SIAM J. SCI. COMPUT.
Vol. 40, No. 4, pp. C581–C604

SCALING STRUCTURED MULTIGRID TO 500K+ CORES THROUGH COARSE-GRID REDISTRIBUTION*

ANDREW REISNER†, LUKE N. OLSON†, AND J. DAVID MOULTON‡

Abstract. The efficient solution of sparse, linear systems resulting from the discretization of partial differential equations is crucial to the performance of many physics-based simulations. The algorithmic optimality of multilevel approaches for common discretizations makes them good candidates for an efficient parallel solver. Yet, modern architectures for high-performance computing systems continue to challenge the parallel scalability of multilevel solvers. While algebraic multigrid methods are robust for solving a variety of problems, the increasing importance of data locality and cost of data movement in modern architectures motivates the need to carefully exploit structure in the problem. Robust logically structured variational multigrid methods, such as black box multigrid, maintain structure throughout the multigrid hierarchy. This avoids indirection and increased coarse-grid communication costs typical in parallel algebraic multigrid. Nevertheless, the parallel scalability of structured multigrid is challenged by coarse-grid problems where the overhead in communication dominates computation. In this paper, an algorithm is introduced for redistributing coarse-grid problems through incremental agglomeration. Guided by a predictive performance model, this algorithm provides robust redistribution decisions for structured multilevel solvers. A two-dimensional diffusion problem is used to demonstrate the significant gain in performance of this algorithm over the previous approach that used agglomeration to one processor. In addition, the parallel scalability of this approach is demonstrated on two large-scale computing systems, with solves on up to 500K+ cores.

Key words. multigrid, structure, parallel, scalability, stencil

AMS subject classifications. 65F50, 65Y05, 65N55

DOI. 10.1137/17M1146440

1. Introduction.

The efficient solution of large, sparse linear systems resulting from the discretization of elliptic partial differential equations (PDEs) is crucial to the performance of many numerical simulations. Although there has been significant progress in developing general algebraic multigrid (AMG) solvers [34], modern high-performance computing (HPC) architectures continue to pose significant challenges to parallel scalability and performance (e.g., [3, 14, 5]). These challenges include reducing data movement, increasing arithmetic intensity, and identifying opportunities to improve resilience and are more readily addressed in settings where problem structure can be identified and exploited. For example, robust structured variational

*Submitted to the journal's Software and High-Performance Computing section September 6, 2017; accepted for publication (in revised form) May 8, 2018; published electronically July 17, 2018.
http://www.siam.org/journals/sisc/40-4/M114644.html

Funding: This work was carried out under the auspices of the National Nuclear Security Administration of the U.S. Department of Energy, at the University of Illinois at Urbana–Champaign under award DE-NA0002374, and at Los Alamos National Laboratory under contract DE-AC52-06NA25396, and was partially supported by the Advanced Simulation and Computing/Advanced Technology Development and Mitigation Program. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the State of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana–Champaign and its National Center for Supercomputing Applications. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

†Department of Computer Science, University of Illinois at Urbana–Champaign, Urbana, IL 61801 (areisne2@illinois.edu, lukeo@illinois.edu).

‡Applied Mathematics and Plasma Physics, Los Alamos National Laboratory, Los Alamos, NM 87544 (moulton@lanl.gov).

multigrid methods, such as black box multigrid (BoxMG) [10, 11], take advantage of direct memory addressing and fixed stencil patterns throughout the multigrid hierarchy to realize a 10× speed-up over AMG for heterogeneous diffusion problems [24]. In addition, the communication patterns in a parallel BoxMG solve are fixed and predictable throughout the multigrid hierarchy. Here we explore using this information to improve the parallel scalability and performance of elliptic solves for problems with structure.

The meshing strategy used in the discretization of a PDE has a significant impact on the amount of structure that can be exploited by the solver. For example, single-block locally structured grids can be mapped to conform to smooth nonplanar geometries [9] and can use embedded boundary discretization techniques to add additional flexibility to the representation of object boundaries. Robust structured variational methods, such as BoxMG, are directly applicable to these cases. In contrast, fully unstructured grids can capture very complex geometries, including nonsmooth features over a range of scales, but demand general algebraic multilevel solvers, such as AMG or smoothed aggregation AMG. The needs of many applications lie between these extremes, and a variety of adaptive or multimesh strategies have been developed to preserve structure and enable its use in the discretization and solver. These approaches generally lead to specialized hybrid solvers, favoring structured techniques at higher levels of refinement and unstructured techniques below a suitably chosen coarse level [32, 15]. In [15], memory efficient matrix-free geometric methods are used on fine levels with hybrid hierarchical grids to solve a problem reaching 1.1×10^{13} degrees of freedom (dof). While these hybrid solvers present a variety of challenges, a single-block logically structured solver is a critical component of their design and performance.

A common approach to parallel multigrid solvers for PDEs is to partition the spatial domain across available processor cores. However, on coarser levels, the local problem size decreases and the communication cost begins to impact parallel performance. A natural approach to alleviate this problem is to gather the coarsest problem to a single processor (or redundantly to all the processors) and to solve it with a serial multigrid cycle. This approach was first motivated by a performance model of early distributed memory clusters [17] where core counts were quite small, and it was used in the initial version of parallel BoxMG. Unfortunately, as the weak scaling study in Figure 1 shows, this approach scales linearly with the number of cores and hence is not practical for modern systems. A modification of this approach that gathers the coarse problem to a multicore node [28], and then leverages OpenMP threading on that node, improves the performance but does not address the scaling. This challenge of reducing communication costs at coarser levels is even more acute in AMG methods, and this led to exploration of agglomeration to redundant coarse problems at higher levels [3], as well as redistribution of smaller portions of the coarse problem to enhance scalability [14], leading to approximately 2× speedup in the solve phase.

An alternative to this two-level gather-to-one approach is a more conservative redistribution strategy that can be applied recursively as the problem is coarsened. In single-block structured solvers, the decision to redistribute data is driven by balancing communication costs in relaxation with diminishing local work. This approach was first considered in a structured setting [35] and used a heuristic to guide recursive application of nearest neighbor agglomeration. Later, in the Los Alamos AMG solver, a heuristic was developed to guide the reduction of the number of active cores at each level by a power of two [21]. In [13], agglomeration is performed on a per-process basis in AMG. When the number of unknowns on a process falls below a specified

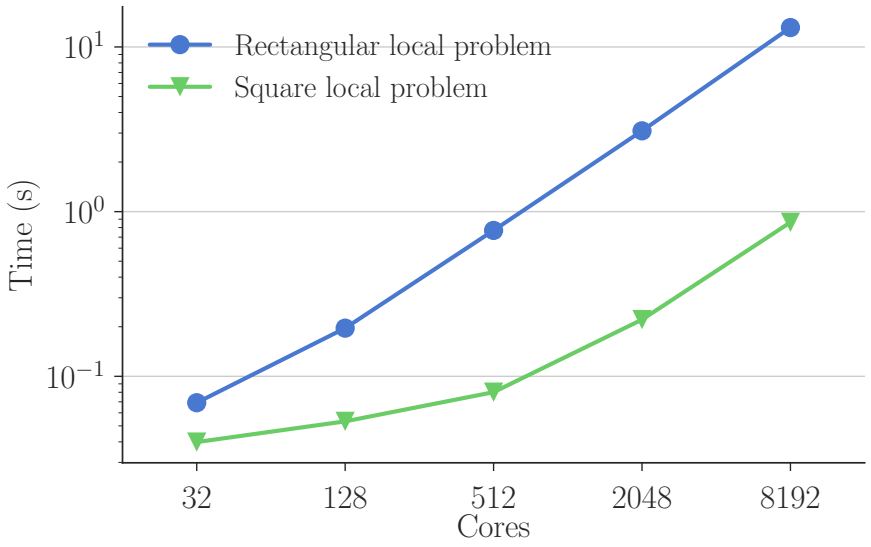


FIG. 1. *Weak scaling on Blue Waters for a local problem size of* 568×71 *(rectangular) and* 200×200 *(square) using* $V(2,1)$*-cycles with a gather-to-one serial BoxMG coarse-grid solver.*

threshold, the process donates their local problem to their neighbor with the fewest unknowns. Hierarchical agglomeration is performed for hierarchical distributed grids in [30] beginning with the coarse problem on one process and performing incremental refinement of the processor grid. In [31], uniform agglomeration is used to minimize communication costs during geometric coarsening. The necessity of processor agglomeration at scale was noted in [25]—motivating the addition of an agglomeration framework in PETSc. While manual specification of agglomeration is required, performance model guided agglomeration is proposed as future work. While these works use recursive agglomeration to address parallel scalability of coarse-grid problems, they use fixed agglomeration strategies and do not explore optimizing agglomeration through predictive performance models.

In this paper, an optimized redistribution algorithm is proposed for robust structured multigrid methods that balances the computation and communication costs at each level. The structured setting enables the enumeration of possible coarse-grid configurations, and a performance model is developed to support optimization of the coarsening path that is selected through these configurations. The utility of this approach is demonstrated through scaling studies extending beyond 100K cores on two modern supercomputers.

The remainder of this paper is organized as follows. Section 2 highlights relevant features of robust variational multigrid methods, examines the domain decomposition implementation in BoxMG, and proposes a redistribution algorithm that can be applied recursively. The performance model for this parallel algorithm with redistribution is developed in section 3, and the optimization algorithm is presented in section 4. Scaling studies are presented in section 5 demonstrating the efficacy of the proposed method, and section 6 gives conclusions.

2. Robust variational multigrid on structured grids. In the case of a symmetric, positive definite matrix problem, a Galerkin (variational) coarse-grid operator

is effective in defining the coarse-level problems because it minimizes the error in the range of interpolation [7]. This variational operator is formed as the triple matrix product of the restriction (transpose of interpolation), the fine-grid operator, and the interpolation, making it well suited for black box and algebraic multilevel solver algorithms. In the case of structured grids, the complexity of the coarse-grid operators is bounded, and the stencil pattern can be fixed a priori.

