

# Parallel Geometric-Algebraic Multigrid on Unstructured Forests of Octrees

Hari Sundar\*, George Biros\*<sup>†</sup>, Carsten Burstedde\*<sup>§</sup>,  
Johann Rudi\*, Omar Ghattas\*<sup>†‡</sup> and Georg Stadler\*

\*Institute for Computational Engineering and Sciences, The University of Texas at Austin, Austin, TX

<sup>†</sup>Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX

<sup>‡</sup>Jackson School of Geosciences, The University of Texas at Austin, Austin, TX

<sup>§</sup>Now at Institute for Numerical Simulation, Rheinische Friedrich-Wilhelms-Universität Bonn, Bonn, Germany

**Abstract**—We present a parallel multigrid method for solving variable-coefficient elliptic partial differential equations on arbitrary geometries using highly adapted meshes. Our method is designed for meshes that are built from an unstructured hexahedral macro mesh, in which each macro element is adaptively refined as an octree. This forest-of-octrees approach enables us to generate meshes for complex geometries with arbitrary levels of local refinement. We use geometric multigrid (GMG) for each of the octrees and algebraic multigrid (AMG) as the coarse grid solver. We designed our GMG sweeps to entirely avoid collectives, thus minimizing communication cost.

We present weak and strong scaling results for the 3D variable-coefficient Poisson problem that demonstrate high parallel scalability. As a highlight, the largest problem we solve is on a non-uniform mesh with 100 billion unknowns on 262,144 cores of NCCS’s Cray XK6 “Jaguar”; in this solve we sustain 272 TFlops/s.

## I. INTRODUCTION

We focus on multigrid solvers for elliptic partial differential operators. Roughly speaking, there are two main challenges in numerically “inverting” elliptic operators, meshing and ill-conditioning of the linear system that we obtain upon discretization. Various schemes that address these two issues exist but they are fragile in that their effectiveness is very sensitive to the domain geometry, boundary conditions, and highly variable coefficients. The difficulties in designing solvers for elliptic operators are amplified when we target extreme scale parallelization since issues related to efficient data structures, latency and communication, and overall efficient utilization of resources come into play.

Multigrid is one of the most effective solvers for elliptic operators. It is algorithmically optimal, robust when combined with Krylov methods, and in the case of uniform grids, quite easy to implement and parallelize. (However, getting good per-core performance is harder.) Parallel scalability becomes much more involved in the case of non-uniform grids, but there are many success stories when the overall geometry is regular (e.g., box, cylinders, periodic domains). For highly unstructured grids, the performance of existing technologies severely degrades for very large core counts.

The key challenges in the parallel scalability of multigrid are the efficient construction of coarsening and prolongation operators and the need for repartitioning and load balancing

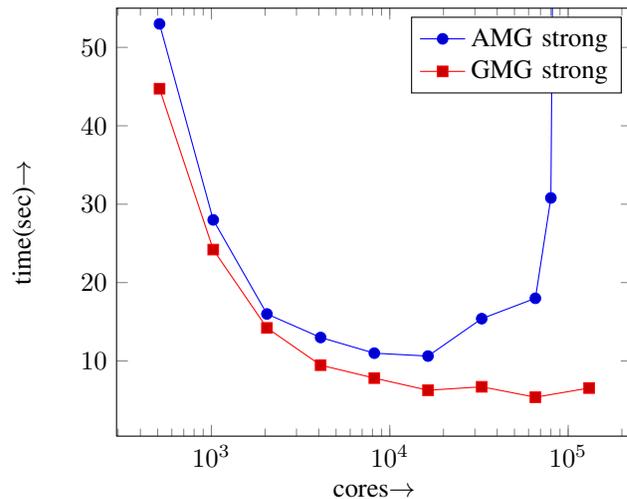


Fig. 1. Strong scalability comparison between algebraic and geometric multigrid for a variable-coefficient Poisson solve on a spherical shell mesh adaptively refined based on the gradient of the coefficient with overall 124M elements. Reported is the total time to solution, i.e., the sum of setup and solve times. The 256-fold increase in the number of cores reduces the number of elements per core from 240K on 512 cores to just 946 on 131,072 cores. Both methods show a similar scaling behavior up to 16K cores. For problems larger than that, our matrix-free geometric multigrid implementation scales significantly better than AMG. We used *ML* [1], which implements smoothed aggregation algebraic multigrid. This comparison was performed on ORNL’s Jaguar XK6 supercomputer.

the mesh all the way to the coarse grid while minimizing communication. Furthermore, exa-scalability requires avoidance of synchronizations—something that seems antagonistic to the complex mesh management operations described above.

**Contributions:** We address these challenges and present a parallel multigrid algorithm that uses the adaptive spatial data-structure framework [2] developed in our group. There, we used the notion of a “macro mesh”, a general unstructured hexahedral mesh that is refined using one octree data-structure per hexahedron (thus, the term “forest of octrees”). We translate this two-tier representation of the mesh to a two-tier algebraic and geometric multigrid solver. In a nutshell, the major contributions of this work are:

- We present a parallel geometric multigrid method on

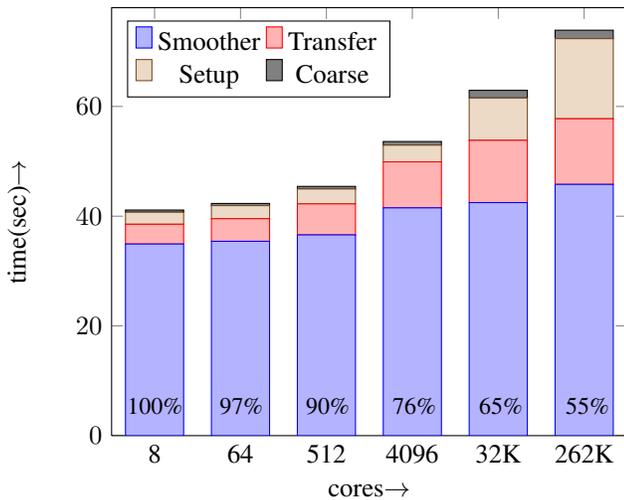


Fig. 2. Weak scalability of a variable-coefficient Poisson solve on a spherical shell mesh adaptively refined based on the gradient of the coefficient. Reported are timings for 8 iterations of a geometric multigrid-preconditioned conjugate gradient method, as well as the overall parallel efficiency in percentage. The overall time is split into the setup time, the smoothing time, the time for the coarse grid solve and for the parallel transfer between the meshes of the hierarchy. A grain size of 215K elements per process is used, which amounts to overall 56 billion elements for the largest problem. Note that we report weak scaling results from 8 to 262,144 cores, which is a 32,768-fold increase in problem size.

complex geometry-conforming adaptive meshes. We use matrix-free element-by-element traversals for the application of the system matrix on all multigrid levels except the coarse one, where the matrix is assembled and we employ algebraic multigrid for the coarse grid solve. The combination of GMG and AMG results in excellent scalability.

- We perform optimizations that enable us to achieve good single-core performance. We test the solver using a second-order discretization and thus, the calculations are memory-bound. We employ vectorization using Streaming SIMD Extensions (SSE) along with on-the-fly computations of geometric factors (the Jacobian and its determinant) resulting in improved performance. Our geometric multigrid does not require global identifiers for the communication, it is synchronization-free and uses only point-to-point, non-blocking communication, resulting in excellent strong and weak scaling.
- Finally, we conduct several experiments in which we test our algorithm and implementation on different meshes for constant and variable coefficient Poisson problems in 3D, and compare with state-of-the-art algebraic multigrid solvers. We report strong (Figure 1) and weak (Figure 2) scaling results.

