Parallel Computing 100 (2020) 102705

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Scalable line and plane relaxation in a parallel structured multigrid solver ☆

Andrew Reisner [b,*], Markus Berndt [c], J. David Moulton [a], Luke N. Olson [d]

[a] Applied Mathematics and Plasma Physics, Los Alamos National Laboratory, United States of America
[b] Applied Computer Science, Los Alamos National Laboratory, United States of America
[c] Computational Physics and Methods, Los Alamos National Laboratory, United States of America
[d] Department of Computer Science, University of Illinois at Urbana–Champaign, United States of America

ARTICLE INFO

Keywords:
Parallel
Line relaxation
Plane relaxation communication
Multigrid
Structured

ABSTRACT

The efficient solution of sparse, linear systems that arise through the discretization of partial differential equations remains a key challenge for a range of high performance scientific simulations. One approach for reducing data movement and improving performance is by exposing and exploiting structure in a problem through the use of robust structured multilevel solvers. By choosing coarsening that preserves the structure of the problem, these methods maintain efficient structured computation and communication throughout the multigrid hierarchy. However, when coarsening is not permitted to be dependent on the operator, anisotropy must be addressed by the smoother — producing error compatible for coarse-grid correction with structured coarsening. In this paper, the components required in a scalable parallel structured solver are described with a focus on memory and communication efficiency of robust smoothers. While the implementation of communication and memory reduction techniques in smoothers integrated in a complete 3D solver present a significant engineering challenge, a novel approach is proposed that addresses these challenges systematically through a change to the solver's execution model. Enabled by user-level threading paired with a set of data and communication abstractions, this approach permits seamless aggregation of communication in plane smoothers — directly reusing code for a 2D distributed multilevel cycle. Results show an effective reduction in communication costs for coarse-grid problems, and result in a speedup of 8.7× in smoothing routines shown in Fig. 12 using this approach. This produces a significant improvement to strong scalability while maintaining favorable weak scaling behavior. Finally, a parallel scaling study using a series of refined meshes is included that demonstrates the effectiveness of this approach in an application of interest.

1. Introduction

The efficient solution of sparse, linear systems that arise through the discretization of partial differential equations remains a key challenge for a range of high performance scientific simulations. While algebraic multigrid solvers are robust for a variety of problems, their parallel performance continues to be challenged by modern HPC architectures [1–3]. By exposing and exploiting structure in a problem, additional performance improvements and parallel scalability may be achieved [4]. For example, robust variational multigrid methods on logically structured grids, such as Black Box Multigrid (BoxMG) [5,6], take advantage of structure using direct memory addressing and stencil-based operators throughout the multigrid hierarchy. This avoids growth in operator complexity and communication costs at coarse levels typical in AMG [3] by maintaining fixed and predictable communication patterns of a structured computation.

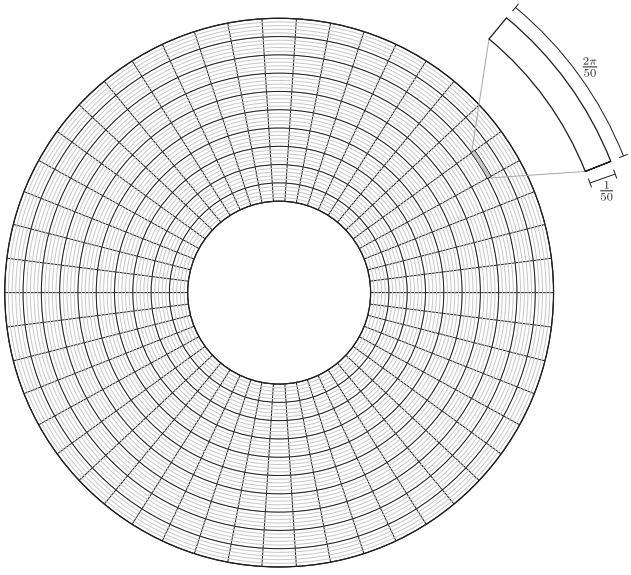
Relaxation methods, such as Gauss–Seidel or weighted Jacobi, play an important role in the success of a multilevel methods (see Section 2 for a detailed description). Yet, in many scenarios such as stretched meshes or anisotropic diffusion operators, more robust relaxation techniques may be necessary. For example, considering discretizations on logically structured grids that lead to nearest neighbor stencils, relaxation along entire lines or planes of the mesh and in alternating directions often improve convergence significantly, and at a substantial cost if parallel communication is not addressed properly, as we show in this paper.

As an example, consider the cylindrical domain in Fig. 1a with a curvilinear discretization of a Poisson problem. Here, the mesh mapping introduces an anisotropy into the resulting operator. Using standard pointwise weighted Jacobi relaxation in a multigrid scheme (BoxMG) results in stagnation of the iterates as shown in Fig. 1b.

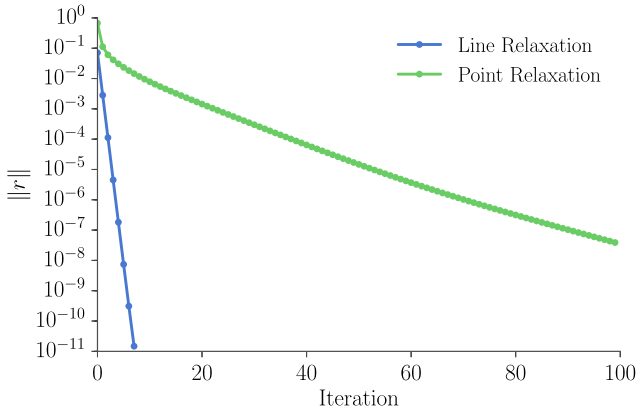
☆ Los Alamos Report LA-UR-20-23447.

* Corresponding author.
E-mail addresses: areisner@lanl.gov (A. Reisner), berndt@lanl.gov (M. Berndt), moulton@lanl.gov (J.D. Moulton), lukeo@illinois.edu (L.N. Olson).

https://doi.org/10.1016/j.parco.2020.102705
Received 28 May 2020; Received in revised form 29 September 2020; Accepted 17 October 2020
Available online 20 October 2020



(a) Cross section of cylindrical mesh



(b) Convergence of point and alternating line smoothers on a curvilinear mesh.

Fig. 1. Impact of anisotropy introduced by mesh mapping on solver convergence using line and point relaxation. This anisotropy leads to poor convergence of BoxMG using point relaxation.

In contrast, using relaxation methods that annihilate the residual on subsets of points (rather than individual points) is more effective. Indeed, for this example, alternating line smoothing (in the radial and in the concentric directions) significantly improves performance, as shown in Fig. 1b.

In a parallel setting, operating on entire lines or planes of a mesh (as is the case in line and plane relaxation) presents a challenge for the parallel efficiency of structured solvers. Unlike point relaxation, these methods involve *global* operations over a subset of dimensions in the problem. Parallel scalability necessitates matching the dimensions of the grid partition to the problem. This implies the global connections will span a non-constant number of processors and increase the communication requirements of the routine. In this paper, we focus on reducing the communication costs in line and plane relaxation methods to improve the parallel performance of structured multilevel solvers.

Distributed-memory parallel tridiagonal solves are heavily used in line and plane relaxation. Efficient tridiagonal solvers, such as the spike algorithm [7–10] target both fine and coarse-grained parallelism to achieve a high-throughput solution to tridiagonal linear systems on both CPU on GPU based architectures. In line and plane relaxation, the solution of many simultaneous tridiagonal systems involving the entire computational grid are needed. In this context, memory efficiency is increasingly important including the need to avoid the storage

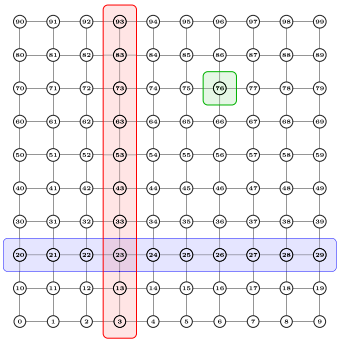


Fig. 2. Updated data in the case of point relaxation (green), y-line relaxation (red), and x-line relaxation (blue).

requirements of local block factorizations. For this reason, we consider the memory efficient tridiagonal solver from [11] although a hybrid approach or memory pooling could be considered to reduce these significant memory costs.

In this paper, the recursive version of the parallel tridiagonal solver proposed in [11] is implemented to improve the parallel scalability of line smoothing in Black Box Multigrid. Parallel results are introduced that confirm the expected communication complexity of this solver. In three dimensions, an automated strategy for aggregating parallel communication in plane smoothing is proposed. This strategy relies on efficient context switching to synchronize communication phases of 2D solvers used in plane smoothing. This approach permits communication within existing 2D solvers to be aggregated when used in a plane smoothing operation. This is especially valuable for parallel 2D solvers with nontrivial communication patterns; for example, a 2D solver that recursively redistributes coarse-grid problems. Parallel results using model problems in addition to a selected application are included to demonstrate the utility of this approach.

The paper is organized as follows. Section 2 introduces smoothers used in structured solvers and motivates the need for block smoothers when structure is preserved on coarse levels. In Section 3, parallel algorithms for block smoothing in two and three dimensions are discussed. Section 4 proposes an automated strategy for reducing communication latency costs in plane smoothing. Performance studies included in Section 5 demonstrate the effectiveness of this approach and Section 6 provides conclusions.