However, in this approach, the interpolation must be sufficiently accurate to ensure the variational coarse-grid operator satisfies the approximation property [34]. For example, in two-dimensional (2D) diffusion problems with discontinuous coefficients, bilinear interpolation is not accurate across discontinuities because the gradient of the solution is not continuous. Thus, a key element in robust multigrid methods is *operator-induced* interpolation [8, 34], which uses the matrix problem to construct intergrid transfer operators. In the case of structured grid problems, operator-induced interpolation is naturally motivated by noting the normal component of the flux is continuous [10], and its impact on the properties of the variational coarse-grid operator is understood through its connection to homogenization [27, 23]. This approach has natural extensions to nonsymmetric problems as well [11, 36].

Robust methods also require careful consideration of the smoothing operator. Although Gauss–Seidel is effective for many problems, anisotropy often demands alternating line smoothing or plane smoothing [8, 34]. With coarse-grid operators and interpolation defined, a standard multigrid cycling is used, for example, a V-cycle as in Algorithm 1.

Algorithm 1: Multilevel V-cycle

Input: u_{L-1} fine-grid initial guess
f_{L-1} fine-grid right-hand side
$A_0, \ldots, A_{L-1}$
$P_1, \ldots, P_{L-1}$

Output: u_{L-1} fine-grid iterative solution

for $l = L-1, \ldots, 1$ **do**
  $\texttt{relax}(A_l, u_l, f_l, \nu_1)$ {relax ν_1 times}
  $r_l = f_l - A_l u_l$ {compute residual}
  $f_{l-1} = P_l^T r_l$ {restrict residual}
$u_0 = \texttt{solve}(A_0, f_0)$ {coarse-grid direct solve}
for $l = 1, \ldots, L-1$ **do**
  $u_l = u_l + P_l u_{l-1}$ {interpolate and correct}
  $\texttt{relax}(A_l, u_l, f_l, \nu_2)$ {relax ν_2 times}

In this paper, interpolation and coarse-grid operators are constructed using the implementation in BoxMG [1, 10, 27] with standard coarsening by a factor of 2; however, the approach applies to any structured method. BoxMG is used because its operator-induced interpolation is designed to handle discontinuous coefficients, and additional heuristics ensure optimal performance for Dirichlet, Neumann, and Robin boundary conditions on logically structured grids of any dimension (i.e., it is not restricted to grids $2^k + 1$). BoxMG supports vertex- and cell-based discretizations of the diffusion equation on logically structured grids that lead to nearest neighbor stencils (i.e., five- and nine-point operators in two dimensions). In addition, only

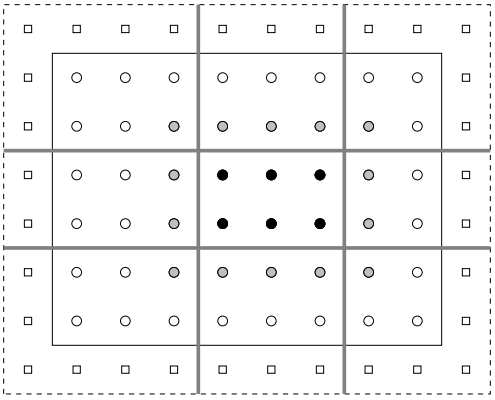


FIG. 2. *Domain partition. Circles represent dof, with* ● *denoting points on process k,* ◍ *representing halo points needed for communication from other processors, and* ○ *points on other processors. Fictitious points on the domain boundary are denoted by* □.

the fine-grid problem is specified; coarse-grid operators are constructed through the variational (Galerkin) product. The package is also released as open source in the Cedar Project [26].

2.1. Domain decomposition parallel implementation. The distributed memory parallel implementation of BoxMG divides the problem domain among available processors. The processors are arranged in a structured grid and points in the fine-grid problem are divided among processors in each dimension—see Figure 2. Since the computation is structured, interprocess communication occurs through nearest neighbor halo updates with a halo width of 1. An important feature of BoxMG is bounded complexity in the coarse-grid operator. By exploiting the structure of the problem, BoxMG produces coarse-grid problems with a fixed structure. This results in known communication patterns and guaranteed data locality.

Examining the distributed memory parallel implementation of BoxMG in the context of parallel scalability, many of the operations are stencil based computations with halo updates. Since the operations are relatively local, they are not expected to significantly limit parallel scalability. For this paper, the particular focus is in parallel decisions at coarse grids. To balance useful local work with communication costs, agglomeration methods are used to redistribute coarse problems. A straightforward approach is agglomeration to one task. This task is then mapped either to a single processor or to all processors redundantly. This method of agglomeration was used in the initial distributed memory implementation of BoxMG; while effective for low processor counts it suffers at scale since the global coarse problem grows linearly with the number of processors. As seen in Figure 1, this is especially true with rectangular local problems for which the coarsest local problem size is larger, and hence, the linear growth is faster and overall parallel scaling is lower. In addition to agglomerating to one task, the coarse problem can be redistributed by agglomerating to n tasks, where n is less than the number of processors. The problem then becomes choosing a desirable value of n which is dependent on the problem distribution on the processor grid. This agglomeration can be applied recursively to gradually reduce the size of the processor grid as the size of the global coarse-grid problem is reduced.

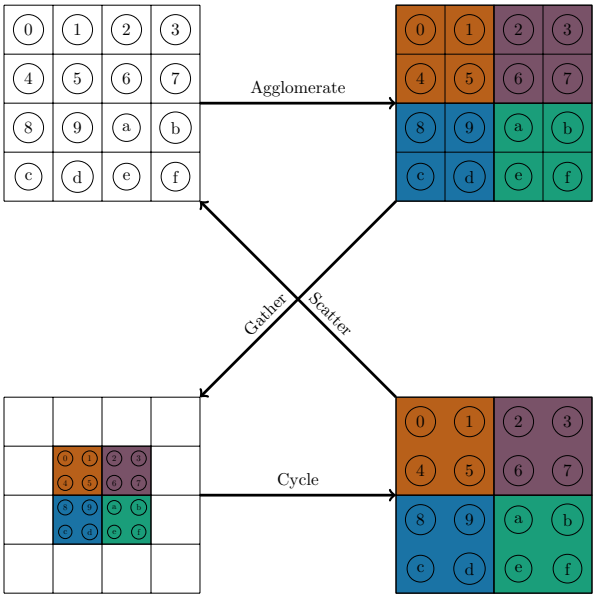


FIG. 3. *A redistribution of a* 4×4 *processor grid to* 2×2 *processor blocks. The boxes represent the processing elements and the circles represent their respective coarse-grid local problems.*

2.2. Redistribution. To extend parallel coarsening in a structured setting, the algorithm introduced in this paper aims to redistribute the coarse-grid problem to an incrementally smaller subset of processors. Parallel coarsening continues until a parameterized minimum local problem size is reached. At this point, a set of possible redistributions is enumerated. Here, a redistribution is given by $(p^0, p^1, \ldots, p^{D-1})$, where p^k is the number of processors in space direction k. These redistributions are evaluated based on a cost derived through a performance model. An optimal redistribution sequence is then selected. It is important to note that since a redistributed problem on a given level also limits parallel coarsening, the selection algorithm is designed to be applied recursively to obtain the highest efficiency possible for the multigrid cycle. Section 4.1 discusses the redistribution of a grid of processors to coarser processor grids.

To redistribute the coarse-grid problem on a smaller number of processors, the processor grid is agglomerated into processor blocks. This agglomeration is performed by dimension to maintain a distributed tensor-product grid structure. Each processor block is then mapped to one task in the redistributed solver.

To map the coarse tasks to processors, two approaches are considered. The first approach uses one processor from each processor block. This is visualized in Figure 3, where the following steps are taken:

1. *Agglomerate*: processors grouped into blocks to define coarse tasks.
2. *Gather*: processors within each block perform gather on coarse problem.
3. *Cycle*: cycling continues with redistributed problem.
4. *Scatter*: iterative solution scattered after redistributed cycle completes.

The second approach employs redundancy by mapping each processor in the processor block to the coarse task. This is visualized in Figure 4 with the following steps:

1. *Agglomerate*: processors grouped into blocks to define coarse tasks.
2. *Allgather*: processors within each block perform allgather on coarse problem.
3. *Cycle*: cycling continues redundantly with redistributed problem.

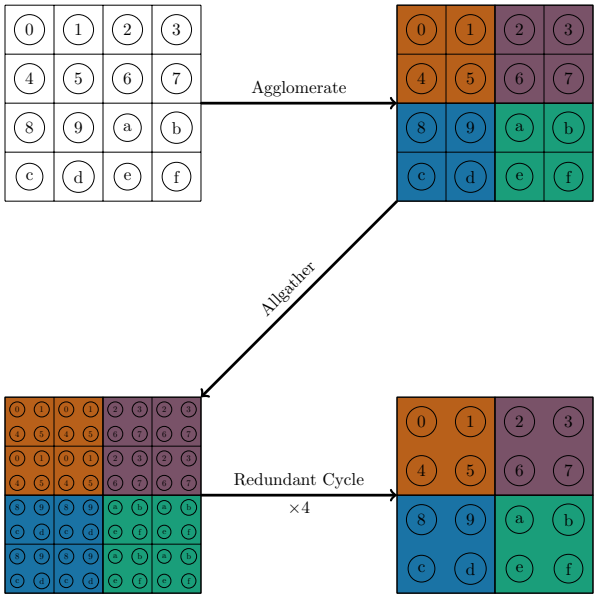


FIG. 4. *A redundant redistribution of a* 4×4 *processor grid to* 2×2 *processor blocks. The boxes represent the processing elements and the circles represent their respective coarse-grid local problems.*

While the second approach avoids an additional communication phase at the end of each cycle and adds an opportunity for resilience through redundant cycling, the first approach avoids the increased network usage involved in simultaneous coarse cycles. Algorithm 2 supplements Algorithm 1 with steps needed for redistribution. The algorithm is annotated with parallel communication required for each step.