*Related work:* There is an extensive amount of work on multigrid and its parallelization [3], [4], [5], [6], [7], [8]. Here, we restrict ourselves to recent work on parallel multigrid methods for high-end distributed memory architectures. The closest work to this paper is the work of two of the authors on octree multigrid solvers [9], [10] in which excellent scalability

has been demonstrated to thousands of cores. However, that approach suffers from two significant limitations, namely, it involves significant use of collective MPI calls at each multigrid cycle and does not handle complex geometries.

Multigrid on hierarchical hybrid grids (HHG) [11], [12] also employs a geometry-conforming unstructured macro mesh. In contrast to our approach, HHGs use uniform mesh refinement in each cell of the macro mesh. This makes it possible to implement extremely memory efficient coarsening and refinement for problems with constant coefficients. By construction, this approach only allows for limited, macro mesh-based adaptivity, while we allow arbitrary localized refinement and coarsening below the macro structure. HHGs were recently used to obtain impressive weak scaling results on up to 300K cores on Blue Gene/P in Jülich, with a maximal overall problem size of  $10^{12}$  unknowns [13].

Based on a new version of the unstructured grids (UG) package [14], recent work [15] presents formidable weak and strong scalability results for a Laplace equation on Blue Gene/P in Jülich on up to 262K cores. These results are for uniformly refined unit cubes, but, in principle, the framework also supports non-uniform meshes and different element types. Starting from a coarse mesh, it creates hierarchically distributed grids for the multigrid sweeps through refinement. To control the amount of communication, coarse grids are only distributed across subsets of processors.

Multigrid within the Peano framework (which uses space filling curves) is presented in [16], [17]. Adaptive grids on cube-shaped domains are constructed by a recursive splitting of each cell into three subcells in each coordinate direction. In [18], weak scalability results for a solver involving geometric multigrid on up to 900 cores are presented.

Geometric multigrid on a combination of globally unstructured and locally structured meshes is implemented in the Finite Element Analysis and Solution Tools (FEAST) [19] and has been scaled to 128 CPU cores. Here, the main target is on implementations and methods that achieve good performance on different architectures, including GPU-accelerated clusters.

An alternative to geometric multigrid are algebraic multigrid (AMG) approaches, for which the setup of the hierarchy uses graph-based algorithms rather than geometric mesh properties. AMG requires a setup phase in which a grid hierarchy and corresponding interpolation and restriction operators are computed solely based on the fine grid system matrix, which must be provided in assembled form. Parallel implementations of AMG require communication during the setup phase to properly aggregate degrees of freedom across processor boundaries. During this phase there is a trade-off between the depth and effectiveness of the multigrid hierarchy, and the memory and time used for the setup and for performing multigrid cycles. Challenges for parallel AMG are surveyed in [20].

One of the most powerful parallel implementation of algebraic multigrid is *BoomerAMG* from the *Hypre* linear solver library [21]. Recently, a significant effort has been made to prepare *BoomerAMG* for next-generation supercomputers

with millions of cores [22], [23]. This requires, for instance, improved coarsening strategies to reduce the complexity in the AMG grid hierarchy and thus maintain sparsity of the system matrix and the interpolation and restriction operators [24]. The latest implementation of BoomerAMG also supports 64 bit integers, and thus accommodates problems with more than  $2 \times 10^9$  unknowns. *ML* [1] from the *Trilinos* Project [25] implements smoothed aggregation, a variant of AMG, and has shown excellent robustness and scalability. Our geometric multigrid method uses *ML* for the coarse grid solve. Other parallel implementations of algebraic multigrid target fine-scale parallelism as required on clusters of GPUs, where the problem must be split into many independent threads to obtain good performance [26], [27], [28].

Finally, we briefly contrast AMG and GMG approaches. The advantages of AMG are that it can be used as a black-box algorithm and does not require geometry or mesh information (except for the fine mesh). Its disadvantages are the communication-intense setup and the increased memory requirement compared to matrix-free geometric multigrid. Advantages of GMG are that it can be used in a matrix-free fashion and that it has low memory overhead. Moreover, operators can easily be modified at different levels, which can be necessary to accommodate certain boundary conditions. The disadvantages of GMG are that it requires a hierarchy of meshes and that it cannot be used in a black-box fashion.

*Limitations:* Our methodology enables the solution of problems that were earlier intractable. Nevertheless, it has several limitations. First, it cannot handle a large number of jumps of the coefficients efficiently since the restriction and prolongation operators do not use coefficient information. Large anisotropies or re-entrant corners cause degradation of the smoother efficiency. Although there is some work on the load balancing quality using space-filling curves [18], these results do not apply to the Morton ordering we use. Thus, we were not able to derive guaranteed theoretical complexity estimates. In practice however, we never observed any issues with the quality of the parallel partition or load balance. Finally, if the mesh does not conform to the idea of the macro mesh, an octree forest cannot be used. Yet, our approach can handle surprisingly complex geometries, as our examples demonstrate.

*Organization of the paper:* In the following §II, we summarize the multigrid method and the overall structure of our forest-of-octrees-based meshing, coarsening and parallelization approach. We discuss the main algorithmic innovations and provide a complexity analysis of our scheme. In §III, we present scalability and robustness tests for constant and variable coefficient Poisson problems on a variety of meshes. We conclude in §IV.

## II. GEOMETRIC MULTIGRID

The multigrid method for solving the discretization

$$A_h u_h = f_h \quad (1)$$

of an elliptic partial differential equation amounts to the recursive application of the following two-grid method to iteratively reduce the residual in (1):

$$\begin{aligned} \textit{Pre-smooth:} & & u_k & \leftarrow & S_k(u_k, f_k, A_k) \\ \textit{Compute Residual:} & & r_k & \leftarrow & f_k - A_k u_k \\ \textit{Restrict:} & & r_{k-1} & \leftarrow & R_k r_k \\ \textit{Recurse:} & & e_{k-1} & \leftarrow & A_{k-1}^{-1} r_{k-1} \\ \textit{Correct:} & & u_k & \leftarrow & u_k + P_k e_{k-1} \\ \textit{Post-smooth:} & & u_k & \leftarrow & S_k(u_k, f_k, A_k) \end{aligned}$$

Here,  $S_k$  is the smoother and  $k \geq 0$  denotes the multigrid level. The solve at the coarsest level  $k = 0$  is done using a direct solver; in our case using algebraic multigrid. This scheme is often referred to as a V-cycle.

Our target is a highly scalable, distributed-memory parallel, matrix-free implementation of the geometric multigrid method on adaptively refined grids and complex geometries. We focus on non-conforming octree-based hexahedral discretizations which allow a high level of adaptivity and parallel efficiency. In the following sections, we first introduce our adaptive octree-based meshing framework and the generation of the multigrid hierarchy, followed by details of the various steps of the V-cycle.

### A. Octree-refined Unstructured Meshes

Octrees are axis-aligned binary space partitions in 3D. Given the axis-aligned nature of the partitions, the root of an octree can be associated with a topologically equivalent 3D domain that can be mapped smoothly to a cube. Recursive subdivision of the tree to generate octants corresponds to a subdivision of the mapped domain and thus generates a mesh that can be used for the discretization and numerical solution of a partial differential equation. A forest is a collection of octrees, which are connected conformingly through faces, edges and corners [2]. Such a forest of octrees allows for a two-tier decomposition of geometric domains, where the first tier, called the macro mesh, is an unstructured mesh of conforming subvolumes, each mapped from a reference cube by a smooth transformation. Recursive octree subdivision then allows for efficient octree-based adaptivity of the mesh in the geometric domain. Localized subdivision generally creates non-conforming meshes, which can still be used to represent continuous finite element solutions by incorporating algebraic constraints at the element level. The first-tier macro mesh can be specified by hand for simple domain shapes, or generated by hexahedral mesh generators to conform to more general shapes. The second-tier adaptivity allows for dynamic refinement and coarsening and can be driven through error indicators or problem-specific parameters such as material properties. The adaptive octree structure can naturally be interpreted to define a recursive space filling curve that traverses all octants in all trees in the so-called Morton- or  $z$ -order. This curve provides the basis for sorted numbering and fast repartitioning of octants between processes, a fact that we exploit in the creation of the multigrid hierarchy.