2. Background

2.1. Smoothers for structured solvers

Given an $n \times n$ matrix problem

$$A\mathbf{x} = \mathbf{b}$$

the ith entry if $\mathbf{x}$ is denoted x_i. Given a vector $\mathbf{x}$, a pointwise weighted Jacobi scheme is defined for each $i = 1, \ldots, n$ as

$$x_i^{(\text{new})} \leftarrow x_i + \omega \frac{1}{A_{ii}} \left(\sum_{j,\, A_{ij} \neq 0} b_i - A_{ij} x_j \right),$$

for some weight ω. In matrix form this becomes

$$\mathbf{x}^{(\text{new})} = \mathbf{x} + \omega D^{-1}(\mathbf{b} - A\mathbf{x}), \tag{1}$$

where D is the diagonal of A. The method effectively annihilates the residual at each successive element of the vector $\mathbf{x}$ to form a new iterate. Point-wise iterative methods, such as Gauss–Seidel or weighted Jacobi, operate on single entries of a vector (see Fig. 2) and are effective smoothers for isotropic problems. With anisotropic problems, either coarsening or relaxation must be modified to account for the anisotropy

(see Fig. 1b). Coarsening in a single direction, referred to as semicoarsening, can be effective for problems where the anisotropy is aligned to the grid [12,13]. For general anisotropy, semicoarsening must be combined with block smoothers for robustness [14]. Semicoarsening also has a drawback of increasing the total work in each cycle, since coarsening is less rapid — e.g. coarsening by a factor of 2 is common in 2D, whereas full coarsening reduces the coarse problem size by a factor of 4.

When coarsening is fixed, block smoothers are used to address anisotropy [14–16]. For example, line smoothing operates on matrix entries corresponding to the same logical grid line collectively — see Fig. 1b. That is, x-line relaxation refers to logical grid lines of constant x. Consider the case of a 2D 9-point discretization on a regular $n \times n$ mesh, which has the sparsity pattern given in Fig. 3a in the case of $n = 5$.

Here, the matrix problem can be written in block form, say by groups of points in y-lines, in the following way

$$\begin{bmatrix} D_1 & U_1 & & & & \\ L_2 & D_2 & U_2 & & & \\ & L_3 & D_3 & U_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & L_p & D_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}. \tag{2}$$

In each case D_i, L_i, and U_i are tridiagonal matrices.

In line relaxation, each step is a block solve, which has the form

$$\begin{cases} D_i x_i = b_i - U_i x_{i+1} & \text{if } i = 1 \\ D_i x_i = b_i - L_i x_{i-1} - U_i x_{i+1} & \text{if } 1 < i < n \\ D_i x_i = b_i - L_i x_{i-1} & \text{if } i = n. \end{cases} \tag{3}$$

This type of block smoothing is effective for grid aligned anisotropy by collectively processing strongly connected unknowns. By simply alternating directions in each smoothing step, line relaxation in two dimensions is robust for anisotropy that is not aligned to the grid [14–16].

Similar to line smoothing in two dimensions, plane smoothing can be used in three dimensions as a robust smoother for anisotropic problems. In plane smoothing, rows in the matrix are grouped by logical planes to reach the structure shown in (2). Consider a 3D 27-point discretization on an $n \times n \times n$ mesh. In the case of $n = 4$, the sparsity patterns is given in Fig. 3b. In this case, each block of (2) represents a 2D problem (with corresponding tridiagonal blocks as in the 2D example above). Each D_i, for example, is a structured 2D operator for each plane — plane relaxation then requires successive 2D solves. While a full 2D solve is relatively expensive, in practice it is unnecessary to solve these subproblems exactly. For example, in [14] a single V(1,1) cycle is shown to be sufficient for recovering expected convergence factors.

2.2. Parallel structured decomposition implementation

In a distributed parallel setting, the processor layout adds an additional layer of mappings to the block structures outlined above. In this paper we consider Cartesian processor topologies, where logically structured problems (e.g. Fig. 1a) are decomposed on a processor grid by dividing the points among the processors by dimension.

Stencil-based communication is accomplished using halo regions with a halo width of one. Parallel grid coarsening is used considering standard coarsening by a factor of two in each dimension. When the local problem size becomes too small, recursive coarse-grid redistribution is used to repartition the work on a subset of involved processors as described in [4].

3. Parallel structured smoothing

Efficient parallel smoothers are critical in achieving high performance in structured solvers as they dominate the cost of a solve. In point smoothing, computation and communication is straightforward. The computation consists of stencil-based updates for each grid point and the communication is performed through single-layer halo exchanges. In the case of line and plane smoothing, however, global operations are executed over a subset of dimensions. The increased communication requirements of these operations motivates the need for scalable and efficient methods for processing a sequence of distributed block solves from (3). In line relaxation, these blocks correspond to tridiagonal systems, thus requiring a scalable distributed memory tridiagonal solver. In plane relaxation, efficient processing of a series of distributed two dimensional cycles is needed.

In this paper, Gauss–Seidel block smoothing is considered for line and plane smoothing. To process blocks in parallel, red–black ordering of blocks is used. However the techniques can be extended to any number of structured smoothers.

3.1. Parallel line smoothing

An efficient tridiagonal solver is key to the performance of line smoothing in parallel. In this paper, we consider a series of tridiagonal systems in a distributed memory environment, which forms the basis of the steps outlined in (3). Many parallel algorithms have been developed for the direct solution of tridiagonal linear systems. Many of the early algorithms, such as cyclic reduction [17] and recursive doubling [18] target fine-grained parallelism. Later, a wide class of algorithms based on parallel partitioning of the matrix rows were developed to target coarse-grained parallelism [7,19–22]. In [23], a framework for the unified analysis and development of such algorithms is also proposed. While many of these algorithms focus on a single level of partitioning, recursive partitioning should be used to avoid the linear scaling of computation and communication with respect to the number of processes.

In this paper, the recursive parallel partitioning algorithm proposed in [11] for scalable distributed line relaxation is considered. In this approach, rows of the matrix are partitioned across p processors. For example in the case of $p = 3$ and a single $n \times n$ tridiagonal matrix, the matrix decomposition has the following form on the first processor:

$$\begin{bmatrix} d_0 & u_0 & & & & \\ l_1 & d_1 & u_1 & & & \\ & & \ddots & & & \\ & & l_{n/3-1} & d_{n/3-1} & u_{n/3-1} \end{bmatrix}, \tag{4}$$

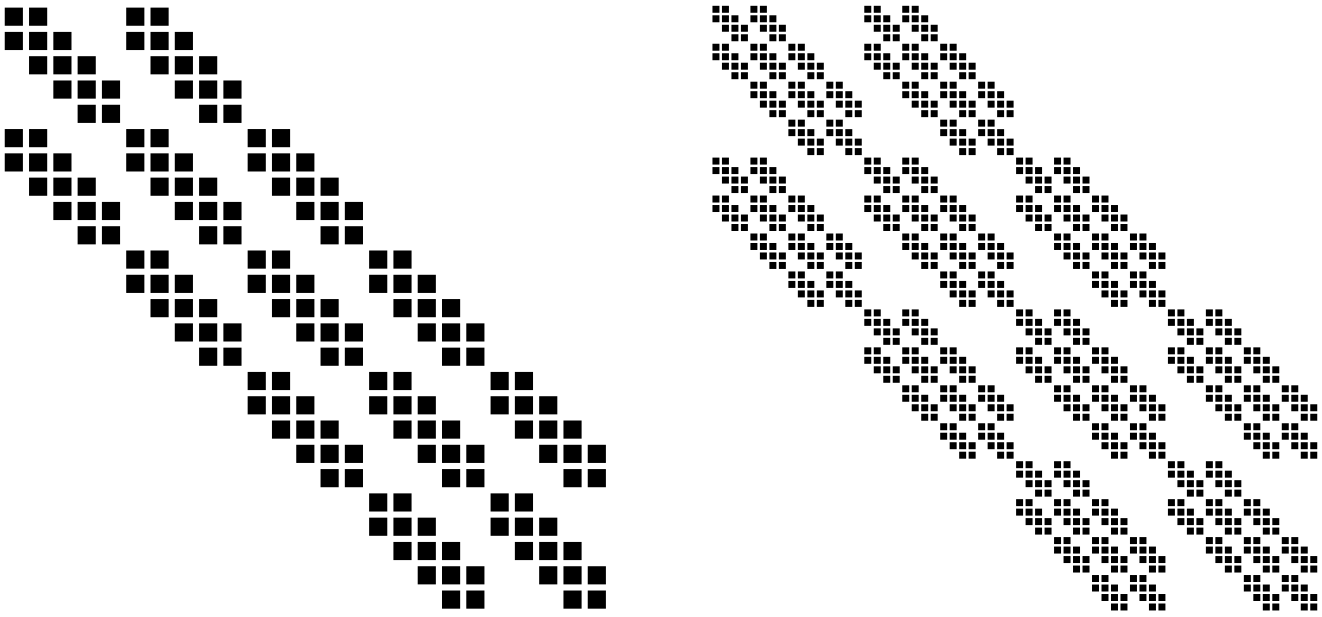
where d_i, l_i, and u_i represent the diagonal and off-diagonal entries. For example, Fig. 4 shows a tridiagonal matrix partitioned across three processors. Rows adjacent to partition boundaries are called interface rows, while rows not connected to a row across a partition boundary are called interior rows. If the unknowns in each partition are reordered with interface unknowns listed last, the tridiagonal system has the following block structure:

$$\begin{bmatrix} A_1 & A_{1,2} & & & & \\ A_{2,1} & A_2 & A_{2,3} & & & \\ & A_{3,2} & A_3 & A_{3,4} & & \\ & & \ddots & \ddots & \ddots & \\ & & & A_{p,p-1} & A_p \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_p \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ \vdots \\ h_p \end{bmatrix}, \tag{5}$$

where vectors z_i and h_i are defined

$$z_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad \text{and} \quad h_i = \begin{bmatrix} f_i \\ g_i \end{bmatrix}, \tag{6}$$

with x_i, f_i corresponding to the interior solution and right-hand-side components respectively and y_i, g_i corresponding to the interface solution and right-hand-side components respectively. Additionally, each



(a) 2D 5×5 grid. (b) 3D $4 \times 4 \times 4$ grid.

Fig. 3. Matrix sparsity patterns for 2D and 3D discretizations.