3. Performance model. In this section, a performance model is introduced for the BoxMG V-cycle (see Algorithm 2). The model helps identify the parallel performance limitations of the V-cycle, particularly at coarse levels in the multigrid hierarchy, and also provides a cost metric that is used to guide the coarse-level redistribution algorithm introduced in section 4.

A key kernel in the V-cycle is that of matrix-vector multiplication. Since the stencil-based computations for this operation are relatively uniform, the cost of parallel communication in matrix-vector multiplication is accurately modeled with a *postal* model [4], leading to a total cost of

$$T = \underbrace{n_f \cdot \gamma}_{\text{computation}} + \underbrace{\alpha + m \cdot \beta}_{\text{communication}} \tag{1}$$

with n_f denoting the number of floating point operations, γ a measure of the computation rate or inverse *effective* FLOP rate, α the interprocessor latency, $1/\beta$ the network bandwidth, and m the number of bytes in an MPI message. The value γ is determined by measuring the computation time of a local stencil-based matrix-vector product, which is a memory bandwidth bound operation. The values α and β are determined through standard machine benchmarks such as `mpptest`.[1] As an example, the parameters derived for a nine-point 2D stencil on Blue Waters, a Cray

[1] http://www.mcs.anl.gov/research/projects/mpi/mpptest/.

Algorithm 2: Multilevel V-cycle with redistribution

Input: u_{L-1} fine-grid initial guess
f_{L-1} fine-grid right-hand side
$A_0, \ldots, A_{L-1}$
$P_1, \ldots, P_{L-1}$
$p_0, \ldots, p_{L-1}$ number of processors on each level

Output: u_{L-1} fine-grid iterative solution

for $l = L-1, \ldots, 1$ **do**
  `relax`(A_l, u_l, f_l, ν_1) {halo exchange}
  $r_l = f_l - A_l u_l$ {halo exchange}
  $f_{l-1} = P_l^T r_l$
  if $p_l > p_{l-1}$
    `gather_rhs`(f_{l-1}, p_l, p_{l-1}) {local gather}
$u_0 =$ `solve`(A_0, f_0)
for $l = 1, \ldots, L-1$ **do**
  if $p_l > p_{l-1}$
    `scatter_sol`(u_{l-1}, p_l, p_{l-1}) {local scatter}
  $u_l = u_l + P_l u_{l-1}$ {halo exchange}
  `relax`(A_l, u_l, f_l, ν_2) {halo exchange}

TABLE 1
Model parameters on Blue Waters.

α	β	γ
0.65 µs	5.65 ns/B	0.44 ns/flop

XE6 machine at the National Center for Supercomputing Applications,[2] are listed in Table 1. A more accurate model for communication may be used, particularly for multiple communicating cores with large message sizes [18] or to account for network contention [6], which may play a prominent role in communication.

In an L-level multigrid V-cycle, Algorithm 1 is modeled through

$$T_{\text{V-cycle}} = T_{\text{cgsolve}} + \sum_{l=1}^{L-1} T^l_{\text{smooth}} + T^l_{\text{residual}} + T^l_{\text{restrict}} + T^l_{\text{interp}} + T^l_{\text{agglomerate}}. \tag{2}$$

In the following expressions, each component of (2) represents the time taken in the *actual* implementation in BoxMG and does not necessarily form a lower bound on each portion of the computation. In the model parameters below, D denotes the number of dimensions in the problem, n_d^l the number of local grid points in dimension d on level l, and n_s the number of points in the stencil. Grid quantities involved in communication and computation are assumed to be 8-byte double-precision floating point numbers. The dimension of the parallel decomposition is also assumed to match the dimension of the problem. The communication required for a halo exchange of

[2] https://bluewaters.ncsa.illinois.edu/blue-waters.

width 1 in D dimensions on level l is modeled as

$$T^l_{\text{exchange}}(D) = 2 \cdot D \cdot \alpha + 2 \cdot \sum_{d=0}^{D-1} n^l_d \cdot 8 \cdot \beta. \tag{3}$$

Smoothing, using Gauss–Seidel with n_c colors, results in

$$T^l_{\text{smooth}} = 2 \cdot n_s \cdot \prod_{d=0}^{D-1} n^l_d \cdot (\nu_1 + \nu_2) \cdot \gamma + n_c \cdot (\nu_1 + \nu_2) \cdot T^l_{\text{exchange}}(D), \tag{4}$$

where the factor of 2 accounts for both multiply and addition operations. Likewise, the residual computation is

$$T^l_{\text{residual}} = 2 \cdot n_s \cdot \prod_{d=0}^{D-1} n^l_d \cdot \gamma + T^l_{\text{exchange}}(D). \tag{5}$$

Since it is unnecessary to communicate halo regions in restriction, the computation is entirely local, leading to

$$T^l_{\text{restrict}} = 2 \cdot n_s \cdot \prod_{d=0}^{D-1} n^l_d \cdot \gamma. \tag{6}$$

Following [12], the interpolation and correction computes

$$u^l \leftarrow u^l + I^l_{l-1} u^{l-1} + r^l / C, \tag{7}$$

where r^l is the previously computed residual and C is the center, diagonal stencil coefficient of the operator. Interpolation for edges (see Figure 5) yields

$$u^l \leftarrow \underbrace{u^l + \omega_{\text{w}} u^{l-1}_{\text{w}} + \omega_{\text{e}} u^{l-1}_{\text{e}} + r^l / C}_{5FLOPs} \tag{8}$$

for the x-direction (and similar for the y-direction). Likewise, the interior stencil (see Figure 5) yields

$$u^l \leftarrow \underbrace{u^l + \omega_{\text{sw}} u^{l-1}_{\text{sw}} + \omega_{\text{se}} u^{l-1}_{\text{se}} + \omega_{\text{nw}} u^{l-1}_{\text{nw}} + \omega_{\text{ne}} u^{l-1}_{\text{ne}} + r^l / C}_{9FLOPs}. \tag{9}$$

Here interpolation is given as weights stored at the coarse points. For example, the interpolation stencil stored at a given coarse point is

$$\begin{bmatrix} \omega_{\text{nw}} & \omega_{\text{n}} & \omega_{\text{ne}} \\ \omega_{\text{w}} & & \omega_{\text{e}} \\ \omega_{\text{sw}} & \omega_{\text{s}} & \omega_{\text{se}} \end{bmatrix}.$$

In total (with injection), interpolation in two dimensions is modeled as

$$T^l_{\text{interp}} = \left(\prod_{d=0}^{1} n^l_d + 20 \cdot \prod_{d=0}^{1} n^{l-1}_d + 6 \cdot \sum_{d=0}^{1} n^{l-1}_d \right) \cdot \gamma + T^l_{\text{exchange}}(2). \tag{10}$$

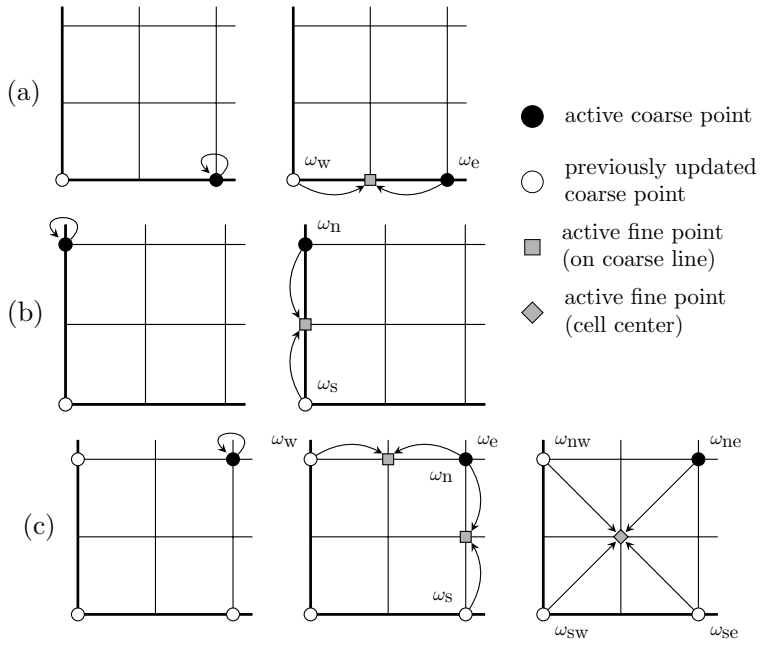


FIG. 5. *Interpolation computation in two dimensions.* (a) *The top two grids show computation performed on the first coarse line in the first dimension.* (b) *Similarly, the middle two grids show computation performed on the first coarse line in the second dimension. Each coarse point performs injection to the corresponding fine point. Interpolation to the preceding fine point embedded in the coarse line is then computed using the surrounding coarse points.* (c) *The bottom three grids show computation performed for interior coarse points. For each coarse point, the corresponding fine point is injected. The preceding coarse points in each dimension are then interpolated. Finally, the logical cell center preceding the coarse point is interpolated using the surrounding coarse points.*

Using a similar derivation, interpolation in three dimensions is

$$T^l_{\text{interp}} = \left(\prod_{d=0}^{2} n^l_d + 60 \cdot \prod_{d=0}^{2} n^{l-1}_d + 15 \cdot n^{l-1}_0 \cdot n^{l-1}_2 + 6 \cdot n^{l-1}_1 \cdot n^{l-1}_2 + n^{l-1}_2 \right) \cdot \gamma + T^l_{\text{exchange}}(3). \tag{11}$$

