A scalable geometric multigrid method for non-symmetric linear systems arising from elliptic equations

By M. Esmaily-Moghadam, L. Jofre, G. Iaccarino AND A. Mani

1. Motivation and objectives

Elliptic partial differential equations (PDE) are commonplace in physics. A few examples range from heat transfer in solids to the effect of pressure in incompressible flows with variable density. Solutions to these PDE systems, under various boundary conditions as well as geometries, are not often available analytically. As a result, various numerical methods have been adopted to discretize the problem in space and time and to reduce it to a system of linear equations. Solving these systems remains a challenge, because scientific and engineering applications are continuously growing in size, demanding more efficient solvers (Saad 2003; Shakib *et al.* 1989; Saad *et al.* 1998; Esmaily-Moghadam *et al.* 2013, 2015*a*,*b*).

In a common PDE solver, a large portion of the computational cost is dedicated to solving the underlying linear system. These linear systems have a typical size of over hundreds of thousands of unknowns. Since direct methods are of $\mathcal{O}(N^p)$ with N being the number of unknowns and p generally greater than 1.5, they quickly become an unattractive option in terms of computational cost (Saad 2003). To offset the cost, iterative methods are generally favored in these applications. The high computational cost, however, remains a critical issue in solving linear systems obtained from the Poisson equation (a subset of elliptic PDEs). This high cost is caused by the dependency of the local solution on the solution of the entire domain. In this class of problems, the entire domain is affected by a perturbation at any arbitrary point in space. This strong coupling introduces several challenges in terms of ill-conditioning of the linear system as well as designing efficient parallel algorithms.

The ill-conditioning becomes a pressing issue as the size of the system increases (Saad 2003). To explain this in simple terms, consider two fixed points in space that are at the location of two unknowns on the discrete setting. The number of unknowns between these two points varies proportional to the total number of unknowns. Considering the intrinsic characteristic of the iterative methods, in which at each iteration information is propagated from a point to its neighbors, the number of iterations required for propagation of information between two points increases proportionally with the total number of grid points, i.e., the size of the system. This effect, that is also referred to as ill-conditioning of the linear system, leads to a scaling versus size worse than $\mathcal{O}(N)$ in iterative solvers. Although techniques based on Fourier transformation, which are of $\mathcal{O}(N)\log(N)$, do not encounter this challenge, their applicability is limited to only symmetric systems with very simple geometries and boundary conditions. Another effective technique to tackle this issue is the multigrid approach, which can be divided into two classes: algebraic and geometric (Wesseling 1995; Vaněk *et al.* 1996; Ghia *et al.* 1982). The algebraic multigrid techniques, although applicable to a wider range of linear systems, are often less robust, more complex to implement, and less efficient than the geometric counterpart. The geometric multigrid methods, on the other hand, are well suited for applications with simple geometry that are large enough in size to justify the added complexity.

In this study, we target a class of systems that arise from a specific physical problem. The underlying physical problem is flow in a duct, which is a cuboid domain with walls on four sides and an inflow-outflow boundary conditions on the third direction. Point particles are dispersed in the flow and the entire apparatus is subjected to radiative heating, causing temperature and density variations. The Navier-Stokes equations are formulated based on the low Mach assumption, leading to a variable density, but incompressible governing equations. Using a fractional time-stepping method, the conservation of mass is satisfied by correcting the velocity field through solving a linear system (Jofre *et al.* 2014). This linear system, as a result of density variation and grid non-uniformity, is not symmetric. For accuracy and cost considerations, the grid is refined near the walls, leading to a wide spectrum of cell sizes that translates to a linear system with diagonal entries varying significantly. Therefore, the objective of this study is to design an efficient and scalable linear solver for a non-symmetric linear system obtained from an elliptic operator on a cuboid domain. We target systems with up to a billion unknowns, a typical size for direct numerical simulation. Based on the earlier discussion, the geometric multigrid method is well suited for this class of problems, considering the simplicity of cuboid geometry and the size of the linear system.

The massive size of the linear systems under consideration mandates the use of parallel computations. The main challenge here is to design an algorithm that scales up to a large number of processors without significantly losing parallel efficiency. The design of such an algorithm is fully intertwined with the underlying iterative method, because a scalable algorithm minimizes processor-to-processor communications and maintains a good load balance on all processors. A multigrid technique that relies on a static partitioning of a stretched grid can encounter load imbalance on coarser levels. To address these challenges, we will discuss a partitioning approach for the introduced geometric multigrid technique.

This paper is organized as follows. First, we present a geometric multigrid technique for stretched grids that relies on a mapper between non-uniform and uniform grids, and discuss a recursive implementation of this algorithm for optimal convergence. Then, we introduce our partitioning approach, designed for improved parallel scalability. To benchmark our method, we compare it against a multigrid technique from the Trilinos package (Hestenes & Stiefel 1952; Heroux *et al.* 2005) and conventional Krylov-based iterative methods. In Section 4 we present: (1) triply periodic isotropic turbulence on uniform grids, and (2) variable density duct simulations on non-uniform grids. We present weak and strong scaling results to establish the parallel performance of our method. Finally, we draw conclusions in Section 5.

2. A geometric multigrid method

In this section, we discuss our geometric multigrid method, which includes restriction (fine-to-coarse) and interpolation (coarse-to-fine) operations, a recursive implementation of the multigrid method, and the main algorithm. In what follows, roman superscripts are used to denote variable names and italic subscripts are used as indices. Superscripts c and f are used to denote variables on coarse and fine grids, respectively. Indices i, j, and k are used for fine, and I, J, K are used for coarse grid.