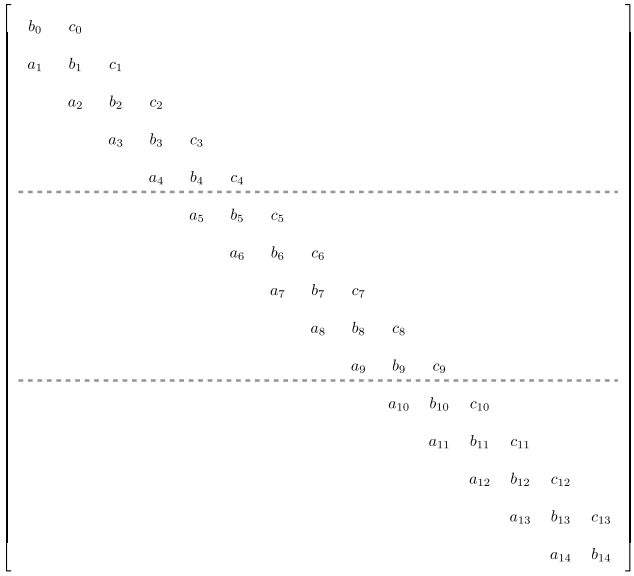


Fig. 4. Rows of a tridiagonal system partitioned across three processors.

$A_{i,i-1}$, A_i, and $A_{i,i+1}$ has the structure

$$A_{i,i-1} = \begin{bmatrix} 0 & 0 \\ 0 & E_{i,i-1} \end{bmatrix} \tag{7}$$

$$A_i = \begin{bmatrix} B_i & E_i \\ F_i & C_i \end{bmatrix} \tag{8}$$

$$A_{i,i+1} = \begin{bmatrix} 0 & 0 \\ 0 & E_{i,i+1} \end{bmatrix} \tag{9}$$

where B_i represents the matrix of interior rows, E_i and F_i represent coupling to and from the interface rows, and C_i represents couplings between interface rows on a given partition. Fig. 5 depicts this structure for the case of the matrix in Fig. 4 when the interface unknowns are ordered last.

The underlying idea in this algorithm is to decouple the interface points from interior points. This is accomplished in three phases.

The first phase: decouple interface system

To form a decoupled reduced system involving only the interface rows, the interior rows are used to eliminate each F_i block. To perform this elimination, two cases are considered. For the lower interface equation which has a nonzero in the last column of F_i, Gaussian elimination on

$$\begin{bmatrix} B_i & E_i \\ \boldsymbol{e_k}^T F_i & \boldsymbol{e_k}^T C_i \end{bmatrix} \tag{10}$$

is used to eliminate this nonzero. Here, $\boldsymbol{e_k}$ is the kth column of the identity matrix with k equal to the number of rows in C_i. For the upper interface equation which has a nonzero in the first column of F_i, Gaussian elimination on

$$\begin{bmatrix} B_i & E_i \\ \boldsymbol{e_1}^T F_i & \boldsymbol{e_1}^T C_i \end{bmatrix} \tag{11}$$

is used to remove this nonzero. During each elimination step, the corresponding row of C_i is modified and fill is introduced from the nonzero columns of E_i. Letting $\widehat{C}_i$ denote the updated C_i matrix and $\widehat{g}_i$ the updated right-hand side, the reduced system produced by eliminating the F_i block for each interface row has the structure:

$$\begin{bmatrix} \widehat{C}_1 & E_{1,2} & & & \\ E_{2,1} & \widehat{C}_2 & E_{2,3} & & \\ & E_{3,2} & \widehat{C}_3 & E_{3,4} & \\ & & \ddots & \ddots & \ddots \\ & & & E_{p,p-1} & \widehat{C}_p \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_p \end{bmatrix} = \begin{bmatrix} \widehat{g}_1 \\ \widehat{g}_2 \\ \widehat{g}_3 \\ \vdots \\ \widehat{g}_p \end{bmatrix} \tag{12}$$

The second phase: solve interface system The reduced system in (12) is a tridiagonal system with $2p - 2$ rows. To solve this system, recursion can be used by partitioning (12) to s processors with $s < p$. This is done by grouping processors from the original partition. Recursion is terminated when $s = 1$. In this case, the reduced system is gathered to one processor. To solve the reduced system any sequential tridiagonal solver can be used, such as the Thomas algorithm. When this phase is complete, the solution y_i is scattered from processors in the merged partition to their corresponding processors in the original partition.

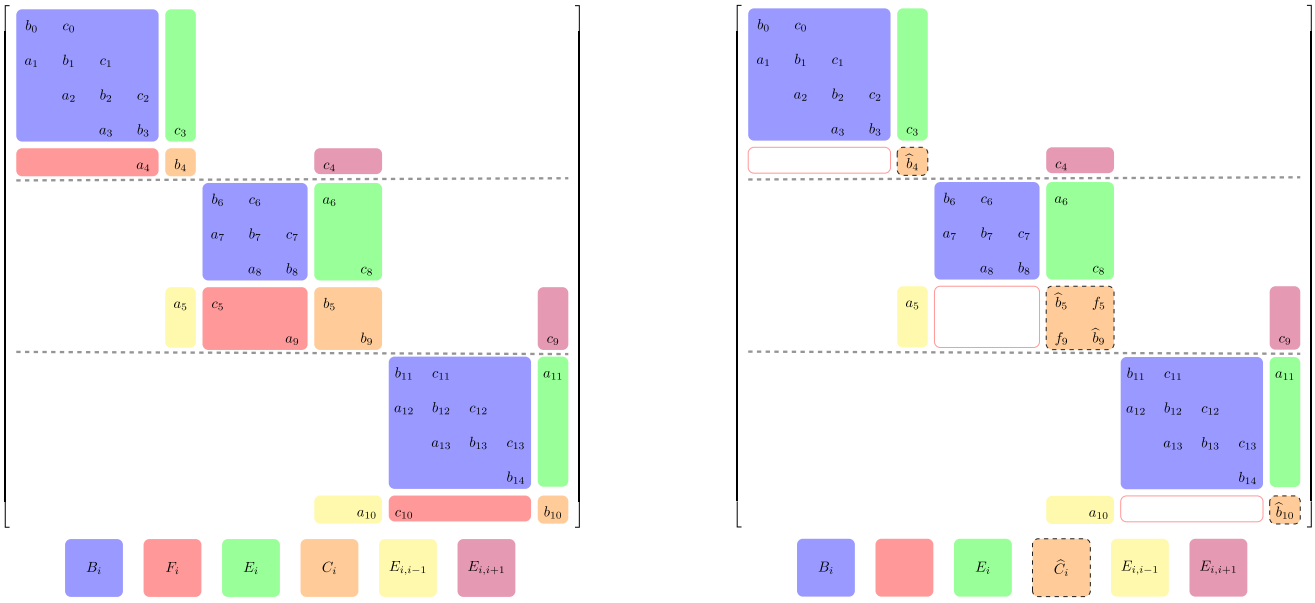


Fig. 5. Block structure of partitioned tridiagonal matrix (left) and decoupling of interface rows (right). $\widehat{b}_i$ denotes b_i values modified by elimination and f_i denotes fill resulting from elimination.

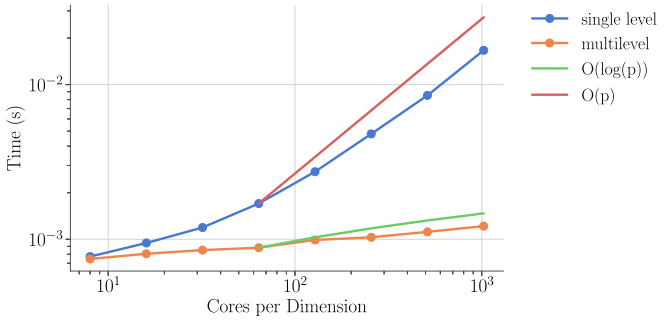


Fig. 6. Weak scaling of line relaxation on Blue Waters with 100 × 100 local problem using single level and recursive partitioning.

The third phase: solve local system The nonzero elements in E_i are first eliminated by substituting the solution values y_i from the second phase. Once complete, the interior rows form independent tridiagonal systems which can be solved on each processor using a sequential tridiagonal solver such as the sequential Thomas algorithm.

By grouping a constant number of processors in the second phase, this approach provides a direct solver with a communication complexity that is logarithmic with respect to the number of processors. As described in [11], the Gaussian elimination steps in the first phase can be completed using storage for only six values to build the $\widehat{C}_i$ blocks thus avoiding the increased cost of storing a factorization. This is especially important in reducing the significant memory requirements in line and plane relaxation where a tridiagonal solve is used on every logical grid line.

Fig. 6 shows the importance of using recursive partitioning to avoid a serial fraction that grows linearly with the number of processes.

3.2. Parallel plane smoothing

Plane smoothing in parallel involves a series of distributed 2D multilevel cycles to solve (3) (see Fig. 8). Using red–black ordering of planes, half of the planes can be processed in parallel without communication across planes. This involves a series of independent parallel tasks with 2D structured communication. Planes on a given processor have uniform communication and computation patterns. They perform the same 2D operations on the same logically structured region. Communication phases for planes on a given processor then involve the same remote processors. Instead of processing the planes sequentially, the uniform communication pattern of planes on a process can be exploited to aggregate communication. As noted in [15,24], the parallel efficiency of plane relaxation can be improved by collecting data to be sent from each 2D parallel solver. This data can then be sent as a single message to reduce the overall latency cost in plane relaxation. To aggregate communication across planes in this way, concurrent execution of 2D parallel plane cycles is needed. In the context of a scalable robust structured solver, coordination of the concurrent execution of these 2D plane cycles is not trivial.

Fig. 7 shows a high-level view of operations needed when using block smoothing with a scalable 3D structured solver. Within each 3D V-cycle, plane smoothing is used at a variety of resolutions. Each plane smoothing step performs a series of 2D V-cycles each of which includes line smoothing at a variety of resolutions. The line smoothing operations use multilevel tridiagonal solvers which recursively repartition data to improve parallel scalability. In addition to the block smoothing operations, each 2D and 3D distributed V-cycle performs coarse-grid redistribution needed for parallel scalability. Since these operations lead to nontrivial execution, we focus on automating the coordination of 2D cycling needed to aggregate communication in plane smoothing.

