T. Helenbrook and Harold L. Atkins. 2006. Application of p-multigrid to discontinuous Galerkin formulations of the poisson equation. *AIAA J.* 44, 3 (2006), 566–575. https://doi.org/10.2514/1.15497
[41] Brian T. Helenbrook and Harold L. Atkins. 2008. Solving discontinuous Galerkin formulations of poisson's equation using geometric and p multigrid. *AIAA J.* 46, 4 (2008), 894–902. https://doi.org/10.2514/1.31163
[42] Brian T. Helenbrook and Harold L. Atkins. 2008. Solving discontinuous Galerkin formulations of poisson's equation using geometric and p multigrid. *AIAA J.* 46, 4 (2008), 894–902.
[43] Brian T. Helenbrook and Brendan S. Mascarenhas. 2016. Analysis of implicit time-advancing p-multigrid schemes for discontinuous Galerkin discretizations of the Euler equations. In *Proceedings of the 46th AIAA Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics. https://doi.org/10.2514/6.2016-3494

[44] Brian T. Helenbrook, Dimitri Mavriplis, and Harold L. Atkins. 2003. Analysis of "p"-multigrid for continuous and discontinuous finite element discretizations. In *Proceedings of the 16th AIAA Computational Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics. https://doi.org/10.2514/6.2003-3989

[45] Magnus Rudolph Hestenes and Eduard Stiefel. 1952. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards* 49, 6 (1952), 409–436.

[46] Koen Hillewaert, Jean-François Remacle, Nicolas Cheveaugeon, Paul-Emile Bernard, and Philippe Geuzaine. 2006. Analysis of a hybrid p-multigrid method for the discontinuous Galerkin discretisation of the Euler equations. In *Proceedings of the European Conference on Computational Fluid Dynamics*, Pieter Wesseling, Eugenio Oñate, and Jacques Périaux (Eds.). ECCOMAS CFD 2006, Egmond aan Zee.

[47] Torsten Hoefler, Christian Siebert, and Andrew Lumsdaine. 2010. Scalable communication protocols for dynamic sparse data exchange. *ACM SIGPLAN Not.* 45, 5 (2010), 159–168.

[48] Ning Hu, Xian-Zhong Guo, and I. Norman Katz. 1997. Multi-p preconditioners. *SIAM J. Sci. Comput.* 18, 6 (1997), 1676–1697.

[49] Ning Hu and I. Norman Katz. 1995. Multi-p methods: iterative algorithms for the p-version of the finite element analysis. *SIAM J. Sci. Comput.* 16, 6 (1995), 1308–1332.

[50] Oleg Iliev and Dimitar Stoyanov. 2001. Multigrid-adaptive local refinement solver for incompressible flows. In *International Conference on Large-Scale Scientific Computing*, Svetozar Margenov, Jerzy Waśniewski, and Plamen Yalamov (Eds.). Springer, Berlin, 361–368.

[51] Bärbel Janssen and Guido Kanschat. 2011. Adaptive multilevel methods with local smoothing for Hˆ1-and Hˆcurl-conforming high order finite element methods. *SIAM J. Sci. Comput.* 33, 4 (2011), 2095–2114.

[52] Zhenhua Jiang, Chao Yan, Jian Yu, and Wu Yuan. 2015. Practical aspects of p-multigrid discontinuous Galerkin solver for steady and unsteady RANS simulations. *Int. J. Numer. Methods Fluids* 78, 11 (2015), 670–690. https://doi.org/10.1002/fld.4035

[53] Jean-Christophe Jouhaud, Marc Montagnac, and Loïc P. Tourrette. 2005. A multigrid adaptive mesh refinement strategy for 3D aerodynamic design. *Int. J. Numer. Methods Fluids* 47, 5 (2005), 367–385.

[54] Guido Kanschat. 2004. Multilevel methods for discontinuous Galerkin FEM on locally refined meshes. *Comput. Struct.* 82, 28 (2004), 2437–2445.

[55] Guido Kanschat and Youli Mao. 2015. Multigrid methods for hdiv-conforming discontinuous Galerkin methods for the stokes equations. *J. Numer. Math.* 23, 1 (2015), 51–66.