The multigrid technique tackles the ill-conditioning issue by reducing the number of grid points in all directions, thus reducing the number of iterations required for the propagation of information. Hence, given the linear system defined on a fine grid

$$\boldsymbol{A}^{\mathrm{f}}\boldsymbol{x}^{\mathrm{f}} = \boldsymbol{b}^{\mathrm{f}}, \tag{2.1}$$

the objective is to solve an equivalent system on a coarse grid as

$$\boldsymbol{A}^{\mathrm{c}}\boldsymbol{x}^{\mathrm{c}} = \boldsymbol{b}^{\mathrm{c}}, \tag{2.2}$$

such that the difference between $\boldsymbol{x}^{\mathrm{c}}$ and $\boldsymbol{x}^{\mathrm{f}}$ is minimal in a physical sense. To obtain Eq. (2.2) from Eq. (2.1), one needs to map $\boldsymbol{b}^{\mathrm{f}}$ to $\boldsymbol{b}^{\mathrm{c}}$ and define $\boldsymbol{A}^{\mathrm{c}}$ such that solutions to both systems are similar. To achieve this goal, we take the PDE that produces Eq. (2.1) and re-discretize it on a coarse grid. This process reduces to two core operations, which are mapping a field from the fine grid to the coarse grid and vice versa. We denote the former (restriction) by $\mathcal{C}$ and the latter (interpolation) by $\mathcal{F}$. Exploiting these two operators, the right-hand side (RHS) of Eq. (2.1) is mapped to the coarse grid using

$$\boldsymbol{b}^{\mathrm{c}} = \mathcal{C}(\boldsymbol{b}^{\mathrm{f}}), \tag{2.3}$$

and the solution to Eq. (2.2) is mapped back to the fine grid using

$$\boldsymbol{x}^{\mathrm{f}} = \mathcal{F}(\boldsymbol{x}^{\mathrm{c}}). \tag{2.4}$$

We denote the cardinality of a set or vector by $|\bullet|$, $\mathcal{F} : \mathbb{R}^{|x^{\mathrm{c}}|} \mapsto \mathbb{R}^{|x^{\mathrm{f}}|}$ and $\mathcal{C} : \mathbb{R}^{|b^{\mathrm{f}}|} \mapsto \mathbb{R}^{|b^{\mathrm{c}}|}$.

The left-hand side matrices in Eqs. (2.1)-(2.2) are discrete non-symmetric elliptic operators. In a continuous form, they represent a class of PDEs as

$$\nabla \cdot (\kappa \nabla T) = q, \tag{2.5}$$

which fits to the heat equation in a solid with variable material property, κ, under volumetric heating, $-q$, and temperature, T. The heat equation is chosen to provide a physical intuition. It directly translates to the equation solved for satisfying the continuity equation in flow with variable density by replacing T with pressure, κ with the inverse of density, and q with the divergence of uncorrected velocity field plus the contribution from the energy equation. Denoting the discrete form of an elliptic operator on grid $\mathcal{G}$ for a given κ by $\mathcal{D}_\kappa\{\mathcal{G}\}$, by definition

$$\boldsymbol{A}^{\mathrm{f}} = \mathcal{D}_{\kappa^{\mathrm{f}}}\{\mathcal{G}^{\mathrm{f}}\}. \tag{2.6}$$

Through numerical experiments, in which we considered various combination of calculating $\boldsymbol{A}^{\mathrm{c}}$ directly from $\boldsymbol{A}^{\mathrm{f}}$ and the underlying PDE itself, we found that

$$\boldsymbol{A}^{\mathrm{c}} = \mathcal{D}_{\kappa^{\mathrm{c}}}\{\mathcal{G}^{\mathrm{c}}\} \tag{2.7}$$

provides a good estimate of an optimal $\boldsymbol{A}^{\mathrm{c}}$. Specifically, an optimal $\boldsymbol{A}^{\mathrm{c}}$, given arbitrary non-singular $\boldsymbol{A}^{\mathrm{f}}$ and $\boldsymbol{b}^{\mathrm{f}}$, can be potentially defined as

$$\mathrm{argmin}_{\boldsymbol{A}^{\mathrm{c}}} \left\| (\boldsymbol{A}^{\mathrm{f}})^{-1}\boldsymbol{b}^{\mathrm{f}} - \mathcal{F}\left((\boldsymbol{A}^{\mathrm{c}})^{-1}\mathcal{C}(\boldsymbol{b}^{\mathrm{f}}) \right) \right\|. \tag{2.8}$$

Before proceeding any further, it is necessary to exactly specify operators $\mathcal{C}$ and $\mathcal{F}$. To satisfy conservation of energy (or mass), one needs to ensure that the integral of q in Eq. (2.5) over the entire computational domain is preserved. In a discrete setting, this constraint translates to a similar condition on $\mathcal{C}$ pertaining to Eq. (2.3). To ensure this constraint is satisfied on any interval, we enforce it on all grid points, i.e., b^{c}_I with $I \in \{1, \cdots, |\boldsymbol{b}^{\mathrm{c}}|\}$. This is accomplished by summing $\boldsymbol{b}^{\mathrm{f}}$ over all the cells that overlap with b^{c}_I. This is schematically shown for a one-dimensional grid in Figure 1.

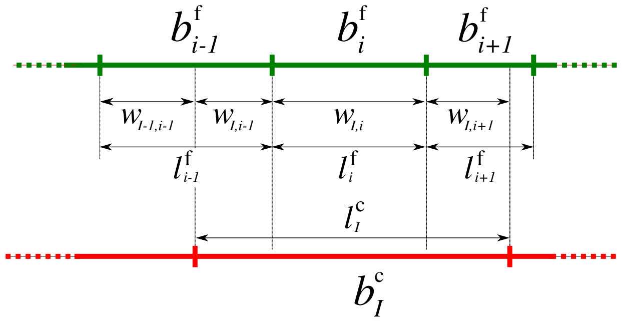

FIGURE 1. One-dimensional mapping of a field $\boldsymbol{b}$ between a fine (top/green) and a coarse (bottom/red) grid.