3.3. Structured data redistribution

Recursive data redistribution is employed in various structured multigrid components to avoid communication complexities that scale proportional to the number of processors used. One dimensional structured data redistribution is used in multilevel line smoothing to solve (12) on a subset of involved processors. This is accomplished by grouping processors into processor blocks that will share the same rows of the reduced system. In this case, a fixed processor grid coarsening factor is used representing the minimum size of a processor group. Some processor groups may be larger than this coarsening factor if it does not evenly divide the total number of processors. An `MPI_Gather` operation is then used within each processor block to gather the interface rows for a block to a single processor. The root processors from each processor block proceed to solve (12). This process is repeated recursively as the parallel solution of (12) forms another interface system on the reduced

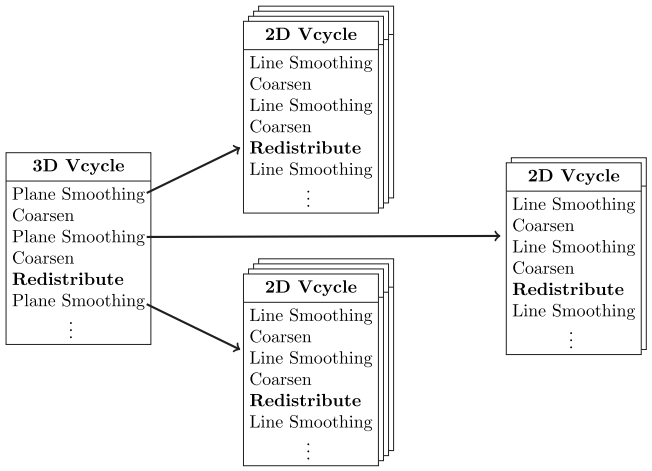


Fig. 7. High-level view of operations in a scalable 3D robust structured V-cycle using block smoothers. A local view of operations is shown, that is, the number of 2D V-cycles (visualized as pages) represents the approximate number of planes per process at a given plane smoothing step. After 3D redistribution, more local work is again available and the number of 2D V-cycles increases. In this example, the first level of 3D plane smoothing includes four local 2D planes. After coarsening, the next level of plane smoothing involves two 2D planes on each process. Finally, after 3D redistribution the local problem size increases and four local planes are again involved in 3D plane smoothing.

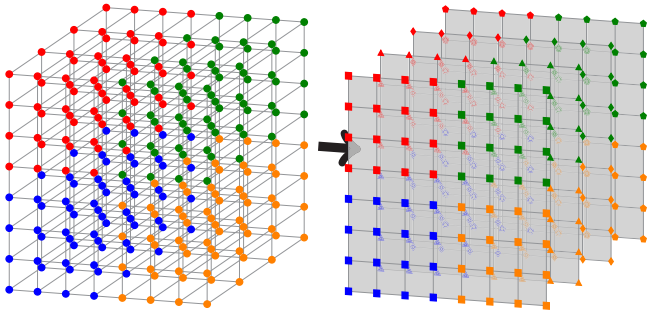


Fig. 8. Distributed plane solves. The colors denote processors, points degrees of freedom, and lines connections in the mesh. The original 3D problem distributed on four processors is shown on the left and distinguished by color. The right visualizes plane solves for this distribution where off plane coefficients are moved to the right-hand side.

set of processors. Once solved, an `MPI_Scatter` is used to propagate the solution to all processors in a processor block.

Structured data redistribution is utilized in two and three dimensions to handle coarse-grid problems. As parallel structured coarsening proceeds, in 2D for plane smoothing and 3D for the outer solve, processors eventually run out of local data. Once a local problem dimension reaches a parameterized minimum size, redistribution of data is considered. For simplicity, a fixed processor grid coarsening factor is used to reduce the processor grid for each dimension that reaches the minimum local size. In the future, development of performance models for line and plane smoothing would enable the use of [4] to optimize this redistribution.

4. Automated communication aggregation

The automation of communication aggregation in plane smoothing enables the reuse of complex distributed solver components, such as scalable 2D multilevel cycling with optimized coarse-grid redistribution and multilevel line smoothing. This approaches involves a simple change to the solver's execution model combined with a set of communication and memory abstractions. This facilitates a 2D distributed solver to be used directly in a plane smoothing routine while having its communication aggregated behind-the-scenes.

4.1. Service abstractions

To automate communication aggregation, we introduced a software layer realized as a set of services to be provided to a distributed multilevel solver instance. The services used for aggregating communication are:

1. **Memory Pool:** Services requests for grid function memory allocations
2. **Halo Exchange:** Exchanges halo data on each multigrid level
3. **Message Passing:** Provides a message passing interface

Implementations of these services are selected dynamically to facilitate aggregation when a 2D solver instance is used within plane relaxation. The memory pool service is used to ensure allocations of fine and coarse-grid vectors that will later be involved in communication are contiguous in memory across planes of the same color. This service is provided to a 2D solver instance and used as a memory backend for its data structures. When a solver instance is involved in plane relaxation and memory for a vector is requested, a memory region is allocated collectively with space for all vectors of a given relaxation color and an address returned with the correct offset into the collective buffer. When the manager plane performs communication, it then has direct access to this collective buffer. This removes the need for the data collection stage when aggregating plane communication thus providing improved memory efficiency. The halo exchange and message passing services provide an interface through which communication in the 2D solver is performed. In this way, aggregated communication routines may be selected when used within a plane relaxation sweep. Even when communication aggregation is not used, the message passing service is necessary to avoid creating redundant MPI communicators in plane relaxation. For instance, coarse-grid redistribution involves creating new communicators for redistributed cycling. Multilevel line relaxation also creates communicators for collective communication at each level and halo exchange libraries typically duplicate the communicator passed to them. When run at scale, the plane relaxation routine quickly reaches the MPI communicator limit in modern implementations if each plane creates these communicators independently.

In plane relaxation, the two main communication routines are a halo exchange and the gather/scatter operations used in multilevel line relaxation. Since vectors involved in communication are contiguous in memory across planes of the same color, the aggregated halo exchange service is implemented by simply using a 3D halo exchange with communication in the direction orthogonal to the plane disabled. The halo regions communicated are then 2D regions consisting of the 1D halos of each plane on a processor. To implement aggregated gather and scatter operations without needing a data collection stage, an MPI user defined data type is used to correctly interleave plane data as shown in Fig. 9.

4.2. Execution model with user-level threading

While the service layer provides a mechanism for presenting different data layouts and communication routines to the plane solvers, a method for coordinating the concurrent execution of the planes is also needed. In contrast to the entirely parallel operation of plane solves of the same color, the addition of communication aggregation necessitates synchronization for each communication operation. Since each 2D solve involves the nontrivial control flow and communication patterns associated with recursive coarse-grid redistribution and multilevel line relaxation, avoiding manual coordination is desired. Context switching is well suited for this type of coordination thus allowing for planes to be suspended and resumed when communication occurs. Using context switching, plane coordination is implemented using Algorithm 1 for

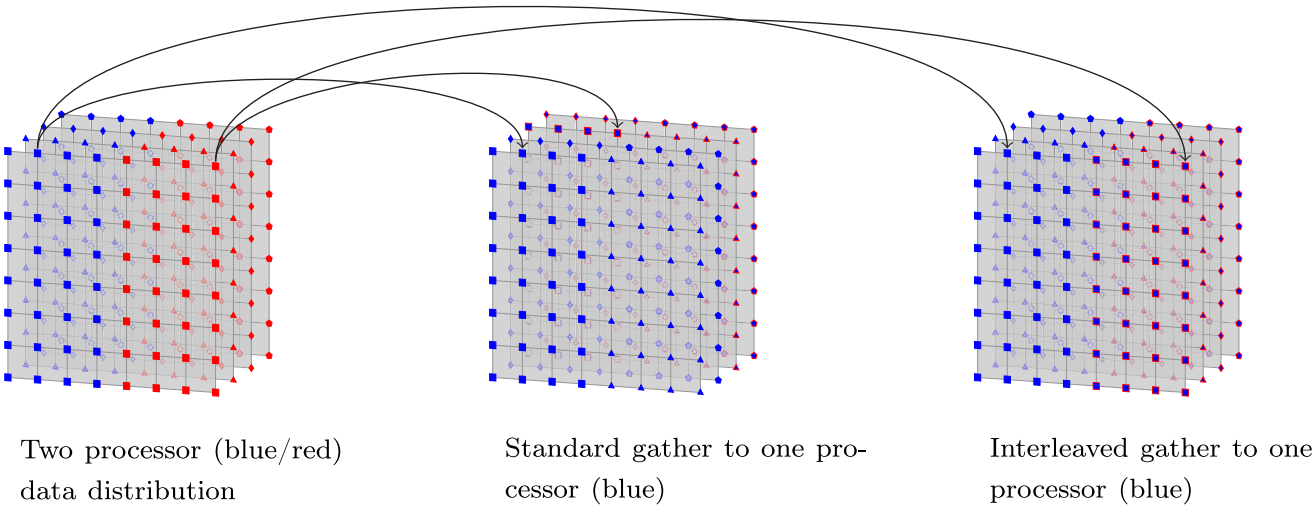


Fig. 9. Aggregated gather/scatter operation of four planes on two processors. Data associated with the first, second, third, and fourth plane are denoted by □, △, ◇, and ⬠. In the case of the first plane, denoted by □, the points reside on two processors labeled in blue and in red: ■, and ■. In the right two figures, all data resides on the blue processor; the edge is colored by its original location — i.e., ■ represents data on the square plane now on the blue process and originally on the red process. The gather and scatter operations interleave plane data to keep consecutive planes contiguous in memory. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

each communication routine. This assumes a one-to-one mapping of plane execution on an MPI rank to computational resources, although synchronization constructs could be added if extra cores are available (e.g., fewer MPI ranks than compute cores). To manage the concurrent execution of plane solves and provide efficient context switching, the lightweight user-level threading library Argobots[1] [25] is used.