Completely unstructured meshes naturally provide the largest geometrical flexibility. However, their adaptation and the construction of mesh hierarchies and intergrid operators needed in geometric multigrid methods can be technical and computationally expensive, in particular in parallel. Compared to that, meshes based on *single* octrees provide high levels of adaptivity, allow efficient parallel mesh adaptation [29], [30], and are well suited for matrix-free parallel multigrid [9], [10]. However, their applicability is limited either to domains that can be mapped to a cube or to problems with simple boundary conditions, which can be approximated reasonably well for an embedding of the computational domain within a cube. Unfortunately, many important problems require geometric domains that are topologically distinct from a cube and have complicated boundary conditions. The Earth’s mantle and the Antarctic ice sheet—both used in this paper—are just two of many examples.

A two-tier approach combing an unstructured macro mesh with octree refinement for each macro element combines the flexibility of unstructured meshes with the efficiency and high levels of adaptation of a single octree approach, and thus provides a suitable starting point for highly scalable geometric multigrid. In the next section, we describe the setup of the multigrid mesh hierarchy in this context.

### B. Setting up the Multigrid Hierarchy

Multigrid requires the construction of a hierarchy of meshes such that every octant at level  $k$  is either present at the finer level  $k+1$  or is replaced by its eight children. This construction constitutes the setup phase of GMG and needs to be done only once as long as the fine mesh does not evolve in time. The main steps in building a grid hierarchy, such as displayed in Figure 3, are the following:

1) *Coarsen*: Coarsen the fine mesh by one level by traversing the forest in order. Since we use a Morton-ordered linear representation of the octree, a family of eight sibling octants, where present, will be contiguous in memory. The coarsen operation replaces these with their parent to produce the coarse octree. To make coarsening a processor-local operation, we partition in such a way that every family of eight sibling octants that is a candidate for coarsening is placed on the same process; see §II-F. In the context of a forest of octrees it is important to note that we cannot coarsen beyond the first-tier macro mesh. More details on the coarsening algorithm can be found in [2], [9].

2) *2:1-Balance*: We require that the sizes of adjacent octants can at most differ by a factor of two. This is known as the 2:1-balance constraint. Details on enforcing this constraint in parallel can be found in [30] and [31]. We call this routine after coarsening, which effectively refines locally wherever the coarsening operation is found to be invalid. While it is thinkable to combine Coarsen and Balance, we prefer to keep them separate since both are useful modular operations on their own.

3) *Partition*: Coarsening and the subsequent 2:1-balancing of the octree can result in a non-uniform distribution of coarse

octants across the processes, leading to load imbalance. The Morton ordering enables us to equipartition the octants by performing a parallel scan on the number of octants on each process followed by point-to-point communication to redistribute the octants. Additional issues related to load balancing are discussed in §II-E.

4) *Extract Mesh*: By meshing we refer to the construction of the (numerical) data required for finite element computations from the (topological) octree data. In addition to the mesh extracted on the fine grid that is relevant for the simulation as a whole, we extract two meshes per multigrid level; see Figure 4. First, we extract a surrogate mesh after coarsening and 2:1-balance of the forest. It is used to provide information required for the restriction of the residual, namely the depth of the elements. Second, we extract the true mesh for this level after repartitioning the forest. This way we separate the processor-local numerical restriction from the parallel partition that does not perform any numerical computation. The true meshes always contain all information for applying the elliptic operator  $A_k$  in addition to the encoding of the partition. We have successfully used this two-stage cycle for time-dependent dynamic AMR simulations in the past [32]. The use of the surrogate meshes for intergrid transfers is further detailed in §II-C.

5) *Recurse*: To build the next multigrid level in the hierarchy, repeat steps 1–4 using the coarsened true mesh as the new fine mesh. The recursion is stopped when either the required number of multigrid levels has been created or when no further coarsening of the mesh is possible, that is, each octree has been reduced to its root element.

### C. Restriction & Prolongation

We implement the restriction and prolongation operators in a matrix-free manner, using matrix-vector product computations (MatVecs) that do not require assembly of the matrix entries. Similar to matrix-free applications of the system matrix, all operations required for the intergrid operations are done at the element level. As in [10], the restriction operator is the transpose of the prolongation operator. Since we use Morton-ordered linear octrees, these operators can be applied via a single simultaneous traversal over both the fine and the coarsened forest. Furthermore, since both octrees are complete in the sense that they contain a contiguous subset of the space-filling curve, we only need information on the depth of the octants within the octree to perform the restriction and prolongation. It is important to note that one of the main advantages of implementing geometric multigrid on complete linear octrees is that *no intergrid element searches or look-up tables are needed*. As we traverse both forests simultaneously, only two possibilities exist<sup>1</sup>:

- Both the fine and coarse octants are the same,
- The fine octants (eight of them) are children of the coarse octant.

<sup>1</sup>This is only true when the forests are at most 1 level apart, as is the case in our multigrid hierarchy.

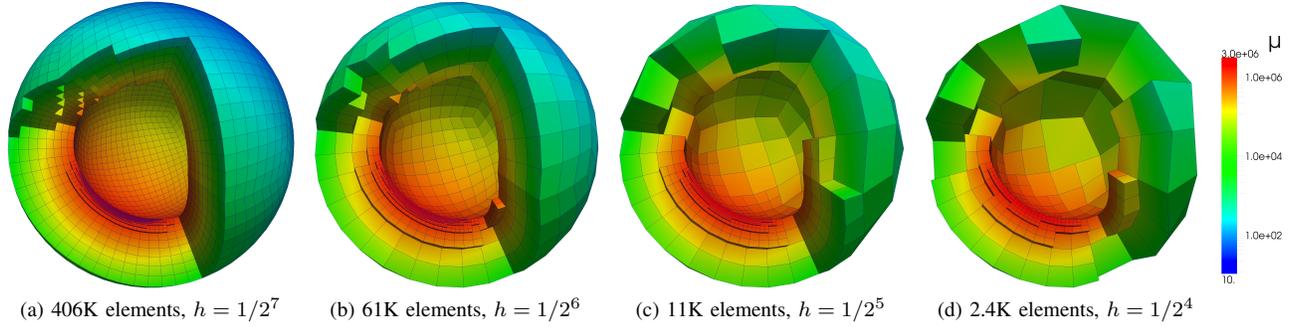


Fig. 3. Example for multigrid hierarchy for the spherical shell. Four grids are shown starting from (a) the finest to (d) the coarsest. As can be seen, the factor by which the total number of elements reduces between grids in the hierarchy can vary, which is due to adaptive nature of our meshes. The color represents the coefficient field  $\mu$ , which was used to create the fine mesh (a). Please zoom in on the electronic version to see additional detail.

In the first case, we simply copy the finite-element values associated with this octant, while in the second case we identify all independent fine grid nodes that lie within the support of each coarse grid shape function. Our multigrid implementation builds on a framework for the discretization of partial differential equation using continuous and discontinuous Galerkin finite element methods [33]. Using the transformations from the continuous to the discontinuous shape functions, we can implement the restriction and prolongation MatVecs on the discontinuous element without having to keep track of the local arrangement of hanging nodes using complex masks as in [10]. The elemental prolongation is done by evaluating the local discontinuous shape functions at the locations of the child nodes. When mapping back from the discontinuous to the continuous representation, multiple elements will contribute to the same continuous node. The number of contributions will depend on the local arrangement of hanging nodes, and so we keep a count of the number of replications for each continuous node and average out the values during the transformation. The computational cost for this step is  $\mathcal{O}(N/p)$  requiring an update of ghost values. We interleave computation and communication to hide the communication costs associated with this step.