Based on Figure 1, defining $w_{I,i}$ as the fraction of length of cell i that overlaps with cell I, $\mathcal{C}$ operator turns into a multiplication by matrix $\boldsymbol{w}$ in one dimension, i.e., $\boldsymbol{b}^{\mathrm{c}} = \boldsymbol{w}\boldsymbol{b}^{\mathrm{f}}$. Note that $w_{I,i} \geq 0$ and $\boldsymbol{w}\boldsymbol{e}^{\mathrm{f}} = \boldsymbol{l}^{\mathrm{c}}$, where $\boldsymbol{l}^{\mathrm{c}}$ is an array that contains the length of coarse grid cells and $\boldsymbol{e}^{\mathrm{f}}$ is a vector with one at all its entries defined on $\mathcal{G}^{\mathrm{f}}$. Also, $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{e}^{\mathrm{c}} = \boldsymbol{l}^{\mathrm{f}}$, in which $\boldsymbol{l}^{\mathrm{f}}$ is an array that contains length of coarse grid cells. These two properties are a direct consequence of the fact that the integral of a quantity is preserved by $\boldsymbol{w}$.

In three dimensions, $\boldsymbol{w}$ is extended to three matrices, $\boldsymbol{w}^{\mathrm{x}}$, $\boldsymbol{w}^{\mathrm{y}}$, and $\boldsymbol{w}^{\mathrm{z}}$ for x, y, and z directions, respectively. $\mathcal{C}$ results into three summations as

$$\mathcal{C}(\boldsymbol{b}^{\mathrm{f}})_{IJK} = b^{\mathrm{c}}_{IJK} = \sum_i \sum_j \sum_k w^{\mathrm{x}}_{I,i} w^{\mathrm{y}}_{J,j} w^{\mathrm{z}}_{K,k} b^{\mathrm{f}}_{ijk}. \tag{2.9}$$

In Eq. (2.9), each summation must be carried out on a limited number of overlapping cells; hence, the computational cost remains as $\mathcal{O}(|\boldsymbol{b}^{\mathrm{f}}|)$.

Operator $\mathcal{F}$ is also defined and extended to three dimensions in a similar manner. The only difference is that $\boldsymbol{w}$ is normalized by $\boldsymbol{l}^{\mathrm{f}}$ in this case. Defining

$$u_{i,I} = \frac{w_{I,i}}{l^{\mathrm{f}}_i}, \tag{2.10}$$

in one dimension we have $\boldsymbol{b}^{\mathrm{f}} = \boldsymbol{u}\boldsymbol{b}^{\mathrm{c}}$, and in three dimensions

$$\mathcal{F}(\boldsymbol{b}^{\mathrm{c}})_{ijk} = b^{\mathrm{f}}_{ijk} = \sum_I \sum_J \sum_K u^{\mathrm{x}}_{i,I} u^{\mathrm{y}}_{j,J} u^{\mathrm{z}}_{k,K} b^{\mathrm{c}}_{IJK}, \tag{2.11}$$

in which $\boldsymbol{u}^{\mathrm{x}}$, $\boldsymbol{u}^{\mathrm{y}}$, and $\boldsymbol{u}^{\mathrm{z}}$ are the corresponding weights for x, y, and z directions, respectively. Note $\boldsymbol{u}\boldsymbol{e}^{\mathrm{c}} = \boldsymbol{e}^{\mathrm{f}}$, viz. a field on $\mathcal{G}^{\mathrm{c}}$ with constant value is mapped to a constant value on $\mathcal{G}^{\mathrm{f}}$.

Conventionally, multigrid methods have a preset pattern of restriction and interpolation. The most common patterns are V cycle, i.e., m restriction followed by m interpolation, and W cycle, i.e., m levels of restriction followed by $\tilde{m} \leq m$ interpolation, $\tilde{m}$ restriction, and m interpolation. The performance of each of these methods depends on the quality of the solution on the coarser levels. For one class of problems, one cycle of restriction might be sufficient, while for another, several cycles might be necessary. Therefore, to achieve an optimal performance, we do not preset a pattern, but rather allow it to be determined dynamically (see Figure 2). This is achieved by adopting a recursive implementation. The goal of this algorithm is to optimally solve a linear system at any given level, regardless of the number of previous restrictions. Achieving this

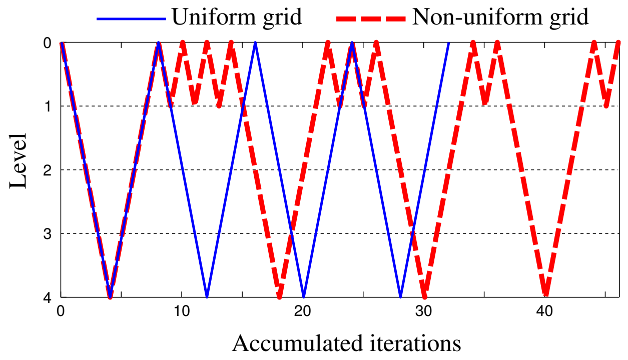

FIGURE 2. Dynamic pattern of multigrid restriction and interpolations for two representative cases. Blue is a uniform 128^3 grid and red is a non-uniform $120 \times 80 \times 80$ grid.

objective at any level (including the coarsest level), by recursion, the entire algorithm would be optimal.

A second important factor for achieving an optimal performance is to ensure smoothing operations, viz. reducing residuals at any given level including the coarsest level, are also optimal. The Krylov-based iterative methods, namely the conjugate gradient (CG) for symmetric matrices and the bi-conjugate gradient (BCG) for non-symmetric matrices, are among the most efficient smoothing techniques. Therefore, we use the CG and BCG for symmetric and non-symmetric matrices, respectively. Experimenting with other smoothing techniques such as Gauss-Seidel technique has confirmed near optimality of our choice of smoothers.