Algorithm 1: Aggregated Communication Coordination

Input:
`communicate(. . . )`
  aggregated communication routine
`switch_context(i)`
  routine to switch to plane `i`
`nplanes`
  number of planes
`plane_idx`
  index of plane $(0, \ldots,$ `nplanes` $- 1)$

```
1 switch_context(mod(plane_idx+1, nplanes))
2 if plane_idx = 0                    // manager plane
3 |  communicate(. . . )
```

Fig. 10 shows the execution of planes on an MPI rank. Planes are grouped by color into teams of manager and worker planes. The manager plane is then responsible for running the aggregated communication routines and splitting MPI communicators. During each red sweep, execution of local work begins on the manager plane. When a communication phase is reached, execution is suspended on this plane as a context switch to the first red worker thread is processed. This process repeats until the first stage of local work is completed on each red plane. At this point, a context switch from the last red worker plane back to the red manager plane is processed. The manager plane then performs communication for all red planes and this process continues as the manager plane continues to its next chunk of local work. The overhead of using user-level threads to automate aggregation in this approach is specifically the overhead of the lightweight context switches between each plane incurred by the user-level threading library.

Using lightweight user-level threads with the service layer provides an automated method for aggregating communication for a sequence of distributed tasks with uniform communication patterns. In the context of plane smoothing in a 3D structured solver, this allows the 3D smoothing routine to directly use 2D solver routines for each plane and have their communication aggregated.

5. Experimental results

To assess the parallel efficiency of the approach detailed in Section 4, we consider a number of scaling studies in this section. Throughout these tests, we use the Blue Waters[2] system, a Cray XE machine at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana–Champaign. MPI without multithreading was used with the number of MPI ranks matching the number of floating-point units for all multi node jobs. User-level threading with Argobots was used to orchestrate plane execution; however, only one user-level thread on a process was run at a given time. To improve communication performance for structured grids, we use a rank ordering that matches the application topology to the machine topology [26]. The line and plane smoothers described in this article were implemented in the Cedar framework [27]. This open source package was used for the results in this section.

5.1. Series of uniform distributed tasks

In this section we consider a series of uniform distributed tasks to highlight the value of aggregated communication.

Strong scaling

In the first suite of numerical experiments, we focus on the effectiveness of communication aggregation in reducing communication costs. To this end, we consider a series of uniform distributed tasks comprised of 2D smoothing routines. These routines are run on a sequence of 2D problems with the same parallel data distribution. Runtimes are then compared using automated communication aggregation versus independent execution of the 2D routines.

We consider a 2D model diffusion problem discretized with finite differences on a series of 100 separate 1024×1024 grids. The number

[1] http://www.argobots.org.

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

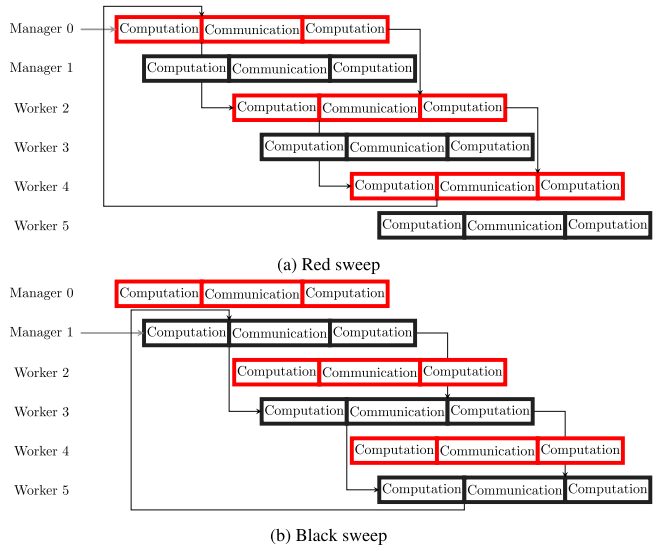


Fig. 10. Execution schedule for red–black plane relaxation. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Strong scaling: 2D grid and processor layouts for a series of distributed smoothing sweeps.

Grid points	Cores	Grid points per core
1024 × 1024	8 × 8	**16,384** = 128 × 128
1024 × 1024	16 × 16	**4,096** = 64 × 64
1024 × 1024	32 × 32	**1,024** = 32 × 32
1024 × 1024	64 × 64	**256** = 16 × 16
1024 × 1024	128 × 128	**64** = 8 × 8

of cores in each coordinate direction is doubled from 8 to 128 cores, to create a range from 16,384 to 64. The corresponding local problem sizes also vary from 64 to 16,384 degrees of freedom per core as summarized in Table 1.

Fig. 11 shows strong scaling behavior of point and line relaxation on a series of 2D grids. In both cases, communication aggregation significantly improves strong scaling with greater speedup as the local problem size decreases.

In the case of sequential point smoothing, parallel efficiency is under 10% at approximately 4k cores in contrast to aggregated point smoothing which reaches an efficiency of 37%. Aggregation in this case extends the strong scaling limit of 10% efficiency from approximately 1k cores to 4k cores. Similarly, sequential line smoothing reaches an efficiency under 10% at approximately 4k cores whereas aggregated line smoothing reaches an efficiency of 30%. Even at 16k cores, aggregated line smoothing retains an efficiency above 10% and the strong scaling limit is extended from 1k cores to 16 cores using aggregation.

The extension of the strong scaling is attributed to dominance of the latency cost, in contrast to local computation and communication bandwidth, as the local problem size decreases. Since communication aggregation specifically targets the overhead of latency, it is most useful in reducing communication on coarser grids.

The use of user-level threads (ULTs) to automate communication aggregation also incurs the overhead of a lightweight context switch for each plane when a communication phase is reached. To quantify this, we consider point smoothing routine that is implemented directly to aggregate communication. This is shown in Fig. 11 as the "manual aggregate point" line. By manually aggregating communication, the overhead incurred by context switches among the 100 2D problems is avoided. As seen in Fig. 11, this overhead is relatively small due to the lightweight user-level threading provided by Argobots.

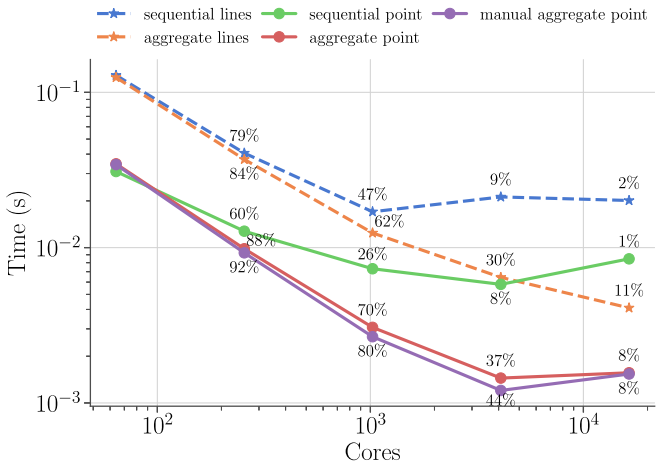


Fig. 11. Strong scaling relaxation routines on Blue Waters with problem size: 1024 × 1024 as in Table 1. The "aggregate" label refers to relaxation run with automated communication aggregation using ULTs; the "sequential" label refers to relaxation that is run independently; and the "manual aggregate" label serves as a baseline, implementing relaxation that is hard-coded to aggregate communication. The parallel efficiency (%) relative to 64 processors is shown for each data point.

Table 2
Weak scaling: 2D grid and processor layouts for a series of distributed smoothing sweeps.

Grid points	Cores	Grid points per core
6,400 = 80 × 80	8 × 8	10 × 10
25,600 = 160 × 160	16 × 16	10 × 10
102,400 = 320 × 320	32 × 32	10 × 10
409,600 = 640 × 640	64 × 64	10 × 10
1,638,400 = 1280 × 1280	128 × 128	10 × 10
640,000 = 800 × 800	8 × 8	100 × 100
2,560,000 = 1600 × 1600	16 × 16	100 × 100
10,240,000 = 3200 × 3200	32 × 32	100 × 100
40,960,000 = 6400 × 6400	64 × 64	100 × 100
163,840,000 = 12,800 × 12,800	128 × 128	100 × 100

Weak scaling

Turning to weak scaling, we again use the 2D model diffusion problem on a series of 100 grids. Local problem sizes are selected near both ends of the strong scaling range, with one chosen away from the strong scaling limit and one near this limit where weak scalability is best challenged, as summarized in Table 2.

Fig. 12 shows that aggregation is critical for improving the performance of point smoothing for the smaller local problem size where communication latency costs dominate. For the larger problem size, aggregation also improves weak scaling efficiency above 1k cores. For this problem size, sequential point smoothing reached an efficiency of 68 percent at approximately 4k cores in contrast to aggregated point smoothing which retains an efficiency of 99 percent. The overhead of using user-level threading to coordinate aggregation is more pronounced in the smaller problem size; however, this cost is relatively low and does not significantly impact weak scalability.

Similar to Fig. 12, Fig. 13 highlights the importance of aggregation for *line* smoothing in the case of the smaller local problem size of 10 × 10. Unlike the constant communication complexity of point smoothing, line smoothing communication scales logarithmically with respect to the number of processors. As result, we observe an increase in cost as the number of processors increases.

5.2. Plane relaxation

To investigate the performance of communication aggregation with plane relaxation, we consider a 3D problem on a range of grids — see Table 3 for details on the strong scaling study. We consider a

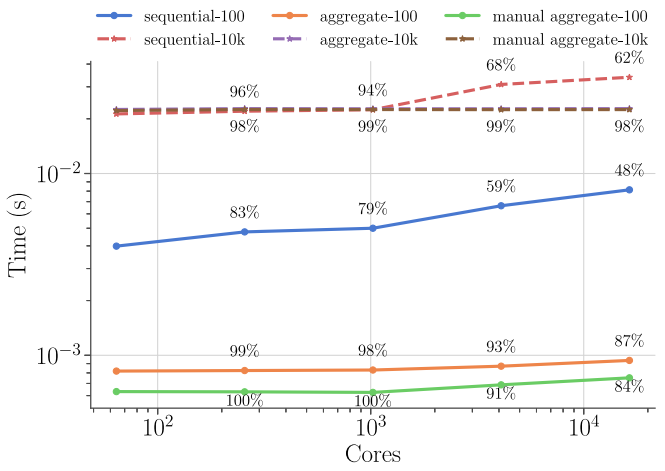


Fig. 12. Weak scaling point relaxation for the problems in Table 2. The "aggregate" label refers to relaxation with automated communication aggregation using ULTs, while the "sequential" label resents the case when the 2D routines were run independently. The parallel efficiency (%) relative to 64 processors is shown for each data point.