[56] Donald W. Kelly, Jorge P. De S. R. Gago, Olgierd Zienkiewicz, and Ivo Babuska. 1983. A posteriori error analysis and adaptive processes in the finite element method: Part I–error analysis. *Int. J. Numer. Methods Eng.* 19, 11 (1983), 1593–1619. https://doi.org/10.1002/nme.1620191103

[57] Tzanio Kolev, Paul Fischer, Misun Min, Jack Dongarra, Jed Brown, Veselin Dobrev, Tim Warburton, Stanimire Tomov, Mark S Shephard, Valeria Abdelfattah, Ahmad Barra, Natalie Beams, Jean-Sylvain Camier, Noel Chalmers, Yohann Dudouit, Ali Karakus, Ian Karlin, Stefan Kerkemeier, Yu-Hsiang Lan, David Medina, Elia Merzari, Aleksandr Obabko, Will Pazner, Thilina Rathnayake, Cameron W Smith, Lukas Spies, Kasia Swirydowicz, Jeremy Thompson, Ananias Tomboulides, and Vladimir Tomov. 2021. Efficient exascale discretizations: High-order finite element methods. *Int. J. High Perf. Comput. Appl.* 35, 6 (2021), 527–552.

[58] Martin Kronbichler and Momme Allalen. 2018. Efficient high-order discontinuous Galerkin finite elements with matrix-free implementations. In *Advances and New Trends in Environmental Informatics*, Hans-Joachim Bungartz, Dieter Kranzlmüller, Volker Weinberg, Jens Weismüller, and Volker Wohlgemuth (Eds.). Springer, Berlin, 89–110.

[59] Martin Kronbichler, Ababacar Diagne, and Hanna Holmgren. 2018. A fast massively parallel two-phase flow solver for microfluidic chip simulation. *Int. J. High Perf. Comput. Appl.* 32, 2 (2018), 266–287. https://doi.org/10.1177/1094342016671790

[60] Martin Kronbichler, Niklas Fehn, Peter Munch, Maximilian Bergbauer, Karl-Robert Wichmann, Carolin Geitner, Momme Allalen, Martin Schulz, and Wolfgang A Wall. 2021. A next-generation discontinuous Galerkin fluid dynamics solver with application to high-resolution lung airflow simulations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'21)*. Association for Computing Machinery (ACM), 1–15.

[61] Martin Kronbichler, Timo Heister, and Wolfgang Bangerth. 2012. High accuracy mantle convection simulation through modern numerical methods. *Geophys. J. Int.* 191 (2012), 12–29. https://doi.org/10.1111/j.1365-246X.2012.05609.x

[62] Martin Kronbichler and Katharina Kormann. 2012. A generic interface for parallel cell-based finite element operator application. *Comput. Fluids* 63 (2012), 135–147. https://doi.org/10.1016/j.compfluid.2012.04.012

[63] Martin Kronbichler and Katharina Kormann. 2019. Fast matrix-free evaluation of discontinuous Galerkin finite element operators. *ACM Trans. Math. Softw.* 45, 3 (2019), 28:1–40. https://doi.org/10.1145/3325864

[64] Martin Kronbichler and Karl Ljungkvist. 2019. Multigrid for matrix-free high-order finite element computations on graphics processors. *ACM Trans. Parallel Comput.* 6, 1 (2019), 2:1–32.

[65] Martin Kronbichler, Dmytro Sashko, and Peter Munch. 2022. Enhancing data locality of the conjugate gradient method for high-order matrix-free finite-element implementations. *Int. J. High Perf. Comput. Appl.* (unpublished). https://doi.org/10.1177/10943420221107880

[66] Martin Kronbichler and Wolfgang A. Wall. 2018. A performance comparison of continuous and discontinuous Galerkin methods with fast multigrid solvers. *SIAM J. Sci. Comput.* 40, 5 (2018), A3423–A3448.

[67] Chunlei C. Liang, Ravishekar Kannan, and Z. J. Wang. 2009. A p-multigrid spectral difference method with explicit and implicit smoothers on unstructured triangular grids. *Comput. Fluids* 38, 2 (2009), 254–265. https://doi.org/10.1016/j.compfluid.2008.02.004

[68] Karl Ljungkvist. 2014. Matrix-free finite-element operator application on graphics processing units. In *European Conference on Parallel Processing*, Luís Lopes, Julius Žilinskas, Alexandru Costan, Roberto G. Cascella, Gabor Kecskemeti, Emmanuel Jeannot, Mario Cannataro, Laura Ricci, Siegfried Benkner, Salvador Petit, Vittorio Scarano, José Gracia, Sascha Hunold, Stephen L. Scott, Stefan Lankes, Christian Lengauer, Jesús Carretero, Jens Breitbart, and Michael Alexander (Eds.). Springer, Berlin, 450–461.

[69] Karl Ljungkvist. 2017. Matrix-free finite-element computations on graphics processors with adaptively refined unstructured meshes. In *SpringSim (HPC)*, The Society for Modeling and Simulation International, San Diego, CA, 1–12.

[70] Salvatore Lopez and Raffaele Casciaro. 1997. Algorithmic aspects of adaptive multigrid finite element analysis. *Int. J. Numer. Methods Eng.* 40, 5 (1997), 919–936.