In general, the CG outperforms the BCG. Exploiting this to maximize the performance, we ensure all restriction operations lead to a symmetric matrix. This way, the BCG is only used to solve a given non-symmetric matrix at the finest level, i.e., smoothing of the original system. To ensure a restriction operation leads to a symmetric matrix, we build the coarse grid such that it is uniform in any given direction, i.e., $l_i^{\mathrm{c}} = \overline{\boldsymbol{l}^{\mathrm{c}}}$, in which

$$\overline{\boldsymbol{l}^{\mathrm{c}}} \equiv |\boldsymbol{l}^{\mathrm{c}}|^{-1} \sum_i l_i^{\mathrm{c}} \tag{2.12}$$

is the average grid size. Note that $|\boldsymbol{l}^{\mathrm{c}}|$ and $\sum_i l_i^{\mathrm{c}}$ are equal to the number of grid points and the total length of the domain, respectively. Through numerical experiments, we found that the best restriction strategy is to increase the average grid size at maximum by a factor of two in each direction such that the average grid size in all directions merges to the same value. On a highly stretched grid this translates to significant coarsening to no coarsening (or even refining) on different segments of the grid. This strategy, however, remains superior in comparison with a coarsening strategy that ensures similar degree of coarsening in the entire domain. Mathematically, given the average grid size of the fine grid in x, y, and z directions as, $\overline{\boldsymbol{l}^{\mathrm{f,x}}}$, $\overline{\boldsymbol{l}^{\mathrm{f,y}}}$, and $\overline{\boldsymbol{l}^{\mathrm{f,z}}}$, respectively, we define a target coarse grid size as

$$\Delta \equiv 2 \min\left(\overline{\boldsymbol{l}^{\mathrm{f,x}}}, \overline{\boldsymbol{l}^{\mathrm{f,y}}}, \overline{\boldsymbol{l}^{\mathrm{f,z}}}\right). \tag{2.13}$$

Setting the size of the grid to Δ in all directions can lead to refinement in the direction with the largest $\overline{\boldsymbol{l}^{\mathrm{f}}}$. Hence, $\overline{\boldsymbol{l}^{\mathrm{c}}}$ in each direction is determined based on $\max\left(\Delta, \overline{\boldsymbol{l}^{\mathrm{f}}}\right)$. More specifically,

$$\overline{\boldsymbol{l}^{\mathrm{c,x}}} = \max\left(\tilde{\Delta}^{\mathrm{x}}, \overline{\boldsymbol{l}^{\mathrm{f,x}}}\right), \tag{2.14}$$

in which $\tilde{\Delta}^{\mathrm{x}}$ is the nearest value to Δ with $\sum_i l_i^{\mathrm{f,x}}/\tilde{\Delta}^{\mathrm{x}} \in \mathbb{N}$. Equations similar to Eq. (2.14) can also be written for $\overline{\boldsymbol{l}^{\mathrm{f,y}}}$ and $\overline{\boldsymbol{l}^{\mathrm{f,z}}}$.

We next put together all these components to build a multigrid algorithm. The main problem is defined as: given $\boldsymbol{A}$, $\boldsymbol{b}$, and ϵ, find $\boldsymbol{x}$ such that $\|\boldsymbol{Ax}-\boldsymbol{b}\| \leq \epsilon$. First, let $\mathcal{A} = \{\boldsymbol{A}_0, \boldsymbol{A}_1, \cdots \boldsymbol{A}_{m^{\max}}\}$ be a set of matrices in which $\boldsymbol{A}_0 = \boldsymbol{A}$, $\boldsymbol{A}_i$ is the matrix obtained from i consecutive restriction operations (Eq. (2.7)), and $m^{\max}$ is the maximum number of multigrid levels. Note $\boldsymbol{A}_i$ relative to $\boldsymbol{A}_{i+1}$ is analogous to $\boldsymbol{A}^{\mathrm{f}}$ relative to $\boldsymbol{A}^{\mathrm{c}}$ with their relationship fully described above. Additionally, we define function $\mathcal{K}(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, n^{\max})$ to be an iterative solver (the CG for a symmetric and the BCG for a non-symmetric $\tilde{\boldsymbol{A}}$) that returns $\tilde{\boldsymbol{x}}$ as the solution to $\tilde{\boldsymbol{A}}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}$ with a tolerance $\epsilon^{\mathcal{K}}$ and a maximum number of iterations $n^{\max}$. Now, we define a recursive multigrid function $\mathcal{M}\left(\mathcal{A}, \hat{\boldsymbol{b}}, m\right)$ that returns $\hat{\boldsymbol{x}}$ as the solution to $\boldsymbol{A}_m\hat{\boldsymbol{x}} = \hat{\boldsymbol{b}}$ with a tolerance ϵ. With this definition, it is clear that $\boldsymbol{x} = \mathcal{M}(\mathcal{A}, \boldsymbol{b}, 0)$ is the solution to the original problem. In this setting, operator $\mathcal{M}$ is defined in details in Algorithm 1 in Appendix.

Note that the maximum number of iterations of the iterative solver at the coarsest level, when $m = m^{\max}$, is $\tilde{n}^{\max}$ as opposed to $n^{\max}$ with $\tilde{n}^{\max} \gg n^{\max}$. This is to ensure the solution to the coarsest problem converges properly before return. In other words, two later calls to $\mathcal{K}$ are smoothers while the first call is for solving a linear system. Also, note the recursive call to the next level of the multigrid solver inside the while loop. To prevent an infinite loop, one may add a counter that calls an exit command in case of excessive iterations.

3. Partitioning approach

The scalability of an iterative algorithm is primarily a function of its communication overhead and load balance. The communication overhead is directly proportional to granularity of the partitioning, which is the ratio between the number of cells shared between processors to the total number of cells. The load balance relies on an equal distribution of the number of cells between processors. Partitioning the grid to equal-size cubes with an equal number of cells in each direction is an optimal choice with regard to both the communication overhead and load balance. However, this is not possible for a given grid with an arbitrary number of cells in combination with utilization of an arbitrary number of processors. Hence, a compromise must be made between these two factors.