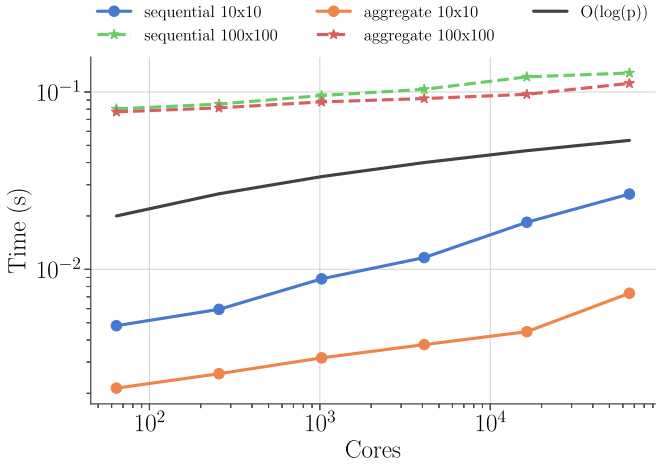
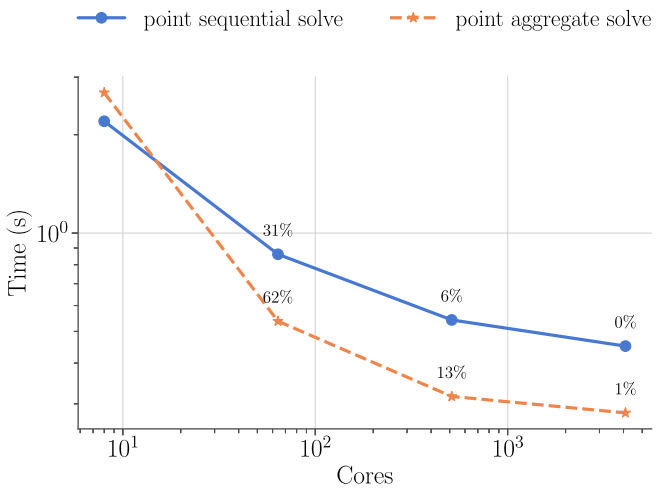


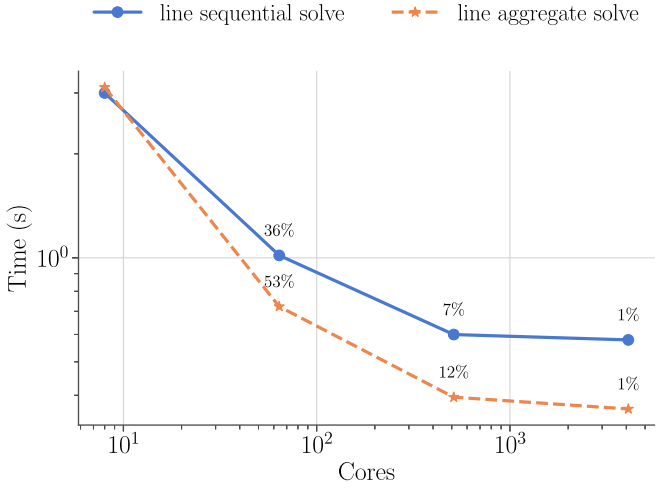
Fig. 13. Weak scaling line relaxation for the problems in Table 2. The "aggregate" label refers to relaxation with automated communication aggregation using ULTs, while the "sequential" label resents the case when the 2D routines were run independently. The parallel efficiency (%) relative to 64 processors is shown for each data point.

problem size of 128 × 128 × 128. The number of processors in each dimension ranged from 2 to 16, and the local problem size ranged from 8 to 64 degrees of freedom in each dimension. Ten V(1,1) 3D multigrid cycles are run using red–black plane relaxation with each plane performing a single V(1,1) cycle. Both the 2D plane cycles and outer 3D cycle use recursive coarse-grid redistribution [4] to balance communication and computation costs at coarse levels. Using a minimum local problem size of 4 grid points in each dimension with coarsening by a factor of 2, a total of 6 multigrid levels are used. The final 4 × 4 × 4 coarse-grid problem is solved using a Cholesky factorization. With the exception of the communication in coarse-grid redistribution, all communication in the multigrid solve phase is aggregated.

Fig. 14 shows strong scaling on Blue Waters where the planes at each level are processed sequentially, and where user-level threads are used to aggregate plane communication on each level. This figure shows communication aggregation improves the strong scalability of the solve — approximately doubling the parallel efficiency.



(a) Planes with point smoothing



(b) Planes with line smoothing

Fig. 14. Strong scaling 3D solve with plane relaxation. The "aggregate" label refers to relaxation with automated communication aggregation using user-level threads, and "sequential" represents independent 2D sweeps. The parallel efficiency (%) relative to 8 processors is shown for each data point.

Table 3
Strong scaling: 3D grid and processor layouts.

Grid points	Cores	Grid points per core
128 × 128 × 128	2 × 2 × 2	**262,144** = 64 × 64 × 64
128 × 128 × 128	4 × 4 × 4	**32,768** = 32 × 32 × 32
128 × 128 × 128	8 × 8 × 8	**4,096** = 16 × 16 × 16
128 × 128 × 128	16 × 16 × 16	**512** = 8 × 8 × 8

5.3. Application

To examine the performance of the approach of Section 4 with an application, we consider calculating the electric potential on several simulation meshes for the ACT-II facility at Illinois [28]. To this end, we will first consider the efficiency of smoothers on an application mesh and then examine the parallel scalability.

Fig. 15 shows a schematic of this facility and the corresponding simulation meshes. To mesh this facility, overlapping logically structured grids are used. The potential of the electric field is given by

$$-\nabla \cdot \epsilon \nabla \phi = f, \tag{13}$$

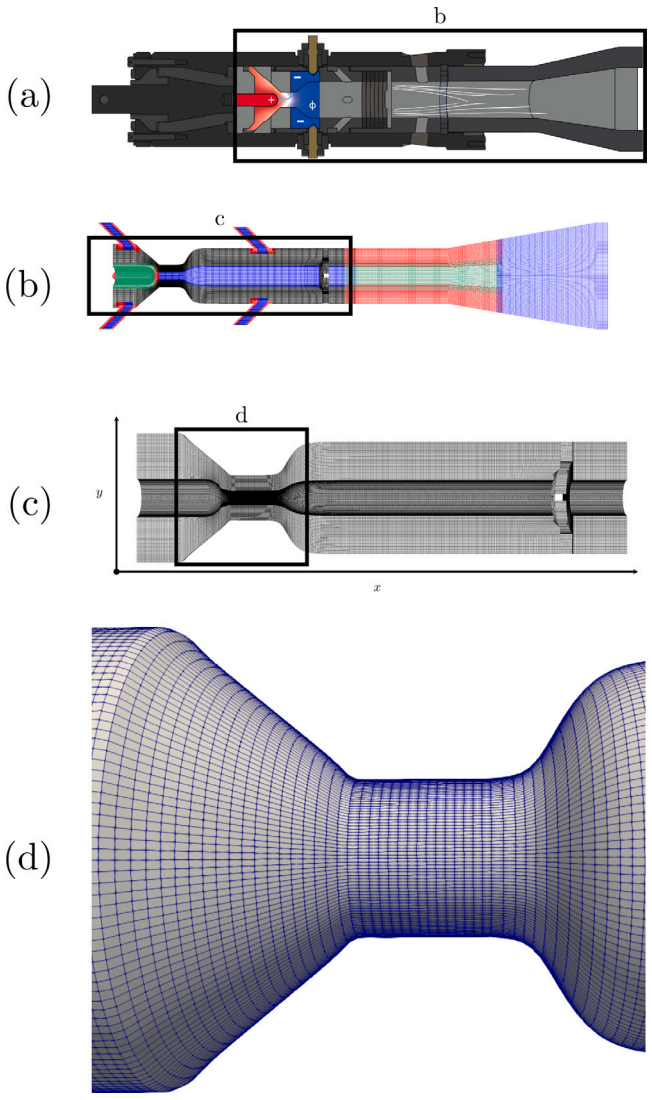


Fig. 15. ACT-II facility at Illinois [28]: (a) a schematic of the upstream section of the facility with labeled electrodes; (b) the set of overset meshes used for simulations; (c) the mesh used for the numerical study; and (d) a zoomed region of the mesh where the cells are most skewed. Mesh and image files produced by Kyle Mackay at the University of Illinois.

where ϵ is the permittivity of the electric field. Computation on each mesh is performed in the logical space $\Xi = (\xi^1, \xi^2, \xi^3)$ given by the unit cube. In curvilinear coordinates, (13) in logical space becomes

$$-\frac{\partial}{\partial \xi^i}\left(\epsilon J g^{ij} \frac{\partial \phi}{\partial \xi^j}\right) = J f, \tag{14}$$

where J is the Jacobian of the transformation, g^{ij} the contravariant metric tensor [29]. The additional metric terms in (14) can introduce anisotropy in the operator depending on the physical geometry of the mesh. To conduct a single-mesh performance study, the main application mesh, shown on the bottom of Fig. 15, is selected. This mesh includes stretching appropriate for plane smoothing and includes the region where the effects of the electric field are most significant to the problem. For this performance study, ϵ is set to 1.

Three resolutions of the mesh in Fig. 15 are considered, with dimensions listed in Table 4. The processor grid sizes are chosen to keep the local problem size approximately constant. In addition, to simplify the study we consider (14) with a constant right hand side and homogeneous Dirichlet boundary conditions.