To agglomerate the coarse problem, the right-hand side is gathered within processor blocks before the redistributed cycle begins and approximate solution scattered after the redistributed cycle completes. This time is then given by

$$T^l_{\text{agglomerate}} = \begin{cases} T^l_{\text{gather\_rhs}} + T^l_{\text{scatter\_sol}} & \text{if } p^l > p^{l-1}, \\ 0 & \text{else,} \end{cases} \tag{12}$$

where $T^l_{\text{gather\_rhs}}$ and $T^l_{\text{scatter\_sol}}$ represent the time to gather and scatter the right-hand side and solution, as follows. The number of processors within a processor block and the local problem size for a processor in a processor block are given as

$$p^l_{\text{block}} = \prod_{d=0}^{D-1} \left\lceil \frac{p^{l+1}_d}{p^l_d} \right\rceil, \quad n^l_{\text{block}} = \prod_{d=0}^{D-1} \left\lceil \frac{N^l_d}{p^l_d} \right\rceil.$$

For this operation, an MPI `allgather` or `gather/scatter` is used depending on whether the redistribution is redundant. Following [33], the cost of these collective

operations is given by

$$T^l_{\text{gather\_rhs}} = \lceil \log_2(p^l_{\text{block}}) \rceil \cdot \alpha + n^l_{\text{block}} \cdot \frac{p^l_{\text{block}} - 1}{p^l_{\text{block}}} \cdot 8\beta, \tag{13}$$

$$T^l_{\text{scatter\_sol}} = \begin{cases} 0 & \text{if redundant,} \\ T^l_{\text{gather\_rhs}} & \text{else.} \end{cases} \tag{14}$$

Finally, the time for the solve on the very coarsest level after agglomerating to one processor is the cost of a Cholesky direct solve:

$$T_{\text{cgsolve}} = T^0_{\text{agglomerate}} + \left(\prod_{d=0}^{D-1} N^0_d \right)^2 \cdot \gamma. \tag{15}$$

This assumes the Cholesky factorization has been computed and stored in the setup phase.

This performance model provides a basic predictive model to begin exploring the guided redistribution algorithm proposed in this paper. In contrast to [16, 15], the performance model in the present work is used to provide an estimate of the communication and computation costs involved in the multigrid solve phase that can be used to predict the impact of different coarse-grid redistribution options on the overall solve time.

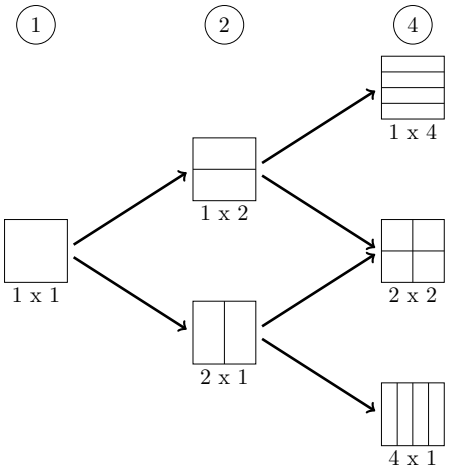
4. Optimized parallel redistribution algorithm.

4.1. Coarse processor grid enumeration. To enumerate potential redistributions, the fine-grid tasks described by the processor grid are agglomerated into coarser tasks called processor blocks. In agglomerating by dimension, the processor blocks form a coarser tensor product grid. The potential redistributions are enumerated by beginning with a 1×1 processor block and refining greedily by dimension with respect to the agglomerated local problem size.

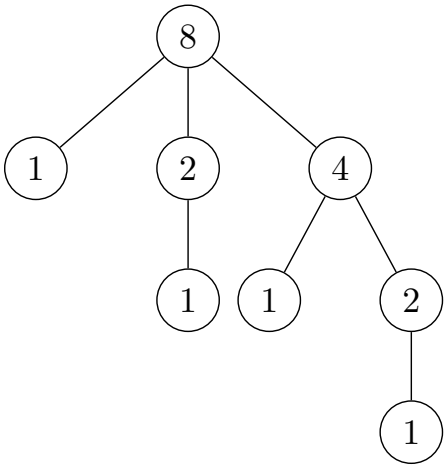
Figure 6 illustrates this process. A dimension is considered for refinement if the refinement is feasible. A refinement is feasible if the number of processor blocks in that dimension after refinement is less than or equal to the initial fine-grid task size in that dimension. If refinements in multiple dimensions are feasible, the dimension with the largest agglomerated local problem size is chosen. As refinement is chosen in each step by dimension, the number of enumerated processor blocks is bounded by $\lceil \log_2 n_p \rceil$ using a refinement factor of 2.

Table 2 illustrates this enumeration strategy using a 16×8 initial fine-grid task fine size. This refinement procedure is used to limit the number of potential redistributions, thereby making the global search feasible within the setup phase.

4.2. Redistribution search. The recursive enumeration of coarse processor grids generates a search space of possible redistributions. To find an optimal redistribution, a state in the search space is given by the size of the processor grid and the size of the coarse-grid associated with a distributed solver. The search space can be viewed as a directed graph—an example is given in Figure 7. The initial state is given by the top-level distributed solver and the goal state is the state with a 1×1 processor grid (in the case of a 2D problem). State transitions have varying costs; the goal is an inexpensive path from the initial state to the goal state.



(a) Processor grid refinement by a factor of 2.



(b) All possible redistributions for 8 processors.

FIG. 6. (a) *Processor grid refinement by a factor of* 2. *The dimension chosen for refinement in each step depends on the agglomerated local problem size.* (b) *A tree of all possible redistributions with an initial processor grid containing eight processors. Nodes in the tree represent the number of processors in the redistribution. Redistributions are enumerated on each level of the tree using refinement by dimension recursively.*

TABLE 2

Example redistribution enumeration using a fine-grid problem of 9088×568 *dof with a* 16×8 *processor grid. The global coarse-grid considered for agglomeration contains* 1136×71 *dof. After step* 4 *refinement in the first dimension is infeasible.*

Step	Redistribution	Agglomerated local problem
1	1×1	1136×71
2	2×1	568×71
3	4×1	284×71
4	8×1	142×71
5	16×1	71×71
6	16×2	71×36
7	16×4	71×18

To search the state space, a path cost function is defined as

$$f(s) = g(s) + h(s), \tag{16}$$

where s is a vertex or state in the graph in Figure 7, g is the cost to reach s from the initial state, and h is an estimate of the cost to reach the goal state from state s. That is, $g(s)$ represents the time predicted by the performance model for a solve phase executed down to coarse-grid state s. For the heuristic function $h(s)$, a weighted combination of the coarse-grid problem and the processor grid size is used to predict the cost. If the performance model is an accurate predictive model, an inexpensive path from the initial state to the goal state will identify a redistributed solver with an efficient run time. With the path cost function defined, the space of redistributions is searched for an optimal path. This is performed in the MG setup phase to dictate agglomeration when a coarsening limit is reached. Using a brute force approach by

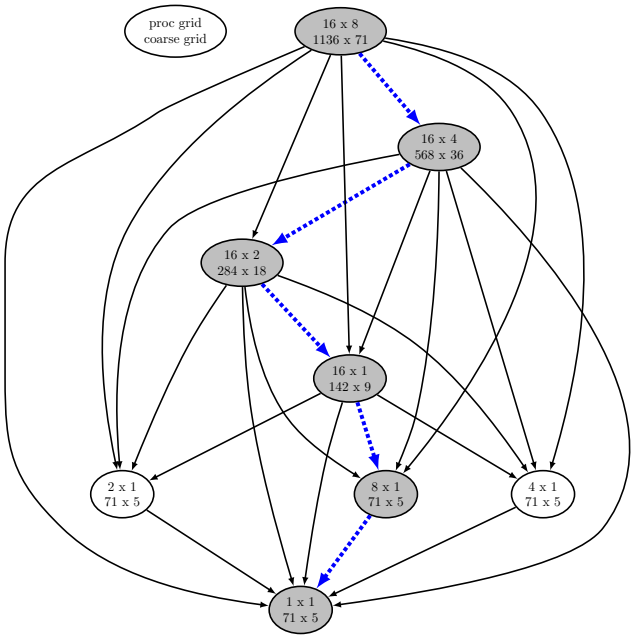


FIG. 7. *Example redistribution search space: The fine-grid global problem is* 9088×568 *dof with a* 16×8 *processor grid, yielding a fine-grid local problem of* 568×71. *The optimal path through this redistribution space is highlighted.*

searching every path for the best redistribution strategy incurs an $\mathcal{O}(n_p)$ cost, since the redistribution search space is constructed recursively. To show this, we let $d = \lceil \log_2(n_p) \rceil$ and consider building a tree of possible redistributions (see Figure 6(b)). We let a node in the tree represent the total number of processors for a given processor grid. Building this tree recursively using the above processor grid enumeration results in the equation for the number of possible redistributions

$$R(d) = \sum_{k=0}^{d-1} R(k) + 1, \tag{17}$$

as the immediate children of a node include the powers of two up to but not including n_p. Using induction, we wish to show $R(d) = 2^d - 1$. For the base case, we have $R(0) = 0 = 2^0 - 1$. Assuming $R(d) = 2^d - 1$, we find

$$\begin{aligned} R(d+1) &= \sum_{k=0}^{d} R(k) + 1 \\ &= (R(d) + 1) + \sum_{k=0}^{d-1} R(k) + 1 \\ &= \big((2^d - 1) + 1\big) + \big(2^d - 1\big) \\ &= 2^d + 2^d - 1 \\ &= 2^{d+1} - 1; \end{aligned}$$

this leads to $R(d) = 2^d - 1 = 2^{\log_2(n_p)} - 1 = n_p - 1$.