In order to be able to coarsen and prolongate in parallel, at each grid level, except the finest, we maintain a surrogate mesh in addition to the true mesh. The surrogate mesh is needed to represent the intermediate stage of the forest after coarsening/2:1-balance and before repartitioning. We denote the redistribution of finite-element values between the pre-partition and post-partition coarse meshes as *Transfer*. The surrogate mesh only stores information needed for the restriction, prolongation and transfer operations (as opposed to an application of the elliptic operator  $A_k$ ) and consequently the memory overhead is low. In summary, we proceed by first restricting the residual from the fine mesh onto the coarse surrogate mesh followed by a transfer to the coarse true mesh. For prolongation, we first transfer the defect correction from the coarse true mesh onto the coarse surrogate mesh followed by a prolongation from the surrogate onto the fine mesh. The multigrid recursion turns this sequence into a V-cycle as

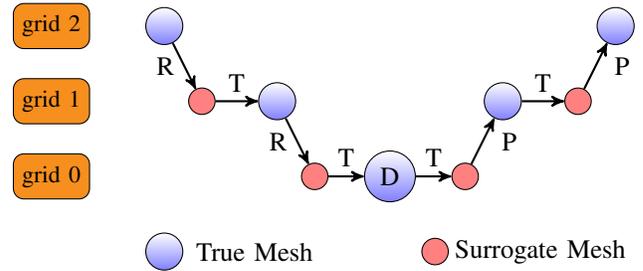


Fig. 4. The concept of surrogate meshes to aid in restriction and prolongation and to ensure proper load balance at all multigrid levels. The downward arrows refer to the creation of a surrogate mesh by coarsening and 2:1-balance. It shares the same parallel partition boundaries with the true mesh it is derived from. Correspondingly, the associated restriction  $R$  of the residual is an operation on the processor-local degrees of freedom. The horizontal arrows in the left half refer to the creation of the true mesh from the surrogate mesh by repartitioning, keeping the refinement structure of the mesh unchanged. Correspondingly, the transfer operation  $T$  communicates degrees of freedom between the old and new owner process of each cell in a point-to-point fashion.  $D$  denotes the direct solve on the coarsest grid and  $P$  the prolongation of the defect correction, which is again a processor-local operation. Note that all meshes at a particular grid level have the same number of global octants. The surrogate and true meshes on coarser levels are created only once during the setup phase, but visited twice during each V-cycle.

illustrated in Fig. 4.

#### D. Smoothing & Coarse Grid Operator

We use damped Jacobi smoothing with  $\omega = \frac{2}{3}$  for all runs reported in this paper. As the coarse grid solver, we use the Smoothed Aggregation Algebraic Multigrid (AMG) implementation ML, which is part of the Trilinos project [1]. This combined GMG-AMG approach is particularly suitable for our two-tier geometric decomposition of the domain, since we are unable to coarsen the mesh beyond the unstructured macro mesh. In the mesh coarsening process, we can switch from the octree-based geometric coarsening to algebraic coarsening at any multigrid level. A direct solver is used on the coarsest grid of the AMG hierarchy.

The discrete operators on all grids are based on a standard FEM discretization of the PDE operator with trilinear finite element basis functions. It can be shown that this is equivalent to the projection of the coarse grid operators using

the restriction, provided the same bilinear form is used on all grids [10]. Dirichlet boundary conditions are enforced through a modification of the MatVec and the right hand side. For Dirichlet nodes, the off-diagonal elements in the corresponding column and row are removed from the system matrix, and the right hand side is modified to enforce the Dirichlet value. This operation is performed in the MatVec elemental loop.

One could argue that AMG could directly be used as a solver instead of the proposed two-tier approach. However, obtaining good parallel scalability for the AMG setup phase at very high process counts is challenging [20], [22], [23], as we also demonstrate in our experiments. Additionally, the octree-based coarsening and repartitioning algorithms have been shown to be fast and require little memory; see also the timing for *Setup* in Figure 2. The combination of GMG with AMG allows both algorithms to play to their strengths, keeping the overall solve and setup time low and yielding excellent strong and weak scaling.

### E. Partitioning & Load Balancing

The reduction in the problem size at successive grids creates some problems from the perspective of load balancing and care has to be taken to ensure that the communication costs do not dominate over the computation.

As explained in §II-B we partition each grid separately to ensure that all processes have an equal number of elements. However, keeping all processes active at all grids is not efficient and therefore, while partitioning the surrogate mesh, we dynamically reduce the number of processes that are active at the coarser grid in order to ensure that communication does not dominate the computation. This can occur when the number of elements at interprocess boundaries, and therefore the associated communication cost, becomes larger than the number of internal elements. For this purpose we define a minimal grain size, i.e., the minimum number of elements per process below which the communication costs start to dominate. The grain size can be computed either analytically for predictably partitioned meshes or empirically by performing experiments on the target architecture<sup>2</sup>. We use the same `MPI_Comm` at all levels and only the number of processes with zero elements is increased progressively. Note that our implementation only uses non-blocking point-to-point communication in the V-cycle, and in particular does not use any barriers—consequently idle processes do not cause significant overhead. The only barrier in our multigrid implementation is due to the reduction needed for the computation of the norm at the end of the V-cycle (and this is an optional computation), or when multigrid is used as preconditioner for the CG method.

We observed suboptimal scalability for the ML setup using the fullsize `MPI_Comm` with zero-element processes (even though they work just fine in our GMG setup and transfer). Therefore, in the setup of the AMG hierarchy we create a new `MPI_Comm` comprising of only the processes with non-zero elements to be used for the coarse grid solve.

<sup>2</sup>We used a minimal grain size of 1000 elements for all experiments.

### F. Partition Correction for Coarsening

In defining the V-cycle we require that the coarsening operation remains communication-free. Furthermore, we require that the topology of the coarsened mesh remains invariant under variable process counts and partitions. Thus the distribution of octants has to be such that a complete set of eight sibling octants is never divided between multiple processes. We accomplish this parallel invariance by introducing a correction to the octant-to-processor assignment in every partitioning step.

The partitioning algorithm for octree meshes begins by computing an optimally load-balanced distribution of octants, which is possible in  $\mathcal{O}(1)$  time by exploiting the properties of the space-filling curve. At this point, the goal is to determine whether this hypothetical new partition would separate complete sets of octant siblings and, wherever that is the case, to correct it by assigning all siblings to the same process.

The correction is made by having each process determine independently what changes to the new partition have to be made for its own subset of octants. To this end, processes exchange octants that are in critical neighborhoods about their respective first assigned local octants of the new partition via point-to-point communication. Since any given critical octant neighborhood has 14 or less elements, the associated communication usually involves only a few processes with consecutive ranks. Finally, the locally computed corrections—one integer per process—are shared with all other processes by a call to `MPI_Allgather`, which is sufficient to compute the global correction simultaneously on all processors.

We have designed this algorithm in a way that in the case of a split octant family, the process with the most siblings will receive all other siblings. This interferes with the load balance as little as possible. Furthermore, the correction will never assign octants to empty processors. The latter property is crucial to maintain proper clustering of octants when the mesh becomes very coarse.

### G. Single Core Optimizations

We use Streaming SIMD Extensions (SSE) technology, available in most modern CPUs, to speed up the floating point computations. We use SSE to accelerate the elemental MatVec as well as the restriction and prolongation operators. Finally, we perform the elemental MatVecs without precomputing the geometric factors (the Jacobian and its determinant) associated with the element to improve float-to-memory access ratios and improved overall performance.