[71] Cao Lu, Xiangmin Jiao, and Nikolaos Missirlis. 2014. A hybrid geometric+algebraic multigrid method with semi-iterative smoothers. *Numer. Lin. Algebr. Appl.* 21, 2 (2014), 221–238. https://doi.org/10.1002/nla.1925

[72] Hong Luo, Joseph D. Baum, and Rainald Löhner. 2006. A p-multigrid discontinuous Galerkin method for the Euler equations on unstructured grids. *J. Comput. Phys.* 211, 2 (2006), 767–783. https://doi.org/10.1016/j.jcp.2005.06.019

[73] Hong Luo, Joseph D. Baum, and Rainald Löhner. 2008. Fast p-multigrid discontinuous Galerkin method for compressible flows at all speeds. *AIAA J.* 46, 3 (2008), 635–652. https://doi.org/10.2514/1.28314

[74] Yvon Maday and Rafael Munoz. 1988. Spectral element multigrid. II. theoretical justification. *J. Sci. Comput.* 3, 4 (1988), 323–353. https://doi.org/10.1007/BF01065177

[75] Brendan S. Mascarenhas, Brian T. Helenbrook, and Harold L. Atkins. 2009. Application of p-multigrid to discontinuous Galerkin formulations of the Euler equations. *AIAA J.* 47, 5 (2009), 1200–1208. https://doi.org/10.2514/1.39765

[76] Brendan S. Mascarenhas, Brian T. Helenbrook, and Harold L. Atkins. 2010. Coupling p-multigrid to geometric multigrid for discontinuous Galerkin formulations of the convection-diffusion equation. *J. Comput. Phys.* 229, 10 (2010), 3664–3674. https://doi.org/10.1016/j.jcp.2010.01.020

[77] Stephen F. McCormick. 1989. *Multilevel Adaptive Methods for Partial Differential Equations*. SIAM.

[78] J. Markus Melenk, Klaus Gerdes, and Christoph Schwab. 2001. Fully discrete hp-finite elements: Fast quadrature. *Comput. Methods Appl. Mech. Eng.* 190, 32 (2001), 4339–4364. https://doi.org/10.1016/S0045-7825(00)00322-4

[79] William F. Mitchell. 2010. The hp-multigrid method applied to hp-adaptive refinement of triangular grids. *Numer. Lin. Algebr. Appl.* 17, 2-3 (2010), 211–228.

[80] Peter Munch, Katharina Kormann, and Martin Kronbichler. 2021. hyper.deal: An efficient, matrix-free finite-element library for high-dimensional partial differential equations. *ACM Trans. Math. Softw.* 47, 4 (2021), 33:1–34. https://doi.org/10.1145/3469720

[81] Peter Munch, Karl Ljungkvist, and Martin Kronbichler. 2022. Efficient application of hanging-node constraints for matrix-free high-order FEM computations on CPU and GPU. In *High Performance Computing*. Ana-Lucia Varbanescu, Abhinav Bhatele, Piotr Luszczek, and Marc Baboulin (Eds.). Springer International Publishing, Cham, 133–152.

[82] Steffen Müthing, Marian Piatkowski, and Peter Bastian. 2017. High-performance implementation of matrix-free high-order discontinuous Galerkin methods. arXiv:1711.10885. Retrieved from https://arxiv.org/abs/1711.10885.

[83] Cristian R. Nastase and Dimitri J. Mavriplis. 2006. High-order discontinuous Galerkin methods using an hp-multigrid approach. *J. Comput. Phys.* 213, 1 (2006), 330–357. https://doi.org/10.1016/j.jcp.2005.08.022

[84] Benedict O'Malley, József Kópházi, Richard P. Smedley-Stevenson, and Monroe D. Eaton. 2017. Hybrid multi-level solvers for discontinuous Galerkin finite element discrete ordinate diffusion synthetic acceleration of radiation transport algorithms. *Ann. Nucl. Energy* 102 (April 2017), 134–147. https://doi.org/10.1016/j.anucene.2016.11.048

[85] Benedict O'Malley, József Kópházi, Richard P. Smedley-Stevenson, and Monroe D. Eaton. 2017. P-multigrid expansion of hybrid multilevel solvers for discontinuous Galerkin finite element discrete ordinate (DG-FEM-SN) diffusion synthetic acceleration (DSA) of radiation transport algorithms. *Progr. Nucl. Energ.* 98 (2017), 177–186. https://doi.org/10.1016/j.pnucene.2017.03.014

[86] Steven A. Orszag. 1980. Spectral methods for problems in complex geometries. *J. Comput. Phys.* 37, 1 (1980), 70–92. https://doi.org/10.1016/0021-9991(80)90005-4