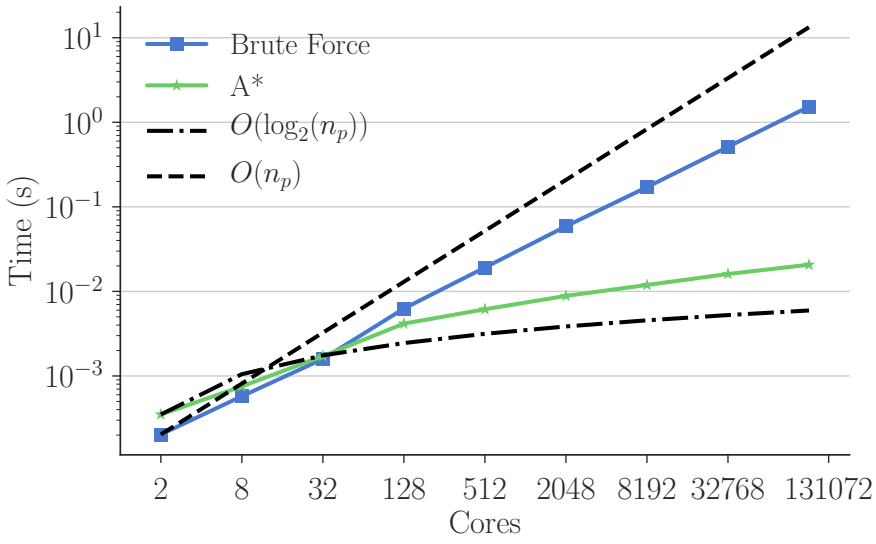


FIG. 8. *Performance of redistribution search algorithms weak scaling with local problem size* 568×71. n_p *denotes the number of cores.*

To address the $\mathcal{O}(n_p)$ cost, the A* algorithm [20] is used to determine the optimal path. While the worst case complexity for A* is $\mathcal{O}(n_p)$ for this search, the cost with an optimal heuristic with evaluation cost $\mathcal{O}(1)$ is the length of the solution path. The length of this path in the redistribution search is $\mathcal{O}(\log_2(n_p))$. Figure 8 shows the A* heuristic is effective in avoiding the brute force cost but does not reach the optimal cost.

5. Experimental results. To explore the performance of the proposed redistribution algorithm a standard five-point finite volume discretization of the diffusion equation is used. Since square problems may lead to a natural or predictable redistribution path, rectangular local problems are used in order to fully stress the redistribution algorithm in a weak scaling study. In particular, the ratio of processors in the processor grid is fixed at $2 : 1$, and the ratio of unknowns in the fine-grid local problem is fixed at $8 : 1$. Other processor grid ratios lead to similar findings.

Note that for an isotropic diffusion problem this grid stretching results in anisotropy in the discrete problem, causing the convergence of the solver to deteriorate when using pointwise smoothing. Consequently, a diffusion problem with compensating *anisotropy* is defined in order to focus on the parallel scalability of the coarse-grid redistribution algorithm (rather than the well-established convergence rate of the multigrid algorithm itself). Specifically, the following diffusion problem is used in these numerical experiments:

$$-\nabla \cdot (D \nabla u) = f(x, y) \quad \text{in } \Omega = (0, 1) \times (0, 1)\,, \tag{18}$$

$$u = 0 \quad \text{on } \partial\Omega\,, \tag{19}$$

where the diffusion tensor is $D = \text{diag}[\frac{1}{r}, r]$ and $r \approx 16$. This compensating anisotropy results in optimal convergence of multigrid V-cycles with pointwise smoothing.

5.1. Scaling studies. Both the weak and strong scalability of multigrid V-cycles that use the proposed redistribution algorithm are important for applications on

exascale systems. Here, the discretization of the diffusion problem given above is used for both weak and strong scaling studies and the algorithmic components of the BoxMG multigrid library (e.g., interpolation, restriction) are used in the redistributed multigrid V-cycles.

Since the cost of coarsening (with redistribution) and the cost of the coarse-grid problem are dominated by parallel communication, network speed and machine topology play an important role in timings. To explore this dependency, two different petascale systems are considered in the scaling tests:

Mira.[3] An IBM Blue Gene/Q system at Argonne National Laboratory. Mira uses an IBM network which comprises a 5D torus with 49,152 compute nodes using PowerPC A2 processors. Each compute node and Mira has a shared cache size of 32MB.

Blue Waters.[4] A Cray XE system at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana–Champaign. Blue Waters employs a 3D torus using a Cray Gemini interconnect and has 22,640 XE compute nodes each with two AMD Interlagos processors. Each XE compute node has a total cache size of 32MB, with a 16MB L3 cache for each socket.

In each case, the machine parameters used in the performance model of section 3 are determined using the b_{eff} benchmark [29].

Weak scalability. In this section, a weak scaling study is conducted to highlight the scalability of the redistribution algorithm at large core counts. For the numerical experiments below, 10 V(2,1)-cycles are executed using two different local problem sizes: 568×71 =40,328 and 288×36 =10,368. We observed an average geometric convergence factor of 0.1 for these runs with the largest error norm being approximately 10^{-10}. The memory footprint for these two local problems was approximately 7MB per core for the 40k local problem size, and 1.8 MB per core for the 10k local problem size. For both machines, the per-core cache size when using 16 cores on each node is 2MB.

Figures 9 and 10 show the run times on Mira of various computational kernels in the multigrid solve phase. The "solve" line shows the time of the entire solve phase and includes the other timings. The "redistribution" line shows communication needed to redistribute coarse problems. This communication is low in comparison to the cost of relaxation. Overall, the algorithm exhibits high parallel scalability in the solve time for both local problem sizes on Mira.

With different network capacities, *redundant* redistribution (see section 2.2) may not yield the lowest communication costs. Indeed, the results in Figure 11 for the Blue Waters system show that while redundant redistribution of the data at course levels is inexpensive, the network contention introduced by redundant cycling contributes to an increase in communication at high core counts. This suggests that triggering redundancy on a per-level basis could lead to reduced costs. In contrast, nonredundant redistribution (in Figure 11) exhibits high scalability—thus, it is used for the following runs on Blue Waters.

Figures 12 and 13 show run times on Blue Waters for the multigrid solve phase, highlighting that the proposed algorithm achieves good parallel scalability here as well. However, comparing runs on the two machines, it is apparent that the weak scalability is superior on Mira. This is in part attributed to the network capabilities

[3] https://www.alcf.anl.gov/mira.

[4] https://bluewaters.ncsa.illinois.edu/blue-waters.

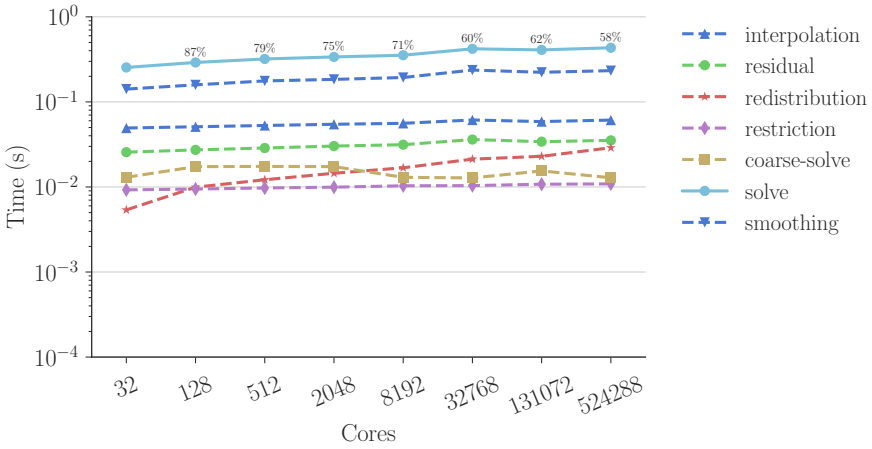


FIG. 9. *Weak scaling on Mira with local problem size* 568×71. *The parallel efficiency (%) relative to* 32 *cores for the overall solve is shown for each data point.*

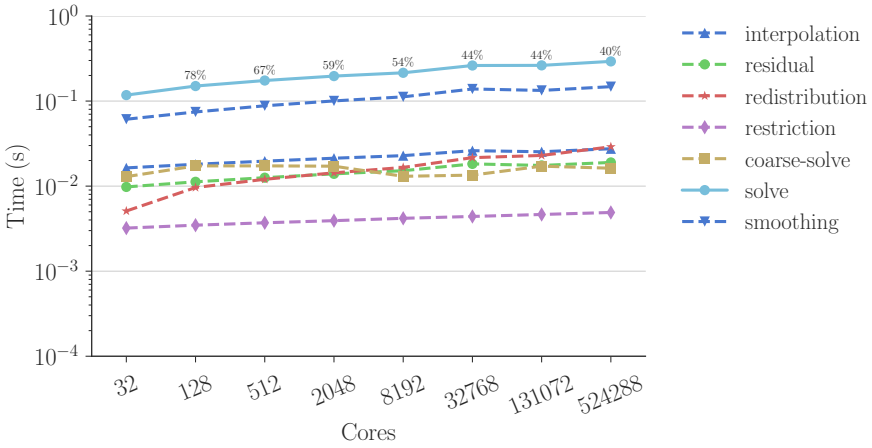


FIG. 10. *Weak scaling on Mira with local problem size* 288×36. *The parallel efficiency (%) relative to* 32 *cores for the overall solve is shown for each data point.*

and scheduling differences of the two machines, noting that communication is included in these measurements (see Figure 14). Jobs on Mira receive a dedicated, full torus partition of the machine, which often reduces contention resulting from neighboring jobs on the machine, since partitions receive a full torus network. Moreover, the lower dimensional 3D torus on Blue Waters also contributes to an increase in contention within the running job.

Nevertheless, on Blue Waters communication cost for redistribution remains relatively low in comparison to smoothing, which remains the dominant kernel in the overall solve. Figure 14 decomposes the cost of smoothing into communication and computation. This decomposition indicates that the communication cost of the halo exchange is the primary contributor to the reduced scalability.