### H. Complexity

Let the total number of elements in the fine mesh be  $N$  and  $p$  the total number of processes. The time complexity of forest construction and coarsening is  $\mathcal{O}(N/p)$ , and the time complexity of enforcing 2:1-balance is  $\mathcal{O}(N/p \log N/p)$  [2]. The cost of a MatVec is  $\mathcal{O}(N/p)$ . Assuming a regular grid, we can estimate the communication costs as well: Each coarsen, 2:1-balance, and partition-correction call requires additional  $\mathcal{O}(\log p)$  time to `MPI_Allgather` the local octant count

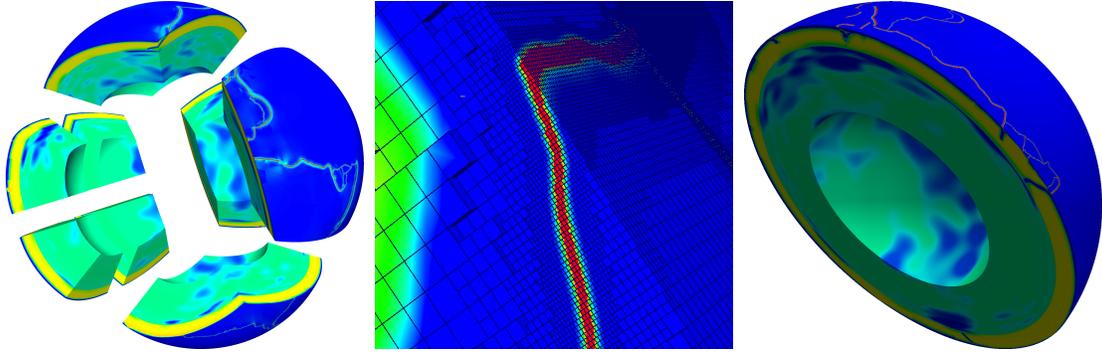


Fig. 5. Spherical shell consisting of 24 octrees (left), slice through the mesh (right) and zoom showing adaptively refined mesh (middle). The color depicts the logarithm of the coefficient field coming from the simulation of global mantle flow, where the narrow red zones with lowest viscosity correspond to tectonic plate boundaries. The 6 order of magnitude variation in the coefficient is resolved on the mesh with 23B finite elements corresponding to octree levels between 9 and 19 on 64K processes.

(one long integer). For partition and transfer all elements of a process can potentially be communicated, therefore the communication complexity per process is  $\mathcal{O}(N/p)$ .

### III. RESULTS

Here, we report results for a highly-varying coefficient Poisson problem for examples with forests constructed from an increasing number of octrees, namely a cube (one octree), a spherical shell (24 octrees) and a mesh for the Antarctic ice sheet (45,449 octrees). We study the overall strong and weak scalability of our geometric multigrid implementation and the scalability of its individual components. We compare the parallel performance of combined geometric and algebraic multigrid with the performance of algebraic multigrid, using the ML implementation. We also illustrate the limitations of geometric multigrid methods for problems with strong mesh anisotropy or discontinuous coefficients.

We solve a three-dimensional Poisson problem with Dirichlet boundary conditions on a domain  $\Omega$  and an isotropic, spatially varying coefficient  $\mu$ ,

$$-\operatorname{div}(\mu(x)\nabla u(x)) = f(x) \quad \forall x \in \Omega, \quad u(x) = 0 \quad \text{on } \partial\Omega. \quad (2)$$

All experiments were carried out on Jaguar, the ORNL Cray XK6 system that has a total number of 299,008 cores. The largest runs reported in this work used a total of 262,144 cores.

#### A. Meshes

We used two kinds of meshes for the experiments. The first corresponds to a spherical shell domain. The shell has a radius of 1 and a thickness of 0.45, and is constructed of 24 warped cubes. It is topologically different from a cube, which is why a single octree cannot be used as underlying data structure for a shell geometry—several properly connected cubes, each represented by an octree, must be used. For the scalability experiments on this shell, we consider two coefficient fields. First, we define  $\mu$  as the sum of unity and two Gaussians, which are scaled by a factor of  $10^6$ . One Gaussian is centered

at  $(0.5, 0.5, 0.5)$  and has variance  $\sigma^2 = 0.2$ , the other one is centered at  $(-0.5, -0.5, -0.5)$  and uses  $\sigma^2 = 0.4$ . The underlying mesh was refined based on the gradient of  $\mu$ . A hierarchy of meshes produced by repeated coarsenings of this mesh are shown in Fig. 3.

We also performed simulation on the shell using coefficients  $\mu$  that arise in the simulation of global mantle flow with plate tectonics, [34], [35]. Here, Stokes flow problems with highly varying viscosities on adaptively refined meshes that resolve these variations have to be solved. Plate boundaries are modeled as very narrow (a few kilometers wide) zones with up to 6 orders of magnitude smaller viscosity than the surrounding plates as shown in Fig. 5. For the iterative solution of such large-scale, varying-coefficient Stokes problems, efficient preconditioners for elliptic, positive-definite subproblems such as (2) or its vector versions are the critical ingredient. To study the scalability of our multigrid solver for application problems with highly adapted mesh and large gradients in the coefficient we solve (2) on adaptive refinements of the mesh shown in Fig. 5.

We also ran scalability experiments on a mesh of Antarctica as a demonstration of the scalability of our method for large forests. The Antarctica macro mesh is based on a data set from [36] and consists of 45K octrees, each of which is refined to maintain a distribution of 400K elements per process in our weak scaling experiments. Unlike the mantle mesh, there is large variability in the size of elements in the Antarctica macro mesh as can be seen in Fig. 6. This mesh illustrates the ability of our two-tier mesh in being able to capture complex geometries. The resulting elements are strongly anisotropic since the Antarctic ice sheet has a thickness of only up to 4 km, while spanning several 1000 km in the lateral directions. For our scalability tests, we adjusted the thickness of the elements to keep the anisotropy in each element within a factor of 10. The experiments on the Antarctica mesh were run with constant coefficient  $\mu = 1$ . The largest run on 262,144 cores had more than 100B unknowns.

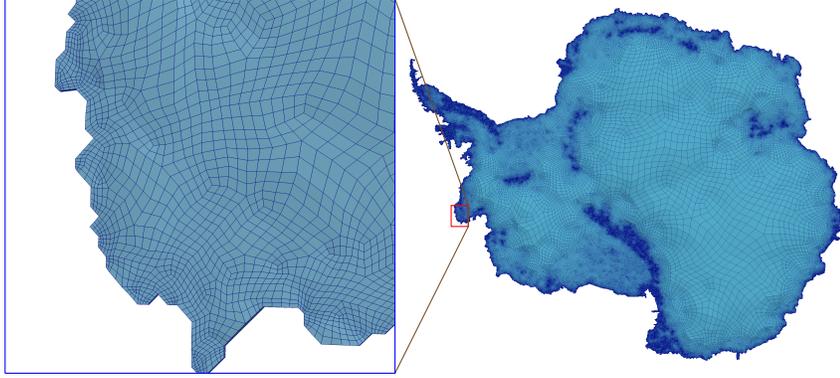


Fig. 6. Antarctica mesh with 45K octrees showing the complexity of the forest. The area under the red box is shown enlarged on the left to highlight the complexity of the mesh. Each element of this mesh is further refined as an octree, resulting in meshes with more than 100B elements. Please zoom in on the electronic version to see further detail.

### B. Parallel Scalability

To study the parallel performance of our implementation, we performed strong (fixed-size) and weak (isogranular) scalability experiments. In all cases, the reported time is the maximum for that stage across all processes. All scalability experiments were performed using multigrid as a preconditioner for the conjugate gradient (CG) method [37]. We evaluated the scalability of our GMG implementation against using AMG directly on the fine mesh. When using AMG as a coarse-grid direct solver for GMG, the number of GMG levels were selected to obtain a coarsest grid (the AMG fine grid) of 32K elements on the shell mesh and to 128K on the Antarctica mesh. These coarse grid problems were solved on a subset of the total processes,  $p$ , with  $p < 32$  for the shell and  $p < 128$  for the Antarctica mesh. We report times for the following stages.