To improve flexibility and increase the number of partitioning combinations, we create a partition-grid † with each of its block having different sizes. This is to obtain an algorithm that remains optimal for a wider combination of grids and number of processors. As a one-dimensional illustration, consider a grid with 10 cells. A partitioning strategy that only allows for equal-size partitions, only accepts the number of processors to be 1, 2, 5, or 10. In three dimensions, such a restriction significantly limits the range of possible partitioning options. With this limitation lifted, the average number of grid points in a direction, denoted by n_i, can be a positive real number, i.e., $n_i \in \mathbb{R}^+ \; \forall i \in \{1, 2, 3\}$, and not necessarily an integer.

† A partition-grid is a rectilinear grid of partitions that is constructed by dividing the computational grid into a certain number of slices in each direction.

To calculate n_i, we first decompose the number of partitions, denoted by n^{p}, to prime numbers

$$n^{\mathrm{p}} = \prod_i p_i^{k_i}, \tag{3.1}$$

in which $\boldsymbol{p} = \{2, 3, 5, 7, 11, 13, 17, 19\}$ is a truncated sequence of prime numbers, and $k_i \in \mathbb{N}$ is the calculated repetition associated with each prime number. Given $\boldsymbol{p}$ and $\boldsymbol{k}$, and the total number of grid points in each direction, $|\boldsymbol{l}|$, $\boldsymbol{n}$ is calculated via Algorithm 2 in Appendix. For example, a $300 \times 200 \times 100$ grid partitioned to $n^{\mathrm{p}} = 30$ will produce $\boldsymbol{n} = \{60, 66.667, 50\}$, viz. partitioning x-direction to 5, y-direction to 3, and z-direction to 2. In y-direction, this leads to slices with a thickness of 67, 67, and 66 grid points, hence committing a negligible load imbalance of 1.5%.

Each of the $m^{\max}$ grids that are generated is separately partitioned according to the algorithm described above. These independent partitions, specifically for a non-uniform grid, can be significantly misaligned at each level of the multigrid method. This has a direct implication on the implementation of $\mathcal{C}$ and $\mathcal{F}$ in terms of processor-to-processor communication. To simplify such implementation, we perform each of those operations in three steps. In the first step, considering $\mathcal{C}$ for example, values from the fine grid are aggregated into a buffer according to Eq. (2.9). Second, the buffer is communicated between processors, and third, it is assigned to the coarse grid. Using a buffer simplifies treatment of boundaries, specifically when multiple cells, each from a different partition, contribute to a single coarse grid cell.

Finally, we note that in the case of large $m^{\max}$ and n^{p}, there can be excessive number of processors for solving a small system at the coarsest grid. To prevent this issue, we impose a lower bound on $\boldsymbol{n}$ (10 in our calculations), hence ensuring that there is always a minimum number of grid points per processor as the grid becomes coarser. Therefore, some processors can be idle when calculations are performed on the coarser grids, yet this has minimal effect on the overall performance due to the negligible cost of coarse grid calculations. Among others, this lower bound depends on the communication latency of the hardware, hence must be calibrated for each cluster. However, in fixing this parameter in all our calculations on various machines, we have not observed a significant change in the overall performance.

4. Results

The parameters of the multigrid method, described in Section 2, are kept fixed for all the results reported in this section. The maximum number of restriction levels is $m^{\max} = 4$. Increasing this number to 3, 5, or 6 has a minimal effect on the results. The tolerance of the solution is $\epsilon = 10^{-7}$. Decreasing or increasing this value will only shift the absolute computational costs, hence our final conclusion is not affected by this value. The maximum number of iterations at the coarsest level is $\tilde{n}^{\max} = 500$. This number is much larger than the actual number of iterations, hence, as long as a large value is used, no effect on the result is expected. The maximum number of iterations of the smoother is $n^{\max} = 8$. The reported results can be moderately affected by this parameter; however, the change in the performance is minimal for $4 \leq n^{\max} \leq 10$. The tolerance of the smoother is $\epsilon^{\mathcal{K}} = 0.15$. This parameter can also moderately affect the results; however, 0.15 is nearly optimal for a wide range of tested cases.

Two sets of cases are studied in this section. The first is the direct numerical simulations of isotropic homogeneous turbulence in a cubic box. In these cases, the grid is uniform in all directions and density (κ in Eq. (2.5)) is constant. These cases test most of the components of the algorithm under simplified conditions. The second is the direct numerical simulation of a radiated particle-laden flow in a duct with square cross section. In these cases, the grid is not homogeneous in y- and z-directions, in which cells near the wall have a much smaller l_i in comparison with those at the duct center. $\max(\boldsymbol{l}^{\mathrm{y}})/\min(\boldsymbol{l}^{\mathrm{y}}) \approx 15$, $\overline{\boldsymbol{l}^{\mathrm{y}}}/\min(\boldsymbol{l}^{\mathrm{y}}) \approx 7$, and $\overline{\boldsymbol{l}^{\mathrm{x}}}/\min(\boldsymbol{l}^{\mathrm{y}}) \approx 25$. The domain is cuboid with an aspect ratio of $6 \times 1 \times 1$. The number of grid points of the finest grid in x-direction is 1.5 times of that of y- or z-direction. For each of these two classes of problem, we ensure that the flow is sufficiently evolved in time such that all wave numbers are present in the RHS of the linear system.

All of the results reported here are obtained from calculations on Titan. Briefly, Titan contains 18,688 compute nodes, each containing a 16-core 2.2GHz AMD Opteron 6274 processor and 32 GB of RAM. Two nodes share a Gemini high-speed interconnect router.

To benchmark our results, we repeated some of the above calculations using the Trilinos software project (Heroux *et al.* 2005). For these cases, the same linear systems employed for testing our algorithm were loaded to the Trilinos. To set up the problem, pre-existing template files, found in the library's documentation, are used. The pre-installed Trilinos library on Titan (version cray-trilinos/11.12.1.3) is used in our calculations. The multigrid and CG solvers from Prokopenko *et al.* (2014) and Bavier *et al.* (2012) packages, respectively, are used for benchmarking our results. The adjustable parameters are tuned based on the recommended criteria. In particular, we set the problem type to Poisson-3D for the multigrid solver, and we followed performance indications for parallel runs. Namely, the Chebyshev polynomial is used as the smoother with a V cycle pattern, as recommended by Trilinos for Poisson linear systems. With these options, as will be shown in the following, the Trilinos multigrid solver scaled poorly on more than 4 nodes (64 processors). We note that these results are obtained based on the best combination of parameters that we were able to find by exploring the parameter space. This does not, however, suggest that there is no combination of parameters that could produce better results.