[87] Sachin Premasuthan, Chunlei Liang, Antony Jameson, and Zhi Wang. 2009. A p-multigrid spectral difference method for viscous compressible flow using 2D quadrilateral meshes. In *Proceedings of the 47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*. American Institute of Aeronautics and Astronautics. https://doi.org/10.2514/6.2009-950

[88] Einar M. Rønquist and Anthony T. Patera. 1987. Spectral element multigrid. I. formulation and numerical results. *J. Sci. Comput.* 2, 4 (1987), 389–406. https://doi.org/10.1007/BF01061297

[89] Johann Rudi, Omar Ghattas, A. Cristiano I. Malossi, Tobin Isaac, Georg Stadler, Michael Gurnis, Peter W. J. Staar, Yves Ineichen, Costas Bekas, and Alessandro Curioni. 2015. An extreme-scale implicit solver for complex PDEs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on (SC'15)*. ACM Press, New York, NY, 1–12. https://doi.org/10.1145/2807591.2807675

[90] Anton Schüller. 2013. *Portable Parallelization of Industrial Aerodynamic Applications (POPINDA): Results of a BMBF Project*. Vol. 71. Vieweg+Teubner Verlag, Wiesbaden.

[91] Khosro Shahbazi, Dimitri J. Mavriplis, and Nicholas K. Burgess. 2009. Multigrid algorithms for high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations. *J. Comput. Phys.* 228, 21 (2009), 7917–7940. https://doi.org/10.1016/j.jcp.2009.07.013

[92] Mark S. Shephard. 1984. Linear multipoint constraints applied via transformation as part of a direct stiffness assembly process. *Int. J. Numer. Methods Eng.* 20, 11 (1984), 2107–2112.

[93] Jörg Stiller. 2016. Nonuniformly weighted Schwarz smoothers for spectral element multigrid. *J. Sci. Comput.* 72 (2016), 81–96. https://doi.org/10.1007/s10915-016-0345-z

[94] Jörg Stiller. 2017a. Nonuniformly weighted Schwarz smoothers for spectral element multigrid. *Journal of Scientific Computing* 72, 1 (2017), 81–96. DOI: http://dx.doi.org/10.1007/s10915-016-0345-z

[95] Jörg Stiller. 2017b. Robust multigrid for cartesian interior penalty DG formulations of the poisson equation in 3D. In *Spectral and High Order Methods for Partial Differential Equations ICOSAHOM 2016. Lecture Notes in Computational Science and Engineering*, Marco L. Bittencourt, Ney A. Dumont, and Jan S. Hesthaven (Eds.). Vol. 119. Springer, Cham, 189–201. DOI: http://dx.doi.org/10.1007/978-3-319-65870-4_12

[96] Mario Storti, Norberto M. Nigro, and Sergio Idelsohn. 1991. Multigrid methods and adaptive refinement techniques in elliptic problems by finite element methods. *Computer Methods in Applied Mechanics and Engineering* 93, 1 (1991), 13–30.

[97] Klaus Stüben. 2001. A review of algebraic multigrid. *J. Comput. Appl. Math.* 128, 1-2 (2001), 281–309. https://doi.org/10.1016/S0377-0427(00)00516-1

[98] Hari Sundar, George Biros, Carsten Burstedde, Johann Rudi, Omar Ghattas, and Georg Stadler. 2012. Parallel geometric-algebraic multigrid on unstructured forests of octrees. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. IEEE Computer Society Press, 1–11.

[99] Hari Sundar, Georg Stadler, and George Biros. 2015. Comparison of multigrid algorithms for high-order continuous finite element discretizations. *Numer. Lin. Algebr. Appl.* 22, 4 (2015), 664–680. https://doi.org/10.1002/nla.1979

[100] Haijun Wu and Zhiming Chen. 2006. Uniform convergence of multigrid V-cycle on adaptively refined finite element meshes for second order elliptic problems. *Sci. Chin. Ser. A: Math.* 49, 10 (2006), 1405–1429.

[101] Pamela Zave and Werner C. Rheinboldt. 1979. Design of an adaptive, parallel finite-element system. *ACM Trans. Math. Softw.* 5, 1 (1979), 1–17.

[102] Weiqun Zhang, Ann Almgren, Vince Beckner, John Bell, Johannes Blaschke, Cy Chan, Marcus Day, Brian Friesen, Kevin Gott, Daniel Graves, Max Katz, Andrew Myers, Tan Nguyen, Andrew Nonaka, Michele Rosso, Samuel Williams, and Michael Zingale. 2019. AMReX: A framework for block-structured adaptive mesh refinement. *J. Open Source Softw.* 4, 37 (May 2019), 1–4. https://doi.org/10.21105/joss.01370

Received 5 April 2022; revised 21 December 2022; accepted 11 January 2023