- **Setup:** The setup time accounts for the construction of the mesh and the multigrid hierarchy, including the surrogate meshes. It also includes creating auxiliary vectors that hold the material properties at every level and the diagonal of the system matrix required by the Jacobi smoother.
- **Transfer:** This includes all computation and communication related to intergrid transfers of residuals, such as the restriction and prolongation as well as the transfer to and from the surrogate meshes.
- **Smoother:** Time spent in applying the damped Jacobi smoother. This is largely dominated by the MatVec with the system matrix.
- **Coarse Setup:** Time spent in setting up the coarse grid solver, AMG in our case.
- **Coarse Solve:** Time spent in performing the coarse grid solves.

All reported times are in seconds and correspond to a drop of the relative tolerance by a factor of  $10^{10}$ .

1) *Strong Scalability:* Strong scalability tests were performed with the shell mesh having 124M elements for the variable-coefficient Poisson problem, starting with 512 pro-

cesses and progressively doubling the number of processes up to 131,076 (a 256-fold increase). A total of 5 GMG levels were used in the experiment. We repeat the same experiments using AMG instead of GMG. The results are tabulated in Table I. The total times for both cases are also plotted in Fig. 1. Similar speedups are observed for both approaches up to 8K processes, but AMG starts to deteriorate at higher process counts. Given that the performance of AMG deteriorates drastically at 131K processes we ran an additional test at 80K cores for AMG.

For GMG, we observe excellent scalability for the **smoother** all the way up to 131K processes. The **setup** costs reduce up to 8K processes but increase slightly at higher counts. We observe the best scalability for the smoother since the time complexity for this step is  $\mathcal{O}(N/p)$ . The transfer and setup times start to increase beyond 16K processes. This happens since we dynamically shrink the number of active processes based on the minimal grain size. The number of elements per process at 16K processes is 7.5K, which after the first coarsening drops below our minimal grain size. Therefore for process counts above 16K most processes are idle at all grids except the finest level. The increase in the setup cost is largely due to the balancing and partition of the first coarsened grid. This also causes the transfer costs to slightly increase as seen in Table I. Given that no transfer occurs on the finest grid, the transfer cost is dominated by the transfer on the finest grid and therefore we observe suboptimal behavior beyond 16K. Assume that at a particular grid level we have  $N/p$  elements per process. With uniform refinement, we shall obtain  $N/8p$  elements at the coarser level and therefore the communication bandwidth for a transfer at this grid is  $\mathcal{O}(N/8p)$ . However, if we shrink the number of processes at this grid by a factor of 8 to compensate for the coarsening, the communication bandwidth at the active processes is now  $\mathcal{O}(N/p)$ . We expect that better scaling for the setup and transfer components can be achieved by adjusting the grain size.

2) *Weak Scalability:* Weak scalability tests were performed on the shell mesh using a grain size of 215K elements per

TABLE I  
STRONG SCALING ON THE SPHERICAL SHELL MESH

| Cores        | 512   | 1024  | 2048  | 4096 | 8192 | 16384 | 32768 | 65536 | 80000 | 131072 |
|--------------|-------|-------|-------|------|------|-------|-------|-------|-------|--------|
| AMG Setup    | 10.87 | 5.68  | 3.5   | 2.45 | 1.99 | 3.03  | 4.36  | 6.11  | 10.15 | 645.37 |
| AMG Solve    | 42.39 | 22.64 | 12.65 | 10.8 | 8.79 | 7.6   | 10.87 | 11.95 | 20.64 | 14.51  |
| GMG Setup    | 2.72  | 1.58  | 1.0   | 0.73 | 0.65 | 0.8   | 1.09  | 1.44  | -     | 2.42   |
| GMG Smoother | 36.59 | 19.03 | 9.61  | 5.31 | 3.21 | 1.54  | 0.78  | 0.27  | -     | 0.14   |
| GMG Transfer | 5.65  | 2.86  | 2.77  | 2.56 | 2.87 | 2.53  | 3.27  | 2.16  | -     | 2.33   |
| Coarse Setup | 0.3   | 0.23  | 0.21  | 0.21 | 0.23 | 0.25  | 0.26  | 0.23  | -     | 0.29   |
| Coarse Solve | 0.47  | 0.51  | 0.63  | 0.67 | 0.86 | 1.15  | 1.31  | 1.28  | -     | 1.37   |
| GMG Total    | 44.73 | 24.21 | 14.22 | 9.47 | 7.82 | 6.27  | 6.71  | 5.38  | -     | 6.55   |

Strong scaling shown on a spherical shell mesh (24 octrees) with approximately 124M elements from 512 to 131K processes for the variable coefficient Poisson problem. Shown are timings (in seconds) of individual components of the solve. One pre- and one post-smoothing are used. All runs converged to a relative tolerance of  $10^{-10}$  in 8 iterations. The first two rows report results for using AMG directly on the fine mesh. Since the ML implementation scaled poorly at 131K we performed an additional solve at 80K processes only for AMG. The remaining rows are for GMG using AMG as the direct coarse-grid solver.

TABLE II  
WEAK SCALING ON THE SPHERICAL SHELL MESH

| Cores        | 8     | 64    | 512   | 4096  | 32768 | 262144 |
|--------------|-------|-------|-------|-------|-------|--------|
| AMG Setup    | 8.02  | 9.71  | 10.64 | 12.74 | 41.3  | -      |
| AMG Solve    | 46.56 | 49.41 | 42.02 | 46.01 | 57.3  | -      |
| GMG levels   | 3     | 4     | 5     | 6     | 7     | 8      |
| GMG Setup    | 1.94  | 2.38  | 2.72  | 3.04  | 7.71  | 14.6   |
| GMG Smoother | 34.93 | 35.42 | 36.59 | 41.52 | 42.48 | 45.8   |
| GMG Transfer | 3.6   | 4.12  | 5.65  | 8.38  | 11.35 | 11.96  |
| Coarse Setup | 0.25  | 0.28  | 0.3   | 0.35  | 0.68  | 0.58   |
| Coarse Solve | 0.40  | 0.40  | 0.47  | 0.67  | 1.40  | 1.54   |
| GMG Total    | 41.12 | 42.6  | 45.73 | 53.96 | 63.62 | 74.48  |

Weak scaling on a spherical shell mesh (24 octrees) with 215K elements per process for the variable coefficient Poisson problem. Shown is the time (in seconds). One pre- and one post-smoothing step was performed. All runs converged to a relative tolerance of  $10^{-10}$  in 8 iterations. The first two rows report results for using AMG directly on the fine mesh. Since the ML implementation does not support matrices of dimension larger than  $2^{31}$  due to the use of 32 bit indices, we only report the numbers up to 32K processes. The remaining rows are for GMG using AMG as the direct coarse-grid solver.

process, from process counts of 8 to 262K with both the problem size and the process count growing by a factor of 8. For the problem size increasing by a factor of 32,768 our total runtime only increases by 81%, corresponding to a parallel efficiency of 55%. The same experiment is also repeated using AMG instead of GMG. These results are reported in Table II. Since the ML implementation does not support matrices of dimension larger than  $2^{31}$  due to the use of 32 bit indices, we only report the numbers up to 32K processes.

We also present weak scaling for the constant coefficient Poisson problem on the Antarctica mesh with 400K elements per process in Table III. The largest mesh contains 100B elements and our implementation sustains a double precision floating point rate of 272 teraflops per second (based on hardware performance counters from the *PAPI* library [38]).