The direct numerical simulation of isotropic turbulence on a uniform 256^3 grid is performed using the present multigrid method and the results are compared against the CG (Figure 3) and the Trilinos multigrid solver (Figure 4). Based on Figure 3, the presented multigrid method is over an order of magnitude faster than the two implementations of the CG. In a practical sense, performing the same simulation without use of the multigrid technique entails approximately 10 times increase in cost. On a larger grid, this ratio will increase.

At a higher number of processors, communications latency becomes a significant portion of the cost for the multigrid method. This leads to a lower cost ratio at higher number of processors in Figure 3. The higher and lower cost of CG-I in comparison with CG-T at a lower and higher number of processors, respectively, shows that our in-house implementation scales better in comparison with Trilinos's. The scalability of the present multigrid method relies heavily on the scalability of its underlying linear solver, i.e., the CG-I. Therefore, the superior scalability of the CG-I also explains the better scalability of the present multigrid in comparison with that of Trilinos, which has led to a rising trend in Figure 4. Employing the present multigrid instead of Trilinos's for computations with an intermediate number of processors (50 to 100 k cells in each partition) entails approximately 3 times reduction in cost.

The strong scalability of the present multigrid on the same 256^3 grid is shown in

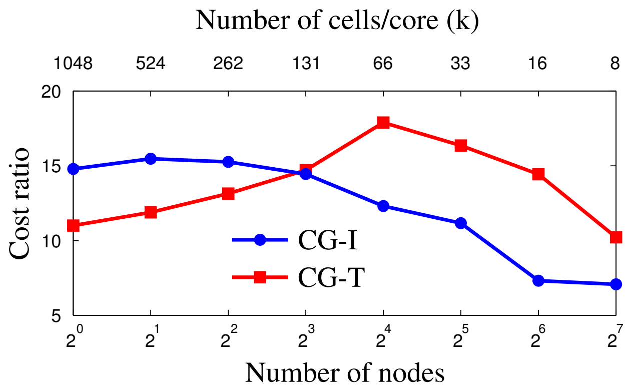

FIGURE 3. The ratio between the cost, defined as the total CPU hours, of CG to multigrid. Our in-house CG is CG-I (blue circles) and Trilinos's CG is CG-T (red squares). Results are related to the isotropic turbulence simulation on a 256^3 grid.

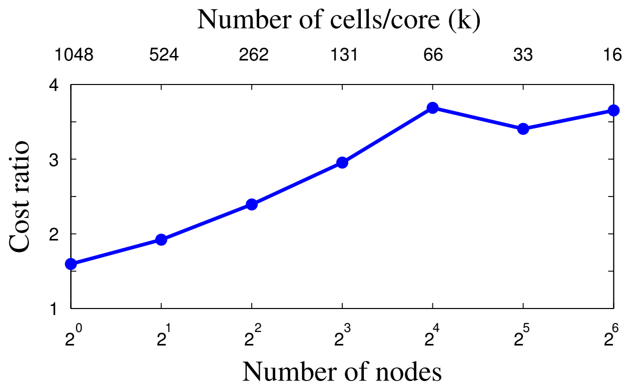

FIGURE 4. The ratio between the cost of the Trilinos multigrid to the present multigrid. Results are related to the isotropic turbulence simulation on a 256^3 grid.

Figure 5. An almost linear speedup is obtained up to 1012 processors, where there are 16,000 cells per partition. Compared to the case with 16 processors, the overall speedup is approximately 38 times.

To show scalability of the present multigrid method on a wider range of grid sizes, we performed a weak scaling study in which the number of cells in each partition is held fixed as the number of partitions is increased. These results, which are also based on the simulations of isotropic turbulence on uniform grids, are presented in Figure 6. Two sets of results are presented in which the number of cells per partition is 25,000 and 50,000. The parallel efficiency of the case with the higher number of cells per partition remains above 60%, while for the other case, it drops to 40% at 4,048 processors. This drop in efficiency is mainly attributable to the latency of collective communications, which increases proportional to the number of processors. However, partitioning the grid with more than 50,000 cells per processor leads to a satisfactory scaling of the problems with size less than a billion grid points.

The grid associated with the simulation run on eight nodes is refined by a factor of two in all directions in comparison with that of a single node in Figure 6. However, the parallel efficiency remains above 90% for the case of 50 k, indicating that the number of iterations is almost independent from the grid size. This independence is the main advantage of the

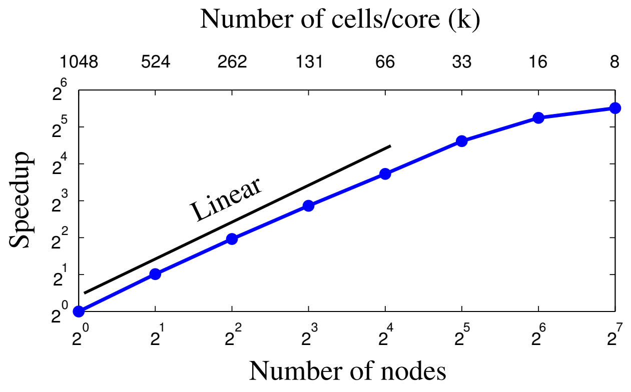

FIGURE 5. Strong scalability of the multigrid on a uniform grid. Results are related to the isotropic turbulence simulation on a 256^3 grid.