Table 4
ACT-II mesh sizes.

	Grid points	Logical grid	Cores
Mesh1	1,002,177	$191 \times 159 \times 33$	$4 \times 2 \times 2$
Mesh2	10,000,650	$550 \times 319 \times 57$	$8 \times 5 \times 4$
Mesh3	100,035,208	$1003 \times 959 \times 104$	$16 \times 12 \times 8$

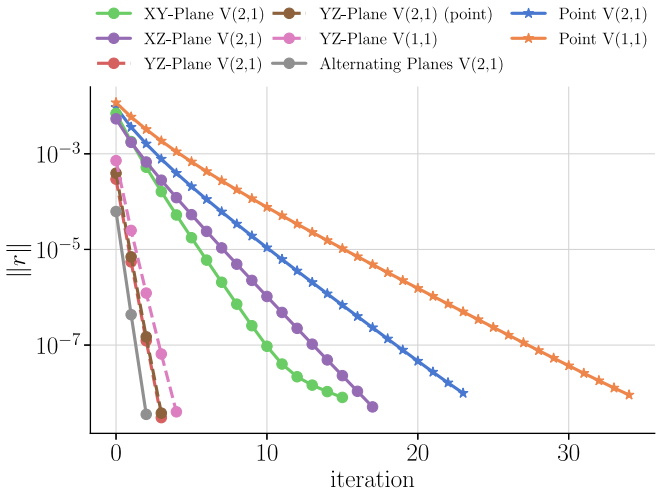


Fig. 16. Convergence of Cedar using a variety of smoothers with Mesh1 from Table 4. This corresponds to the mesh shown in Fig. 15(c). Alternating and yz-plane smoothing result in the best convergence for this mesh.

Fig. 16 shows convergence of Cedar with Mesh1 from Table 4 using a variety of smoothers. Using point smoothing with this problem resulted in the poorest convergence. This is due to anisotropy introduced from mapping this mesh to logical space in (13). Alternating plane smoothing — i.e., alternates among xy, xz, yz planes — is a robust smoother for anisotropic problems and results in the best convergence for this problem. For this problem, yz plane smoothing exhibits similar convergence to alternating plane smoothing, in contrast to both xy and xz plane smoothing. This shows coupling for this problem is in the yz direction.

To analyze the various relaxation sweeps further, we consider the *stretching* in the x, y, and z directions. Fig. 17 shows that there is less stretching in the y and z directions, which leads to stronger coupling in the operator when mapped to the computational domain. Fig. 17 also shows stretching in the y and z directions are related, leading to a nearly isotropic yz plane. For this reason line smoothing is not needed for the yz planes. This is shown in the similar convergence behavior of Cedar with yz planes using point smoothing and line smoothing from Fig. 16.

Fig. 18 shows reduction in the residual norm over wall-clock time using Cedar with a variety of smoothers using Mesh1 from Table 4. xz and xy plane smoothing results in the slowest overall residual reduction. These smoothers demonstrate poor convergence with this problem as seen in Fig. 16 while incurring the cost of plane smoothing. Point smoothing and alternating plane relaxation result in a similar rate of residual reduction for this problem as they are balanced on opposing sides of convergence and cost per iteration.

Fig. 19 shows multigrid setup and solve times of Cedar using point and plane relaxation using the meshes and processor decompositions shown in Table 4. Setup and solve times for the parallel algebraic multigrid solver BoomerAMG from Hypre [30] are also included; this highlights the potential performance benefit of using a robust structured solver for this problem.

The effectiveness of the solver strongly depends on the metric terms in (14). To provide a measure of the anisotropy introduced to the operator from mapping each mesh from Table 4 to a logically structured

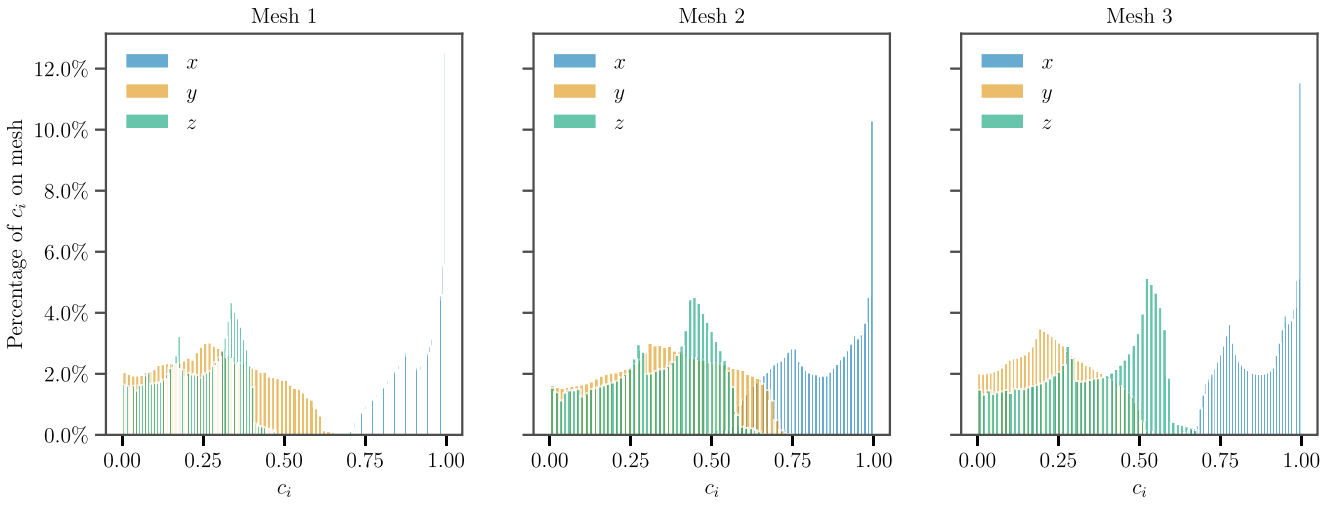


Fig. 17. Components of normalized cell diagonals c_i from (15) on the mesh. Larger values on the x-axis indicate strong stretching in that coordinate direction. Stretching for each mesh is strongest in the x direction as there is a higher percentage of cells with large values in that direction. yz plane smoothing is effective with these meshes due to the strong coupling in the yz plane.

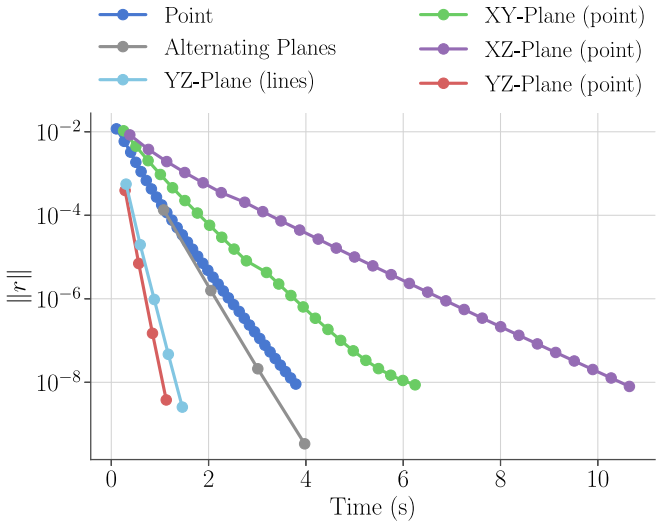


Fig. 18. Residual reduction over wall-clock time of Cedar using a variety of smoothers with Mesh1 from Table 4. (lines) and (point) indicate whether line or point smoothing was used for each plane cycle. YZ-Plane smoothing results in the fastest residual reduction for this problem.

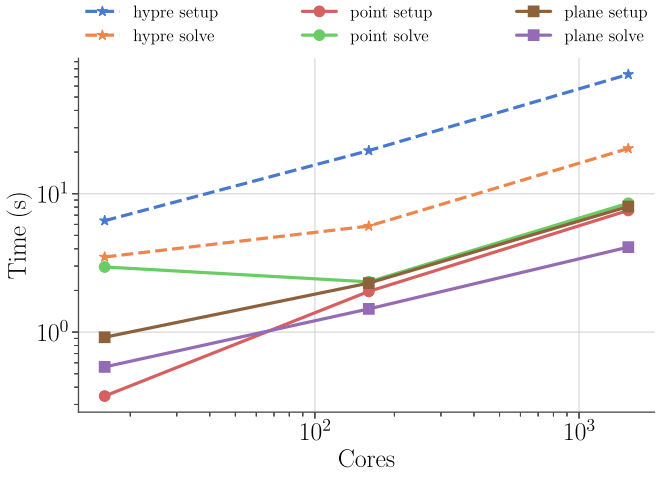


Fig. 19. Scaling study with an approximately constant number of grid points on each processor. Multigrid setup and solve times are shown with BoomerAMG from Hypre and structured multigrid from Cedar using both point and plane relaxation.

grid with unit spacing, the following metric is considered. For each interior vertex on the mesh, the vector

$$\mathbf{c} = \frac{1}{\sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2}} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \tag{15}$$

is computed. This represents the normalized diagonal of each cell on the mesh. The scalar projection of $\mathbf{c}$ onto $\mathbf{1} \in \mathbb{R}^3$ is then used as a single number to quantify mesh anisotropy. Computed for each cell of the mesh, this is used as a metric for mesh stretching as shown in Fig. 20. With a higher percentage of cells close to square, Mesh2 results in a smaller amount of anisotropy compared to Mesh1 and Mesh3. This impacts solver convergence with point smoothing — the iteration counts to reach the fixed relative tolerance using point smoothing in Fig. 19 were 48 with Mesh1, 27 with Mesh2, and 49 with Mesh3. This resulted in a smaller solve time using point smoothing with Mesh2 compared to Mesh1 even though Mesh2 had approximately ten times the number of grid points.