### C. Robustness

In order to test the robustness of the multigrid solver in the presence of strong jumps we performed experiments with the variable coefficient Poisson problem using a checkerboard pattern for the coefficients. The coefficients were alternately 1 and  $10^7$  and the frequency of the pattern was varied and the convergence rate was measured. The experiment was performed on an uniformly-refined unit-cube mesh with approximately 16M elements ( $N_x = 2^8$ ). The number of grids was fixed at 6 for all cases, with AMG being used for

TABLE III  
WEAK SCALING OF THE ANTARCTIC ICE-SHEET MESH

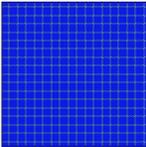
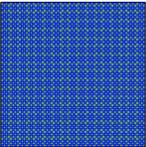
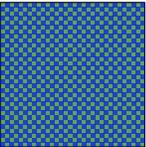
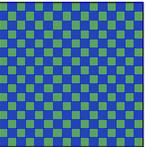
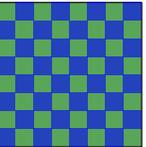
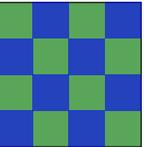
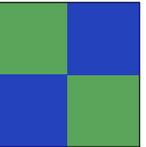
|              | 64    | 512   | 4096  | 32768 | 262144 |
|--------------|-------|-------|-------|-------|--------|
| Setup        | 2.97  | 2.64  | 3.1   | 3.76  | 8.6    |
| Smoother     | 289.7 | 301.5 | 336.3 | 391.3 | 409.1  |
| Transfer     | 7.45  | 8.47  | 11.5  | 11.35 | 15.88  |
| Coarse Setup | 1.85  | 2.13  | 0.82  | 1.27  | 1.63   |
| Coarse Solve | 24.3  | 30.8  | 18.47 | 30.1  | 26.01  |
| Total Time   | 326.3 | 345.5 | 370.2 | 437.8 | 461.2  |

Weak scaling on a large forest (45K octrees), namely the Antarctica mesh with 400K elements per process for the constant coefficient Poisson problem. Four pre- and post-smoothing steps were performed and all runs converged to a relative tolerance of  $10^{-8}$  in 10 iterations. AMG is used as the coarse grid solver. Shown is the time (in seconds) for individual components of the solve.

the coarse grid solve. In Table IV, we report the number of iterations to reduce the 2-norm of the residual by a factor of  $10^{-8}$  for different values of  $K$ , where the frequency of the checkerboard pattern is  $2^K$ . Thus, further work is needed to come up with an appropriate smoothing method for problems with jumps in the coefficient.

In this work we only consider isotropic coefficients, but the effects of anisotropy can be studied by using stretched grids. In this experiment, we changed the anisotropy by stretching the mesh along the  $z$  dimension. The  $x, y$  dimensions were not changed. We considered a stretched mesh generated from a uniform octree with 2M elements and measure the con-

TABLE IV  
EFFECT OF JUMPS IN COEFFICIENTS

|       |   |   |   |   |  |   |   |
|-------|---|---|---|---|--|---|---|
| $\mu$ |  |  |  |  |  |  |  |
| $K$   | 7   | 6   | 5   | 4   | 3  | 2   | 1   |
| Its.  | 107   | 57  | 29  | 15  | 8  | 7   | 7   |

Robustness with respect to jumps in the coefficients of the variable-coefficient Poisson problem. We show the number of iterations required to converge to a relative tolerance of  $10^{-8}$  for different values of  $K$ , which controls the frequency of the jumps (the number of jumps is proportional to  $K$ ). A uniformly-refined unit cube domain with 16M elements was used here.

TABLE V  
EFFECT OF ANISOTROPY

|       |      |     |     |   |   |    |     |
|-------|------|-----|-----|---|---|----|-----|
| $\xi$ | 0.01 | 0.1 | 0.5 | 1 | 2 | 10 | 100 |
| Its.  | 231  | 35  | 9   | 7 | 9 | 36 | 233 |

Effect of stretched elements: The number of iterations required to converge to a relative tolerance of  $10^{-8}$  for different stretching,  $\xi$ , along the  $z$  direction. The unit cube domain with 2M elements was used for this experiment.

vergence rate for different anisotropies. The anisotropy  $\xi$  is measured as the ratio between the  $z$  and the  $x, y$  dimensions. In Table V we report the number of iterations needed to reduce the 2-norm of the residual by a factor of  $10^{-8}$  for different values of  $\xi$ . This experiment shows that the Jacobi smoother used in this work is not able to handle strong anisotropies. Given that the octree performs isotropic refinement, this is a significant limitation of this work. We are currently working on efficient smoothers that are able to overcome this problem.

#### IV. CONCLUSION

We presented a parallel, matrix-free multigrid method on geometry-conforming unstructured forests of octrees. We use algebraic multigrid as the coarse-grid direct solver. Our V-cycle implementation uses only non-blocking point-to-point communications and no barriers, resulting in excellent strong and weak scalability up to 262K cores on ORNL's Jaguar. We achieved such an excellent scalability for a scalar Poisson equation, which has a low floating point to memory operations ratio. Also, note that in all of our runs we use one MPI process per core. It follows that introducing shared memory parallelism would readily provide an immediate  $10\times-100\times$  boost in core count (and performance) thus, suggesting the feasibility of efficient, 10M-core multigrid solvers for unstructured grids with trillions of unknowns.

There are two main extensions for the current work: development of smoothers to handle jump coefficients and anisotropy, and to support higher-order discretizations.

#### ACKNOWLEDGMENT

The authors would like to thank Tobin Isaac for useful discussions and for providing the mesh for the Antarctic ice sheet. Support for this work was provided by: the U.S.

Air Force Office of Scientific Research (AFOSR) Computational Mathematics program under award number FA9550-09-1-0608; the U.S. Department of Energy Office of Science (DOE-SC), Advanced Scientific Computing Research (ASCR), Scientific Discovery through Advanced Computing (SciDAC) program, under award number DE-FG02-09ER25914, and the U.S. National Science Foundation (NSF) Cyber-enabled Discovery and Innovation (CDI) program under awards CMS-1028889 and OPP-0941678, and the PetaApps program under award OCI-0749334. Computing time on the Cray XK6 supercomputer (Jaguar) was provided by the Oak Ridge Leadership Computing Facility at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725. Computing time on the Texas Advanced Computing Center's Lonestar 4 supercomputer was provided by an allocation from TACC.

#### REFERENCES

- [1] M. W. Gee, C. M. Siefert, J. J. Hu, R. S. Tuminaro, and M. G. Sala, "ML 5.0 smoothed aggregation user's guide," Sandia National Laboratories, Tech. Rep. SAND2006-2649, 2006.
- [2] C. Burstedde, L. C. Wilcox, and O. Ghattas, "p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees," *SIAM Journal on Scientific Computing*, vol. 33, no. 3, pp. 1103–1133, 2011.
- [3] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*. London: Academic Press, 2001.
- [4] R. Becker and M. Braack, "Multigrid techniques for finite elements on locally refined meshes," *Numerical Linear Algebra with Applications*, vol. 7, pp. 363–379, 2000.
- [5] P. Bastian and C. Wieners, "Multigrid methods on adaptively refined grids," *Computing in Science Engineering*, vol. 8, no. 6, pp. 44–54, nov.-dec. 2006.
- [6] R. Becker, M. Braack, and T. Richter, "Parallel multigrid on locally refined meshes," *Reactive Flows, Diffusion and Transport*, pp. 77–92, 2007.
- [7] M. F. Adams, H. H. Bayraktar, T. M. Keaveny, and P. Papadopoulos, "Ultrascale implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom," in *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, ser. SC '04. Washington, DC, USA: IEEE Computer Society, 2004.
- [8] D. J. Mavriplis, M. J. Aftosmis, and M. Berger, "High resolution aerospace applications using the NASA Columbia Supercomputer," in *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2005, p. 61.
- [9] R. S. Sampath, S. S. Adavani, H. Sundar, I. Lashuk, and G. Biros, "Dendro: Parallel algorithms for multigrid and AMR methods on 2:1 balanced octrees," in *SC'08: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. ACM/IEEE, 2008.