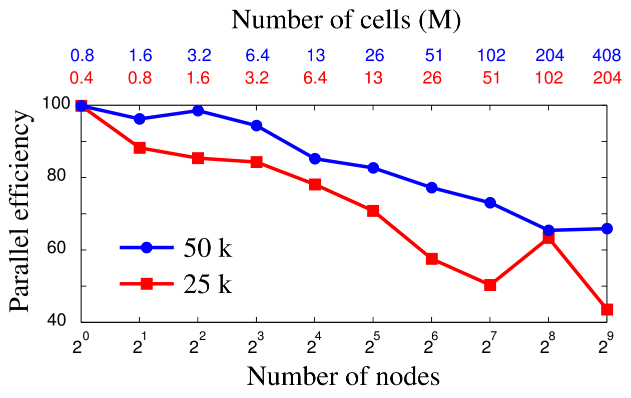

FIGURE 6. Weak scalability efficiency of the multigrid on a uniform grid with 25,000 (red squares) and 50,000 (blue circles) cells per partition. Results are related to the isotropic turbulence simulations.

multigrid in comparison with conventional iterative techniques, in which a degradation in performance is observed as the grid is refined.

The duct flow computation on a $480 \times 320 \times 320$ non-uniform grid is performed and the results are reported in Figure 7. The density variation in the entire domain (i.e., $\max(\kappa)/\min(\kappa)$) is approximately 3. We benchmark the results using the BCG, since the linear system is non-symmetric. For this case with a non-uniform grid, the present multigrid method scales up to 1012 processors (64 nodes), which corresponds to 48,000 cells per partition. At this number of processors, a 40-time speedup is obtained in comparison with a single node simulation.

To establish consistency of these results on various problem sizes, we repeated non-uniform simulations on $120 \times 80 \times 80$ and $240 \times 160 \times 160$ grids. In each case, the number of nodes is scaled proportional to the problem size. The same computations are also carried out using the multigrid method from the Trilinos package and the internal BCG (Table 1).

From Table 1, the present multigrid outperforms Trilinos's multigrid and the BCG by approximately one and two orders of magnitude, respectively. A solve that takes a second with the multigrid takes minutes with the BCG. Moreover, the cost ratio increases as the grid is refined. For Trilinos's multigrid it increases from 3 to 11 or 3.5 to 14 and for

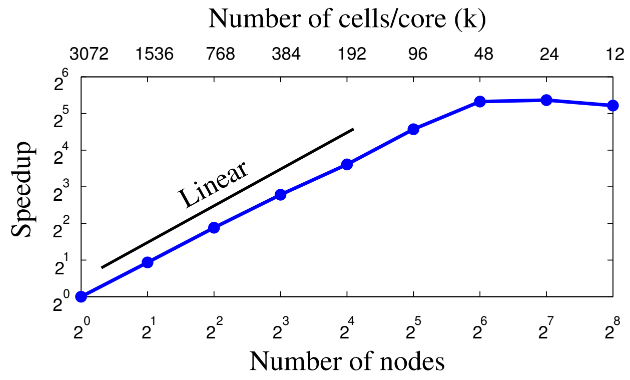

FIGURE 7. Strong scaling of the multigrid solver on a non-uniform grid with variable κ. Results are related to the duct simulation on a $480 \times 320 \times 320$ non-uniform grid.

NND	CPP	Grid	Pres. MG			Trilinos MG			BCG		
			cost	CR	η^{ws}	cost	CR	η^{ws}	cost	CR	η^{ws}
1	48	$120 \times 80 \times 80$	0.94	1	100	2.83	3.0	100	80	85	100
8	48	$240 \times 160 \times 160$	1.25	1	75	8.13	6.5	35	250	200	32
64	48	$480 \times 320 \times 320$	1.37	1	69	15.2	11	19	-	-	-
2	24	$120 \times 80 \times 80$	0.47	1	100	1.65	3.5	86	64	135	63
16	24	$240 \times 160 \times 160$	0.62	1	76	5.08	8.1	28	144	231	28
128	24	$480 \times 320 \times 320$	0.81	1	58	11.7	14	12	-	-	-

TABLE 1. Weak scaling of the present multigrid method (Pres. MG) and its comparison to the Trilinos multigrid (Trilinos MG) and an in-house implementation of the BCG. NND and CPP denote the number of computational nodes and number of cells per partition, respectively. Cost denotes the absolute value of wall-time for solving a single system of linear equations in seconds. The cost ratio, CR, is calculated as the cost normalized by the cost of present multigrid. η^{ws} denotes weak scaling parallel efficiency, using the case run on one node as the reference. The BCG did not converge on the large grids within 20 minutes, hence results are not reported.

the BCG it increases from 85 to 200 or 135 to 231. This is due to the aforementioned ill-conditioning of the system, causing these methods to be more expensive. The cost of the present multigrid method, however, consistent with the results of the uniform grid (Figure 6), remains fairly constant. Additionally, the present multigrid scales well† when the number of cells per partition is decreased from 48,000 to 24,000, showing its excellent weak scalability. This is in comparison with Trilinos's multigrid and the BCG that show approximately 20% average drop in efficiency for the same cases.

5. Conclusions

A geometric multigrid method is presented for solving a non-symmetric system of linear equations that arise from an elliptic PDE operator. This method is constructed

† Except for the largest case in which collective communications hinders the performance.

by directly exploiting the properties of the underlying PDE. An optimal pattern of restriction and interpolations is obtained through a recursive implementation. To improve parallel scalability, a flexible partitioning approach is employed that finds a nearly optimal partitioning, in terms of communication overhead and load balance, for all grids independent of the level.

Taking the multigrid method from the Trilinos library as reference, the present multigrid outperforms it by a factor of approximately 3 on a uniform grid and by up to 14 on non-uniform grids. In comparison with conventional iterative methods, such as the CG and BCG, the present multigrid reduces the cost by orders of magnitude. Tests on a wide range of grid sizes show the robustness of the present multigrid against the grid refinement, i.e., ill-conditioning of the linear system. The scalability studies show remarkable scaling of the present multigrid to over a thousand processors. On uniform and non-uniform grids this translates to 16,000 and 48,000 cells per partition, respectively.