In contrast to the network advantages of Mira, lower floating point performance is observed for the key computational kernels on Mira than on Blue Waters. This

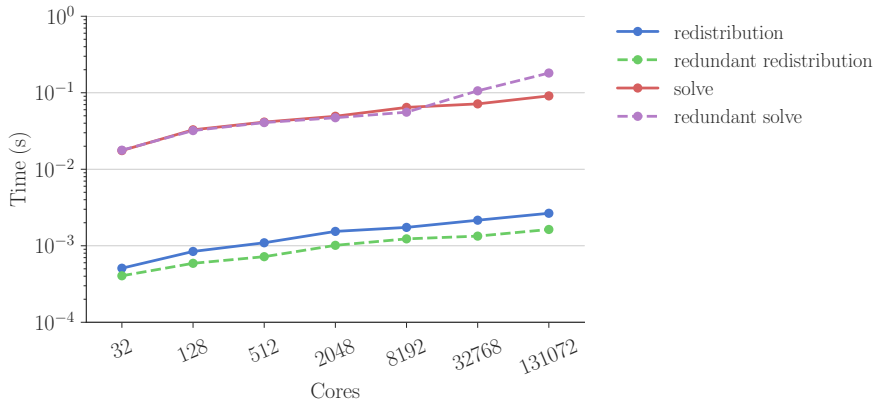


FIG. 11. *Weak scaling on Blue Waters with local problem size* 288×36 *shows the significant improvement of nonredundant redistribution compared to redundant redistribution beyond* 8192 *cores.*

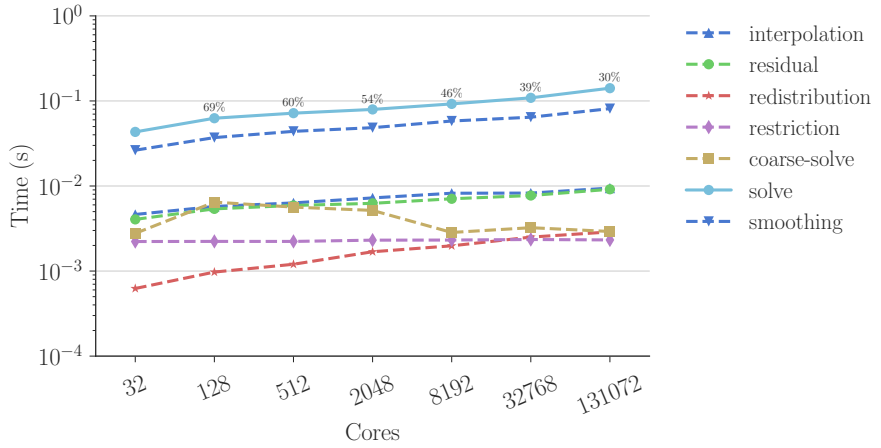


FIG. 12. *Weak scaling on Blue Waters with local problem size* 568×71. *The parallel efficiency (%) relative to* 32 *cores for the overall solve is shown for each data point.*

difference, combined with an extra core dedicated to operating system functions, yields more predictable performance and superior scaling behavior on Mira. This is also observed for a variety of applications [22]: loss in performance on the Cray XE6 in comparison to BG/Q systems as the core count increases. However, with the data locality and fixed communication patterns of the algorithm, the network performance on Blue Waters is good enough to let its superior floating performance yield the best time to solution, solving approximately 2.5 times faster at 131K cores.

Strong scalability. Turning to strong scalability, an isotropic model diffusion problem is set up on a 3200×3200 grid and then executed on a sequence of square processor grids. The number of cores in each coordinate direction is successively doubled from 8 to 128, to create a range of core counts from 64 to 16,384, with a corresponding local problem size ranging from 160,000 to 625 dof. The strong scaling performance for Blue Waters is shown in Figure 15. Here, the strong scaling limit is reached at 1024 cores which is equivalently 10K dof per core. The parallel efficiency

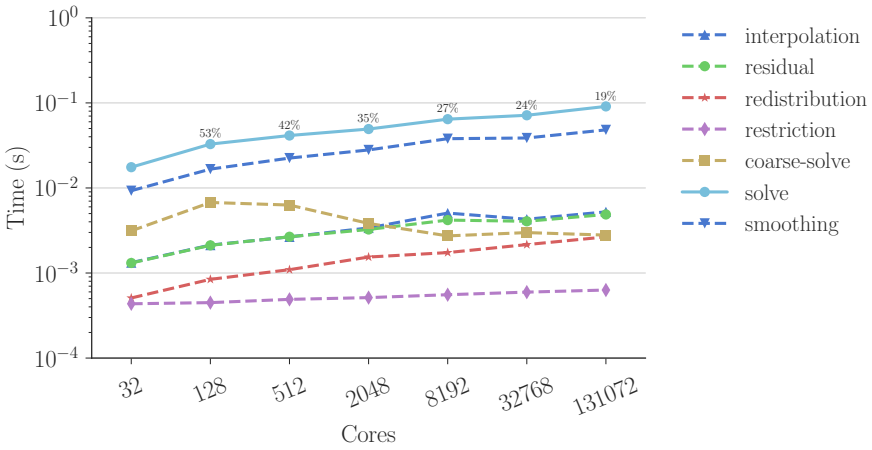


FIG. 13. *Weak scaling on Blue Waters with local problem size* 288×36. *The parallel efficiency (%) relative to* 32 *cores for the overall solve is shown for each data point.*

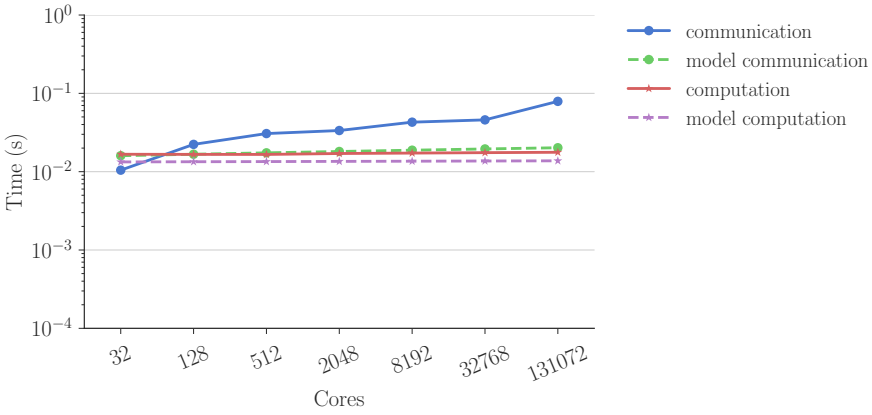


FIG. 14. *Weak scaling of smoothing routine on Blue Waters with local problem size* 568×71.

relative to 64 cores drops from 125% to 53% from 256 cores to 1024 cores and then to 13% at 4096 cores. This scaling limit is consistent with other performance assessments of multilevel solvers in application codes (e.g., [19, 32]). Superlinear speedup is also observed with a parallel efficiency of 125% as the fine-grid problem is able to fit in cache at 256 cores.

The computational cost of the solve is again dominated by smoothing and scaling is limited by communication, as supported in Figure 16 by the performance model of section 3. The steady reduction in computation cost due to a decreasing local problem size is not reflected in the near constant communication cost. The communication cost is expected to be reduced at a slower rate than computation since the size of the halo region is proportional to the surface area of the local problem. The performance model is reasonably accurate, although the slowly increasing underestimation of the communication cost suggests that further investigation of more advanced topology-aware performance models [18] may be worthwhile. In addition, increased network utilization at higher core counts may also contribute to the overestimation. Often a hybrid Gauss–Seidel Jacobi sweep is used to limit communication

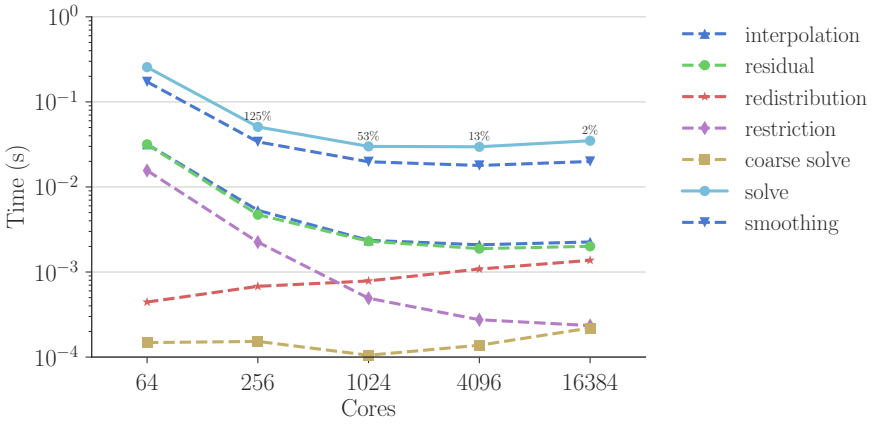


FIG. 15. *Strong scaling on Blue Waters with problem size* 3200×3200. *The parallel efficiency (%) relative to* 64 *cores for the overall solve is shown for each data point.*

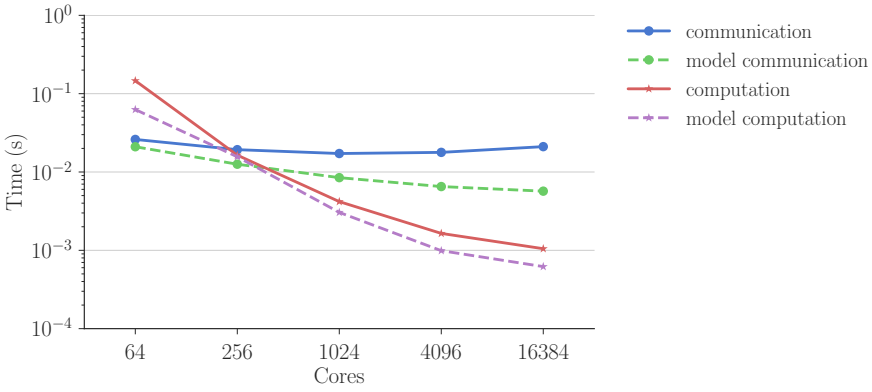


FIG. 16. *Strong scaling of smoothing routine on Blue Waters with problem size* 3200×3200.

to once per relaxation sweep [3]. This improves the cycle's strong scaling behavior but may slow convergence. Here, the focus is on redistribution and the impact of communication is highlighted with a strict implementation of four-color Gauss–Seidel, which results in four halo exchanges per relaxation sweep. The granularity of the component timers is such that the residual and interpolation operations each include a single halo exchange. This leads to strong scaling curves for these operations that are very similar to smoothing but much faster in absolute terms.