6. Conclusions

To achieve performance at scale, robust structured solvers require efficient coarse-grid correction and smoothing routines. While optimized coarse-grid redistribution [4] provides an efficient strategy for scaling the application topology with diminishing local work, efficient smoothing routines to handle anisotropic problems are also needed for robustness. In this paper, the scalable parallel tridiagonal solver proposed in [11] is used to provide efficient line relaxation without the need to store a factorization for each line. Parallel results from Fig. 6 confirm the logarithmic communication complexity. To improve the efficiency of robust structured smoothing in 3D, an automated strategy for aggregating communication for a series of distributed tasks with uniform communication patterns is proposed. Using lightweight user-level threads to manage the concurrent execution of plane solves and a service layer to specialize communication routines and memory allocation, manual coordination and data collection among planes is avoided. Parallel results show this approach is effective in reducing communication costs for coarse-grid computations—producing an 8.7× speedup in relaxation routines shown in Fig. 12. This significantly improves strong scalability—approximately doubling parallel efficiency

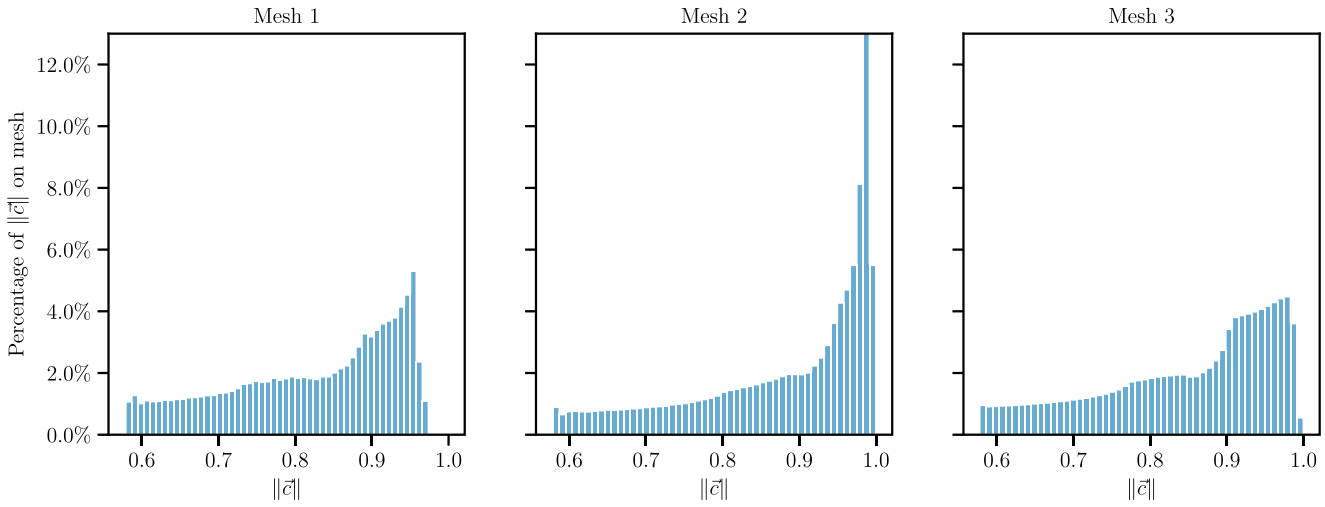


Fig. 20. Scalar projection of normalized mesh cell diagonals onto $\mathbf{1} \in \mathbb{R}^3$. Values of 1.0 indicate square mesh cells.

in plane smoothing for local problem sizes between 8k and 32k degrees of freedom per processor.

Using user-level threads to orchestrate plane execution in a multilevel solve also exposes additional communication optimizations. The ability to inexpensively switch contexts among planes facilitates communication and computation overlap across planes. For example, communication from one plane can be overlapped with computation from another using nonblocking communication routines. Communication aggregation can then be used to maintain useful overlap at a variety of resolutions by aggregating communication for a subset of planes on a process. This is applicable when plane smoothing is used within a multilevel V-cycle where smoothing at a variety of resolutions is required.

The focus of this work is dedicated to solvers on large structured meshes. Revisiting this approach in a multiblock setting may require hybrid methods and a potential avenue for future work.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.parco.2020.102705.

Acknowledgments

Special thanks to Kyle Mackay at the University of Illinois at Urbana–Champaign for the mesh and image files in Fig. 15.

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, United States of America under Award Number DE-NA0002374, and at Los Alamos National Laboratory, United States of America under Contract Number DE-AC52-06NA25396, and was partially supported by the Advanced Simulation and Computing/Advanced Technology Development and Mitigation Program, United States of America.

This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation, United States of America (awards OCI-0725070 and ACI-1238993) and the state of Illinois, United States of America. Blue Waters is a joint effort of the University of Illinois at Urbana–Champaign and its National Center for Supercomputing Applications.

References

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

[2] H. Gahvari, W. Gropp, K.E. Jordan, M. Schulz, 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, in: IPDPSW '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 1675–1682.

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

[4] A. Reisner, L. Olson, J. Moulton, Scaling structured multigrid to 500k+ cores through coarse-grid redistribution, SIAM J. Sci. Comput. 40 (4) (2018) C581–C604, http://dx.doi.org/10.1137/17M1146440.

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

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

[7] A.H. Sameh, D.J. Kuck, On stable parallel linear system solvers, J. ACM 25 (1) (1978) 81–91.

[8] E. Polizzi, A.H. Sameh, A parallel hybrid banded system solver: the spike algorithm, Parallel Comput. 32 (2) (2006) 177–194, Parallel Matrix Algorithms and Applications (PMAA'04).

[9] L.-W. Chang, J.A. Stratton, H.-S. Kim, W.-M.W. Hwu, A scalable, numerically stable, high-performance tridiagonal solver using gpus, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, in: SC '12, IEEE Computer Society Press, Los Alamitos, CA, USA, 2012, pp. 27:1–27:11, URL http://dl.acm.org/citation.cfm?id=2388996.2389033.

[10] H.S. Kim, S. Wu, L. w. Chang, W. m. W. Hwu, A scalable tridiagonal solver for gpus, in: 2011 International Conference on Parallel Processing, 2011, pp. 444–453.

[11] T.M. Austin, M. Berndt, J.D. Moulton, A Memory Efficient Parallel Tridiagonal Solver, Tech. Rep. LA-UR 03-4149, Mathematical Modeling and Analysis Group, Los Alamos National Laboratory, Los Alamos, NM, 2004.

[12] S. Schaffer, A semicoarsening multigrid method for elliptic partial differential equations with highly discontinuous and anisotropic coefficients, SIAM J. Sci. Comput. 20 (1) (1998) 228–242.

[13] J. Dendy Jr, Semicoarsening multgrid for systems, Electron. Trans. Numer. Anal. 6 (1997) 97–105.

[14] C.-A. Thole, U. Trottenberg, Basic smoothing procedures for the multigrid treatment of elliptic 3d operators, Appl. Math. Comput. 19 (1–4) (1986) 333–345.

[15] U. Trottenberg, A. Schuller, Multigrid, Academic Press, Inc., Orlando, FL, USA, 2001.

[16] W.L. Briggs, V.E. Henson, S.F. McCormick, A Multigrid Tutorial, second ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[17] R.W. Hockney, A fast direct solution of poisson's equation using fourier analysis, J. ACM 12 (1) (1965) 95–113.

[18] H.S. Stone, An efficient parallel algorithm for the solution of a tridiagonal linear system of equations, J. ACM 20 (1) (1973) 27–38.

[19] H.H. Wang, A parallel method for tridiagonal equations, ACM Trans. Math. Software 7 (2) (1981) 170–183.
[20] L. Brugnano, A parallel solver for tridiagonal linear systems for distributed memory parallel computers, Parallel Comput. 17 (9) (1991) 1017–1023.
[21] I.N. Hajj, S. Skelboe, A multilevel parallel solver for block tridiagonal and banded linear systems, Parallel Comput. 15 (1) (1990) 21–45.
[22] A. Krechel, H.-J. Plum, K. Stüben, Parallelization and vectorization aspects of the solution of tridiagonal linear systems, Parallel Comput. 14 (1) (1990) 31–49.
[23] P. Amodio, L. Brugnano, T. Politi, Parallel factorizations for tridiagonal matrices, SIAM J. Numer. Anal. 30 (3) (1993) 813–823, http://dx.doi.org/10.1137/0730041.
[24] U. Gärtel, A. Krechel, A. Niestegge, H.-J. Plum, Parallel multigrid solution of 2d and 3d anisotropic elliptic equations: Standard and nonstandard smoothing, in: Multigrid Methods III, Birkhäuser Basel, Basel, 1991, pp. 191–209.
[25] S. Seo, A. Amer, P. Balaji, C. Bordage, G. Bosilca, A. Brooks, P. Carns, A. Castelló, D. Genet, T. Herault, S. Iwasaki, P. Jindal, L.V. Kalé, S. Krishnamoorthy, J. Lifflander, H. Lu, E. Meneses, M. Snir, Y. Sun, K. Taura, P. Beckman, Argobots: A lightweight low-level threading and tasking framework, IEEE Trans. Parallel Distrib. Syst. 29 (3) (2018) 512–526.
[26] W.D. Gropp, Using node information to implement mpi cartesian topologies, in: Proceedings of the 25th European MPI Users' Group Meeting, in: EuroMPI'18, ACM, New York, NY, USA, 2018, pp. 18:1–18:9.
[27] D. Moulton, L.N. Olson, A. Reisner, Cedar framework, 2017, Version 0.1, URL https://github.com/cedar-framework/cedar.
[28] D. Baccarella, Q. Liu, T. Lee, S.D. Hammack, H. Do, The supersonic combustion facility ACT-2, in: 55th AIAA Aerospace Sciences Meeting, 2017, pp. 1–2, http://dx.doi.org/10.2514/6.2017-0103.
[29] V.D. Liseikin, Grid Generation Methods, second ed., Springer Publishing Company, Incorporated, 2009.
[30] R.D. Falgout, U.M. Yang, Hypre: a library of high performance preconditioners, in: International Conference on Computational Science, Springer, 2002, pp. 632–641.