Acknowledgments

This work was supported by the United States Department of Energy under the Predictive Science Academic Alliance Program 2 (PSAAP2) at Stanford University. This research used resources of the OAK Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. We acknowledge the use of computational hours on the Certainty cluster at Stanford University, where all these tools are initially developed.

Appendix A. Detailed description of the algorithms

Algorithm 1. Find $\boldsymbol{x} = \mathcal{M}(\mathcal{A}, \boldsymbol{b}, m)$

```
if $m = m^{\max}$
    $\boldsymbol{x} \leftarrow \mathcal{K}(\boldsymbol{A}_m, \boldsymbol{b}, \tilde{n}^{\max})$
    return $\boldsymbol{x}$

$\boldsymbol{x} \leftarrow \mathcal{K}(\boldsymbol{A}_m, \boldsymbol{b}, n^{\max})$
$\boldsymbol{b}^{\mathrm{f}} \leftarrow \boldsymbol{b} - \boldsymbol{A}_m\boldsymbol{x}$
while $\|\boldsymbol{b}\|^{-1}\|\boldsymbol{b}^{\mathrm{f}}\| > \epsilon$
    $\boldsymbol{b}^{\mathrm{c}} \leftarrow \mathcal{C}(\boldsymbol{b}^{\mathrm{f}})$
    $\boldsymbol{x}^{\mathrm{c}} \leftarrow \mathcal{M}(\mathcal{A}, \boldsymbol{b}^{\mathrm{c}}, m+1)$
    $\boldsymbol{x}^{\mathrm{f}} \leftarrow \mathcal{F}(\boldsymbol{x}^{\mathrm{c}})$
    $\boldsymbol{b}^{\mathrm{f}} \leftarrow \boldsymbol{b}^{\mathrm{f}} - \boldsymbol{A}_m\boldsymbol{x}^{\mathrm{f}}$
    $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{x}^{\mathrm{f}}$
    $\boldsymbol{x}^{\mathrm{f}} \leftarrow \mathcal{K}(\boldsymbol{A}_m, \boldsymbol{b}^{\mathrm{f}}, n^{\max})$
    $\boldsymbol{b}^{\mathrm{f}} \leftarrow \boldsymbol{b}^{\mathrm{f}} - \boldsymbol{A}_m\boldsymbol{x}^{\mathrm{f}}$
    $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{x}^{\mathrm{f}}$

return $\boldsymbol{x}$
```

Algorithm 2. Find average number of partitioned grid points $\boldsymbol{n}$

```
$\boldsymbol{n} \leftarrow \{|\boldsymbol{l}^{\mathrm{x}}|, |\boldsymbol{l}^{\mathrm{y}}|, |\boldsymbol{l}^{\mathrm{z}}|\}$
```

```
do  $i = |\boldsymbol{p}|, \cdots, 1$
    while  $k_i > 0$
        $j \leftarrow \operatorname{argmax}(\boldsymbol{n})$
        $n_j \leftarrow p_i^{-1} n_j$
        $k_i \leftarrow k_i - 1$

return  $\boldsymbol{n}$
```

REFERENCES

Bavier, E., Hoemmen, M., Rajamanickam, S. & Thornquist, H. 2012 Amesos2 and belos: Direct and iterative solvers for large sparse linear systems. *Sci. Program* **20** (3), 241–255.

Esmaily-Moghadam, M., Bazilevs, Y. & Marsden, A. 2015*a* A bi-partitioned iterative algorithm for solving linear systems arising from incompressible flow problems. *Comp. Method Appl. M.* **286** (1), 40–62.

Esmaily-Moghadam, M., Bazilevs, Y. & Marsden, A. 2015*b* Impact of data distribution on the parallel performance of iterative linear solvers with emphasis on CFD of incompressible flows. *Comput. Mech.* **55** (1), 93–103.

Esmaily-Moghadam, M., Bazilevs, Y. & Marsden, A. L. 2013 A new preconditioning technique for implicitly coupled multidomain simulations with applications to hemodynamics. *Comput. Mech.* **52** (5), 1141–1152.

Ghia, U., Ghia, K. N. & Shin, C. 1982 High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *J. Comput. Phys.* **48** (3), 387–411.

Heroux, M. A., Bartlett, R. A., Howle, V. E., Hoekstra, R. J., Hu, J. J., Kolda, T. G., Lehoucq, R. B., Long, K. R., Pawlowski, R. P., Phipps, E. T., Salinger, A. G., Thornquist, H. K., Tuminaro, R. S., Willenbring, J. M., Williams, A. & Stanley, K. S. 2005 An overview of the Trilinos project. *ACM T. Math. Software* **31** (3), 397–423.

Hestenes, M. R. & Stiefel, E. 1952 Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand.* **49**, 409–436.

Jofre, L., Lehmkuhl, O., Ventosa, J., Trias, F. X. & Oliva, A. 2014 Conservation properties of unstructured finite-volume mesh schemes for the Navier-Stokes equations. *Numer. Heat Tr. B-Fund.* **65** (1), 53–79.

Prokopenko, A., Hu, J. J., Wiesner, T. A., Siefert, C. M. & Tuminaro, R. S. 2014 Muelu users guide 1.0. *Tech. Rep.* SAND2014-18874. Sandia National Labs.

Saad, Y. 2003 *Iterative methods for sparse linear systems*. SIAM.

Saad, Y., Sosonkina, M. & Zhang, J. 1998 Domain decomposition and multi-level type techniques for general sparse linear systems. *Contemp. Math.* **218**, 174–190.

Shakib, F., Hughes, T. & Johan, Z. 1989 A multi-element group preconditioned GMRES algorithm for non-symmetric systems arising in finite element analysis. *Comp. Method Appl. M.* **75** (1-3), 415–456.

Vaněk, P., Mandel, J. & Brezina, M. 1996 Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing* **56** (3), 179–196.

Wesseling, P. 1995 Introduction to multigrid methods. *Tech. Rep.* ICASE-95-11. DTIC Document.