In contrast, the restriction has no communication and would exhibit perfect strong scaling in the absence of redistribution. With redistribution fewer cores are used at coarser levels, and strong scaling begins to degrade at 2.5K dof per core, although speedup is still realized even at 625 dof per core. This observation highlights the complexity of trade-offs in multilevel algorithms on modern systems and the important role of performance models in design and run time optimization.

5.2. Extension to three dimensions. This approach was extended naturally to three dimensions. Following the 2D results, the ratio of processors in the processor grid was fixed at $2 : 2 : 1$. We considered two local problem sizes, one with

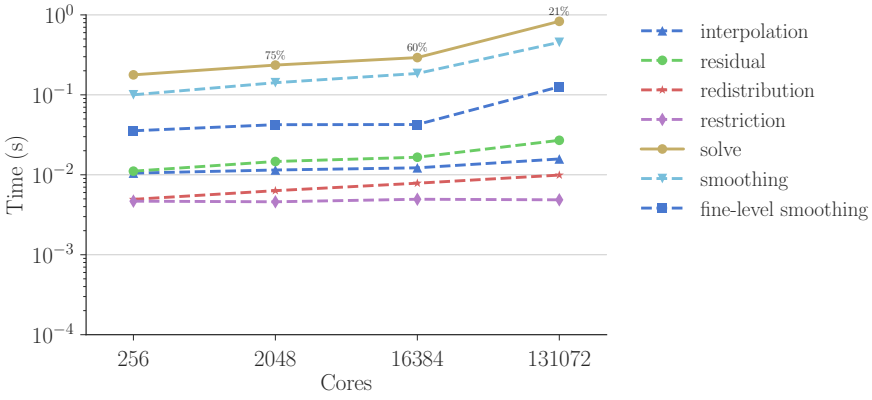


FIG. 17. *Weak scaling on Blue Waters with local problem size* $52 \times 28 \times 28$. *The parallel efficiency (%) relative to* 256 *cores for the overall solve is shown for each data point.*

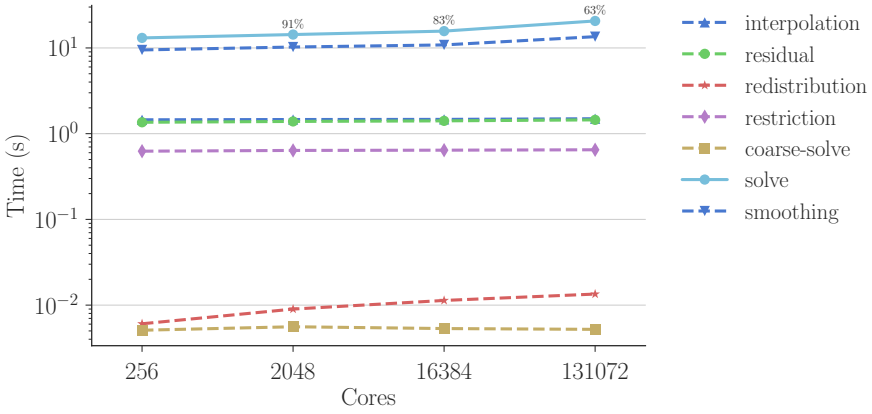


FIG. 18. *Weak scaling on Blue Waters with local problem size* $260 \times 140 \times 140$. *The parallel efficiency (%) relative to* 256 *cores for the overall solve is shown for each data point.*

approximately 40k unknowns per core and one with approximately 5M unknowns per core. The memory footprint was approximately 14MB per core for the 40k local problem size and 1.7GB per core for the 5M local problem size. Figure 17 shows weak scaling on Blue Waters. These results are similar to the 2D results as they have similar computation and communication requirements. The increased communication involved in a 3D halo exchange may account for the decreased parallel efficiency relative to the 2D results. In two dimensions, we also customized the rank ordering on Blue Waters to match the topology of the machine. This resulted in improved performance, especially at higher core counts.

Figure 18 shows weak scaling with a much larger local problem size than the previous weak scaling results. The previous local problem sizes were chosen to be near the strong scaling limit where communication costs dominate and weak scalability is challenged. The local problem size chosen in Figure 18, however, better saturates the memory on each node. Since computation dominates in this case, the weak scaling behavior is superior. The global problem size for this case reaches 6.5×10^{11} degrees of

TABLE 3
Example redistribution paths using a 64×32 *processor grid and* 568×71 *local problem size.*

Path	Processor grid redistribution
0	$64 \times 32 \rightarrow 1 \times 1$
1	$64 \times 32 \rightarrow 64 \times 16 \rightarrow 64 \times 8 \rightarrow 64 \times 4 \rightarrow 32 \times 2 \rightarrow 16 \times 1 \rightarrow 1 \times 1$
2	$64 \times 32 \rightarrow 64 \times 4 \rightarrow 8 \times 1 \rightarrow 4 \times 1$
3	$64 \times 32 \rightarrow 16 \times 2 \rightarrow 1 \times 1$
4	$64 \times 32 \rightarrow 64 \times 16 \rightarrow 2 \times 1 \rightarrow 1 \times 1$
5	$64 \times 32 \rightarrow 4 \times 1 \rightarrow 1 \times 1$
6	$64 \times 32 \rightarrow 64 \times 16 \rightarrow 4 \times 1 \rightarrow 1 \times 1$
7	$64 \times 32 \rightarrow 64 \times 16 \rightarrow 64 \times 8 \rightarrow 2 \times 1 \rightarrow 1 \times 1$
8	$64 \times 32 \rightarrow 2 \times 1 \rightarrow 1 \times 1$

freedom with an overall solve time of 20 seconds. As a reference, in [15], matrix-free geometric methods are used on fine levels of refinement to solve a problem with 1.1×10^{13} dof in under 13 minutes.

While the focus of the present work is on the scalability limitations introduced by coarse levels, future work could consider features of emerging architectures. In addition, a performance analysis using a target architecture [16, 15] could be used to assess the efficiency of the solver.

5.3. Performance of optimized redistribution. To understand the impact of the redistribution path on solve time, the 568×71 weak scaling problem used in Figure 12 with 2048 processors is executed over a variety of redistribution choices. The selected redistribution paths are listed in Table 3. Path 1 indicates the optimal redistribution path used in Figure 12 and is selected by the algorithm from section 4. In stark contrast to Path 1 is Path 0, which is the original all-to-one redistribution path that is known to scale poorly. The remaining paths highlight the flexibility in the agglomeration sequence.

The bar graph in Figure 19 compares the run times and predicted times of the multigrid solve phase for the various redistribution paths shown in Table 3. The predicted times are computed using the performance model from section 3 and are in good agreement with the actual run times. Comparing the run times of Path 0 and Path 1, this graph shows a 40 times speedup of the new redistribution strategy over the original all-to-one strategy. In addition, Path 1 has the lowest run time of several plausible alternatives to the all-to-one strategy. Thus, Figure 19 demonstrates the utility of using a global search guided by a performance model as a predictive tool for choosing optimal redistribution paths. Moreover, using this strategy to select the redistribution path delivered effective weak parallel scaling out to 500K+ cores on Mira and out to 132K+ cores on Blue Waters, while the original code was essentially unusable beyond 4K cores.

6. Conclusions. Emerging architectures place an increasing importance on data locality and minimizing data movement. In these environments, structured approaches benefit from predictable memory access patterns and avoiding indirect addressing. This motivates the development of methods that exploit local structure to avoid incurring the performance consequences of full algebraic generality. A robust structured single-block multigrid implementation that scales well on modern architectures is an important step toward this goal.

In this paper, a new optimized recursive agglomeration algorithm for redistributing coarse-grid work in structured multilevel solvers is introduced. This algorithm

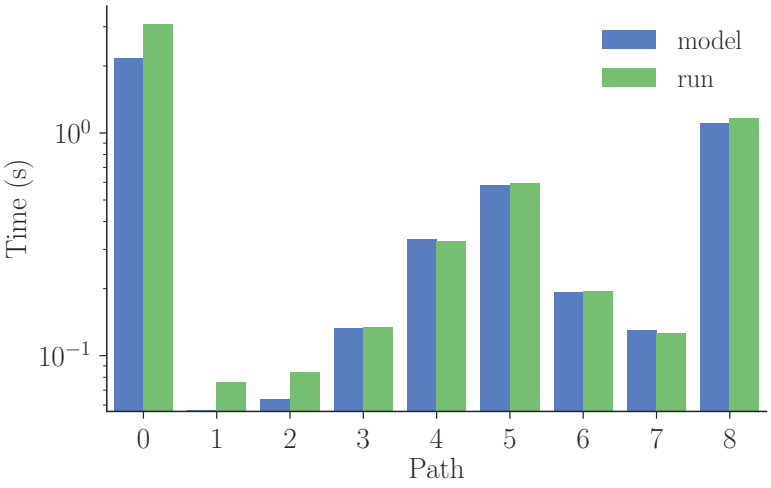


FIG. 19. *Model and run times on Blue Waters of various redistribution paths using a* 64×32 *processor grid and* 568×71 *local problem size.*

combines a predictive performance model with a structure exploiting recursive enumeration of coarse processor grids to enable a global search for the optimal agglomeration strategy. This approach significantly improves the weak parallel scalability of robust, structured multigrid solvers such as BoxMG. In this study using BoxMG operators, this new algorithm delivers very good weak scaling up to 524,288 cores on an IBM BG/Q (Mira) and reasonable weak scaling up to 131,072 cores on Cray XE (Blue Waters). The speedup over the previous all-to-one agglomeration approach is significant even at modest core counts, 40 times speedup on just 2048 cores and 144 times speedup on 8192 cores when comparing the data from Figures 1 and 12. At larger core counts, the all-to-one agglomeration approach became infeasible as the increased memory requirements for the coarse-grid problem exceeded available memory.