- [10] R. S. Sampath and G. Biros, "A parallel geometric multigrid method for finite elements on octree meshes," *SIAM Journal on Scientific Computing*, vol. 32, no. 3, pp. 1361–1392, 2010.
- [11] B. K. Bergen, F. Hülsemann, and U. Rüde, "Is  $1.7 \times 10^{10}$  unknowns the largest finite element system that can be solved today?" in *SC '05: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. ACM/IEEE, 2005.
- [12] F. Hülsemann, M. Kowarschik, M. Mohr, and U. Rüde, "Parallel geometric multigrid," in *Numerical Solution of Partial Differential Equations on Parallel Computers*, ser. Lecture Notes in Computational Science and Engineering, Vol. 51, A. Bruaset and A. Tveito, Eds. Heidelberg: Springer-Verlag, 2005, pp. 165–208.
- [13] B. Gmeiner, T. Gradl, H. Kostler, and U. Rüde, "Highly parallel geometric multigrid algorithm for hierarchical hybrid grids," in *Hierarchical Methods for Dynamics in Complex Molecular Systems*, J. Grotendorst, G. Sutmann, G. Gompper, and D. Marx, Eds. Forschungszentrum Jülich, 2012, p. 323.
- [14] A. Vogel, S. Reiter, M. Rupp, A. Nägel, and G. Wittum, "UG 4—a novel flexible software system for simulating PDE based models on high performance computers," *Computing and Visualization in Science*, 2012, (submitted).
- [15] S. Reiter, A. Vogel, I. Heppner, M. Rupp, and G. Wittum, "A massively parallel geometric multigrid solver on hierarchically distributed grids," *Computing and Visualization in Science*, 2012, (submitted).
- [16] F. Günther, M. Mehl, M. Pögl, and C. Zenger, "A cache-aware algorithm for PDEs on hierarchical data structures based on space-filling curves," *SIAM Journal on Scientific Computing*, vol. 28, no. 5, pp. 1634–1650, 2006.
- [17] H.-J. Bungartz, M. Mehl, and T. Weinzierl, "A parallel adaptive Cartesian PDE solver using space-filling curves," *Euro-Par 2006 Parallel Processing*, pp. 1064–1074, 2006.
- [18] H.-J. Bungartz, M. Mehl, T. Neckel, and T. Weinzierl, "The PDE framework Peano applied to fluid dynamics: an efficient implementation of a parallel multiscale fluid dynamics solver on octree-like adaptive Cartesian grids," *Computational Mechanics*, vol. 46, no. 1, pp. 103–114, 2010.
- [19] S. Turek, D. Göddeke, C. Becker, S. Buijssen, and H. Wobker, "FEAST – realization of hardware-oriented numerics for HPC simulations with finite elements," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 16, pp. 2247–2265, 2010.
- [20] E. Chow, R. D. Falgout, J. J. Hu, R. S. Tuminaro, and U. M. Yang, "A survey of parallelization techniques for multigrid solvers," in *Parallel Processing for Scientific Computing*, M. A. Heroux, P. Raghavan, and H. D. Simon, Eds. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2006.
- [21] R. Falgout, J. Jones, and U. Yang, "The design and implementation of hypre, a library of parallel high performance preconditioners," in *Numerical Solution of Partial Differential Equations on Parallel Computers*, A. Bruaset and A. Tveito, Eds. Springer-Verlag, 2006, vol. 51, pp. 267–294.
- [22] A. Baker, R. Falgout, T. Gamblin, T. Kolev, M. Schulz, and U. Yang, "Scaling algebraic multigrid solvers: On the road to exascale," *Competence in High Performance Computing 2010*, pp. 215–226, 2012.
- [23] A. Baker, R. Falgout, T. Kolev, and U. Yang, "Scaling hypre's multigrid solvers to 100,000 cores," *High-Performance Scientific Computing*, pp. 261–279, 2012.
- [24] H. De Sterck, U. M. Yang, and J. J. Heys, "Reducing complexity in parallel algebraic multigrid preconditioners," *SIAM Journal on Matrix Analysis and Applications*, vol. 27, no. 4, pp. 1019–1039, 2006.
- [25] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, "An overview of the Trilinos project," *ACM Transactions on Mathematical Software*, vol. 31, no. 3, pp. 397–423, Sep. 2005.
- [26] N. Bell, S. Dalton, and L. Olson, "Exposing fine-grained parallelism in algebraic multigrid methods," NVIDIA, Tech. Rep. NVR-2011-002, June 2011.
- [27] G. Haase, M. Liebmann, C. Douglas, and G. Plank, "A parallel algebraic multigrid solver on graphics processing units," in *High Performance Computing and Applications*, ser. Lecture Notes in Computer Science, W. Zhang, Z. Chen, C. Douglas, and W. Tong, Eds. Springer Berlin/Heidelberg, 2010, vol. 5938, pp. 38–47.
- [28] V. Heuveline, D. Lukarski, N. Trost, and J.-P. Weiss, "Parallel smoothers for matrix-based geometric multigrid methods on locally refined meshes using multicore CPUs and GPUs," in *Facing the Multicore - Challenge II*, ser. Lecture Notes in Computer Science, R. Keller, D. Kramer, and J.-P. Weiss, Eds. Springer Berlin/Heidelberg, 2012, vol. 7174, pp. 158–171.
- [29] T. Tu, D. R. O'Hallaron, and O. Ghattas, "Scalable parallel octree meshing for terascale applications," in *SC '05: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. ACM/IEEE, 2005.
- [30] H. Sundar, R. Sampath, and G. Biros, "Bottom-up construction and 2:1 balance refinement of linear octrees in parallel," *SIAM Journal on Scientific Computing*, vol. 30, no. 5, pp. 2675–2708, 2008.
- [31] T. Isaac, C. Burstedde, and O. Ghattas, "Low-cost parallel algorithms for 2:1 octree balance," in *Proceedings of the 26th IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2012.
- [32] C. Burstedde, O. Ghattas, G. Stadler, T. Tu, and L. C. Wilcox, "Towards adaptive mesh PDE simulations on petascale computers," in *Proceedings of Teragrid '08*, 2008.
- [33] L. C. Wilcox, G. Stadler, C. Burstedde, and O. Ghattas, "A high-order discontinuous Galerkin method for wave propagation through coupled elastic-acoustic media," *Journal of Computational Physics*, vol. 229, no. 24, pp. 9373–9396, 2010.
- [34] G. Stadler, M. Gurnis, C. Burstedde, L. C. Wilcox, L. Alisic, and O. Ghattas, "The dynamics of plate tectonics and mantle flow: From local to global scales," *Science*, vol. 329, no. 5995, pp. 1033–1038, 2010.
- [35] C. Burstedde, O. Ghattas, M. Gurnis, E. Tan, T. Tu, G. Stadler, L. C. Wilcox, and S. Zhong, "Scalable adaptive mantle convection simulation on petascale supercomputers," in *SC08: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM/IEEE, 2008.
- [36] A. M. Le Brocq, A. J. Payne, and A. Vieli, "An improved Antarctic dataset for high resolution numerical ice sheet models (ALBMAP v1)," *Earth System Science Data*, vol. 2, no. 2, pp. 247–260, 2010. [Online]. Available: <http://www.earth-syst-sci-data.net/2/247/2010/>
- [37] O. Tatebe and Y. Oyanagi, "Efficient implementation of the multigrid preconditioned conjugate gradient method on distributed memory machines," in *Supercomputing '94. Proceedings*. IEEE, 1994, pp. 194–203.
- [38] Performance applications programming interface (PAPI). [Online]. Available: <http://icl.cs.utk.edu/papi/>