In addition, the strong scaling of multigrid solves using this redistribution algorithm is demonstrated on Blue Waters. Overall, the scaling limit is observed to be approximately 10K dof per core, which is similar to results obtained in other studies and is dominated by the communication cost of the smoother. Future work on additive variants of multigrid and improvements to the performance model to more accurately capture deep memory hierarchies are needed to reduce this limit.

To focus on coarse-grid redistribution, only pointwise Gauss–Seidel relaxation is considered in this paper. In the future, smoothers that may enhance performance, such as hybrid or polynomial smoothers, or enhance robustness, such as line and plane smoothers [2], may be considered and their impact on optimal coarse-grid redistribution explored.

REFERENCES

[1] R. E. ALCOUFFE, A. BRANDT, J. E. DENDY, AND J. W. PAINTER, *The multi-grid method for the diffusion equation with strongly discontinuous coefficients*, SIAM J. Sci. Statist. Comput., 2 (1981), pp. 430–454.

[2] T. M. AUSTIN, M. BERNDT, AND J. D. MOULTON, *A Memory Efficient Parallel Tridiagonal Solver*, Tech. Report LA-UR 03-4149, Mathematical Modeling and Analysis Group, Los Alamos National Laboratory, Los Alamos, NM, 2004.

[3] A. H. Baker, R. D. Falgout, H. Gahvari, T. Gamblin, W. Gropp, T. V. Kolev, K. E. Jordan, M. Schulz, and U. M. Yang, *Preparing Algebraic Multigrid for Exascale*, Tech. Report LLNL-TR-533076, Lawrence Livermore National Laboratory, Livermore, CA, 2012.

[4] A. Bar-Noy and S. Kipnis, *Designing broadcasting algorithms in the postal model for message-passing systems*, in Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, New York, 1992, pp. 13–22, https://doi.org/10.1145/140901.140903.

[5] A. Bienz, R. D. Falgout, W. Gropp, L. N. Olson, and J. B. Schroder, *Reducing parallel communication in algebraic multigrid through sparsification*, SIAM J. Sci. Comput., 38 (2016), pp. S332–S357, https://doi.org/10.1137/15M1026341.

[6] A. Bienz, W. D. Gropp, and L. Olson, *Topology-aware performance modeling of parallel SpMvs*, in Proceedings of the 17th SIAM Conference on Parallel Proceesing for Scientific Computing, SIAM, Philadelphia, 2017, http://meetings.siam.org/sess/dsp_talk.cfm?p=75934.

[7] A. Brandt, S. F. McCormick, and J. W. Ruge, *Algebraic multigrid (AMG) for sparse matrix equations*, in Sparsity and Its Applications, Cambridge University Press, Cambridge, UK, 1984.

[8] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial, 2nd.ed.* SIAM, Philadelphia, 2000.

[9] G. L. Delzanno, L. Chacón, J. M. Finn, Y. Chung, and G. Lapenta, *An optimal robust equidistribution method for two-dimensional grid adaptation based on Monge-Kantorovich optimization*, J. Comput. Phys., 227 (2008), pp. 9841–9864, http://www.sciencedirect.com/science/article/pii/S0021999108004105.

[10] J. E. Dendy, *Black box multigrid*, J. Comput. Phys., 48 (1982), pp. 366–386.

[11] J. E. Dendy, *Black box multigrid for nonsymmetric problems*, Appl. Math. Comput., 13 (1983), pp. 261–283.

[12] J. E. Dendy and J. D. Moulton, *Black box multigrid with coarsening by a factor of three*, Numer. Linear Algebra Appl., 17 (2010), pp. 577–598, https://doi.org/10.1002/nla.705.

[13] M. Emans, *Coarse-grid treatment in parallel amg for coupled systems in CFD applications*, J. Comput. Sci., 2 (2011), pp. 365– 376.

[14] H. Gahvari, W. Gropp, K. E. Jordan, M. Schulz, and U. M. Yang, *Systematic reduction of data movement in algebraic multigrid solvers*, in Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, IPDPSW '13, Washington, DC, 2013, pp. 1675–1682.

[15] B. Gmeiner, M. Huber, L. John, U. Rüde, and B. Wohlmuth, *A quantitative performance study for stokes solvers at the extreme scale*, J. Comput. Sci., 17 (2016), pp. 509–521.

[16] B. Gmeiner, U. Rüde, H. Stengel, C. Waluga, and B. Wohlmuth, *Towards textbook efficiency for parallel multigrid*, Numer. Math. Theory Methods & Appl., 8 (2015), pp. 22–46.

[17] W. Gropp, *Parallel computing and domain decomposition*, in Domain Decomposition Methods for Partial Differential Equations, SIAM, Philadelphia, 1992, pp. 349 – 361.

[18] W. Gropp, L. N. Olson, and P. Samfass, *Modeling MPI communication performance on SMP nodes: Is it time to retire the ping pong test*, in Proceedings of the 23rd European MPI Users' Group Meeting, New York, ACM, 2016, pp. 41–50, https://doi.org/10.1145/2966884.2966919.

[19] G. E. Hammond, P. C. Lichtner, C. Lu, and R. T. Mills, *PFLOTRAN: Reactive flow and transport code for use on laptops to leadership-class supercomputers*, in Groundwater Reactive Transport Models, F. Zhang, G. Yeh, and J. C. Parker, eds., Bentham Science Publishers, Sharjah, UAE, 2012, pp. 141–159, https://doi.org/10.2174/978160805306311201O1.

[20] P. E. Hart, N. J. Nilsson, and B. Raphael, *A formal basis for the heuristic determination of minimum cost paths*, IEEE Trans. Systems Sci. Cybern., 4 (1968), pp. 100–107.

[21] W. Joubert and J. Cullum, *Scalable algebraic multigrid on* 3500 *processors*, Electron. Trans. Numer. Anal., 23 (2006), pp. 105–128.

[22] D. J. Kerbyson, K. J. Barker, A. Vishnu, and A. Hoisie, *Comparing the performance of Blue Gene/Q with leading Cray XE6 and InfiniBand systems*, in Proceedings of the International Conference on Parallel and Distributed Systems, 2012, pp. 556–563, https://doi.org/10.1109/ICPADS.2012.81.

[23] S. P. MacLachlan and J. D. Moulton, *Multilevel upscaling through variational coarsening*, Water Resour. Res., 42 (2006), W02418, https://doi.org/10.1029/2005WR003940.

[24] S. P. MacLachlan, J. M. Tang, and C. Vuik, *Fast and robust solvers for pressure-correction in bubbly flow problems*, J. Comput. Phys., 227 (2008), pp. 9742–9761.

[25] D. A. May, P. Sanan, K. Rupp, M. G. Knepley, and B. F. Smith, *Extreme-scale multigrid components within PETSc*, in Proceedings of the Platform for Advanced Scientific Computing Conference, 2016, pp. 5:1–5:12.

[26] D. Moulton, L. N. Olson, and A. Reisner, *Cedar Framework*, Version 0.1, 2017, https://github.com/cedar-framework/cedar.
[27] J. D. Moulton, J. E. Dendy, and J. M. Hyman, *The black box multigrid numerical homogenization algorithm*, J. Comput. Phys., 141 (1998), pp. 1–29.
[28] K. Nakajima, *Optimization of serial and parallel communications for parallel geometric multigrid method*, in Proceedings of the 20th IEEE International Conference on Parallel and Distributed Systems, 2014, pp. 25–32.
[29] R. Rabenseifner, *Effective Bandwidth (b_{eff}) Benchmark*, www.hlrs.de/mpi/b_eff.
[30] S. Reiter, A. Vogel, I. Heppner, M. Rupp, and G. Wittum, *A massively parallel geometric multigrid solver on hierarchically distributed grids*, Comput. Vis. Sci., 16 (2013), pp. 151–164.
[31] J. Rudi, A. C. I. Malossi, T. Isaac, G. Stadler, M. Gurnis, P. W. J. Staar, Y. Ineichen, C. Bekas, A. Curioni, and O. Ghattas, *An extreme-scale implicit solver for complex PDEs: Highly heterogeneous flow in earth's mantle*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2015.
[32] H. Sundar, G. Biros, C. Burstedde, J. Rudi, O. Ghattas, and G. Stadler, *Parallel geometric-algebraic multigrid on unstructured forests of octrees*, in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE Computer Society Press, 2012, p. 43.
[33] R. Thakur, R. Rabenseifner, and W. Gropp, *Optimization of collective communication operations in MPICH*, Int. J. of High Performance Comput. Appl., 19 (2005), pp. 49–66.
[34] U. Trottenberg and A. Schuller, *Multigrid*, Academic Press, Inc., Orlando, FL, 2001.
[35] D. E. Womble and B. C. Young, *A Model and implementation of multigrid for massively parallel computers*, Int. J. High Speed Comput., 2 (1990), pp. 239–255.
[36] P. D. Zeeuw, *Matrix-dependent prolongations and restrictions in a blackbox multigrid solver*, J. Comput. Appl. Math., 33 (1990), pp. 1–27, https://doi.org/10.1016/0377-0427(90)90252-U.