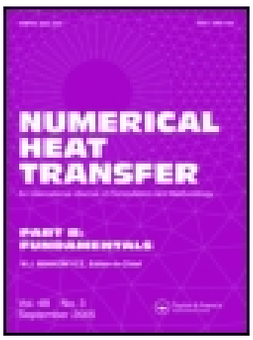


Taylor & Francis
Taylor & Francis Group

Numerical Heat Transfer, Part B: Fundamentals

ISSN: 1040-7790 (Print) 1521-0626 (Online) Journal homepage: http://www.tandfonline.com/loi/unhb20

Parallelization of an Additive Multigrid Solver

M. Darwish , T. Saad & Z. Hamdan

To cite this article: M. Darwish , T. Saad & Z. Hamdan (2008) Parallelization of an Additive Multigrid Solver, Numerical Heat Transfer, Part B: Fundamentals, 54:2, 157-184, DOI: 10.1080/10407790802182638

To link to this article: http://dx.doi.org/10.1080/10407790802182638

Numerical Heat Transfer, Part B, 54: 157–185, 2008

ISSN: 1040-7790 print/1521-0626 online
DOI: 10.1080/10407790802182638

Taylor & Francis
Taylor & Francis Group

PARALLELIZATION OF AN ADDITIVE MULTIGRID SOLVER

M. Darwish[1], T. Saad[1], and Z. Hamdan[2]
[1]*Department of Mechanical Engineering, American University of Beirut, Beirut, Lebanon*
[2]*Department of Civil Engineering, Lebanese University, Tripoli, Lebanon*

This article deals with the implementation and performance analysis of a parallel algebraic multigrid solver (pAMG) for a finite-volume, unstructured computational fluid dynamics (CFD) code. The parallelization of the solver is based on the domain decomposition approach using the single program, multiple data paradigm. The Message Passing Interface library (MPI) is used for communication of data. An ILU(0) iterative solver is used for smoothing the errors arising within each partition at the different grid levels, and a multilevel synchronization across the computational domain partitions is enforced in order to improve the performance of the parallelized multigrid solver. Two synchronization strategies are evaluated. In the first the synchronization is applied across the multigrid levels during the restriction step in addition to the base level, while in the second the synchronization is enforced during the restriction and prolongation steps. The effect of gathering the coefficients across partitions for the coarsest level is also investigated. Tests on grids up to 800,000 elements are conducted for a number of diffusion and advection problems on up to 20 processors.

Results show that synchronization across partitions for multigrid levels plays an essential role in ensuring good scalability. Furthermore, for a large number of partitions, gathering coefficients across partitions is important to ensure a convergence history that is consistent with the sequential solver, thus yielding the same number of iterations for parallel and sequential runs, which is crucial for retaining high scalability. The shadow-to-core elements ratio is also shown to be a good indicator for scalability.

INTRODUCTION

Computational fluid dynamics (CFD) is an essential design tool in many industries (aerospace, automotive, chemical processing, power generation, etc.). At its basic level, CFD involves (1) a discretization step that translates a set of highly nonlinear partial differential equations representing conservation principles (conservation of momentum, mass, energy, etc.) into a sparse system of linearized algebraic equations, (2) a solution step to solve the system of algebraic equations iteratively, iterative solvers are usually used for solving the system of equations (inner loop)

Received 24 February 2008; accepted 21 April 2008.
The authors wish to acknowledge the generous support of the Lebanese National Council for Scientific Research through Grant 022129.
Address correspondence to M. Darwish, Department of Mechanical Engineering, American University of Beirut, P.O. Box 11-0236, Riad El Solh Street, Beirut 1107 2020, Lebanon. E-mail: darwish@aub.edu.lb

NOMENCLATURE

A	left-hand side of system of equations, coefficients of agglomerated element	ϕ	dependent variable
B	right-hand side of agglomerated element	Φ	exact solution
I	multigrid prolongation/restriction operator	ω	face weight for agglomeration algorithm
		Subscripts	
a	coefficient of discretized equations	NB	refers to neighbors of P
b	right-hand side of system of equations	n	refers to neighboring element n
r	residual error, normalized residual error	P	refers to element P
$\mathbf{u}$	velocity vector		
β	proportionality constant	**Superscript**	
Γ	diffusion coefficient	N	refers to coefficient of neighboring element N
ρ	density		

because of their lower computational requirements in memory and CPU time [1]; finally, (3) because of nonlinearities, these two steps need to be performed repeatedly (outer loop) [2, 3] until a final converged solution is reached. For transient problem the whole solution process is repeated as the solution marches in time (transient loop).

With the continuous increase in complexity and size of CFD problems, techniques to accelerate this solution procedure have been the focus of intense effort over the past two decades. Work on resolving the inter-equation coupling (outer loop) of momentum and pressure efficiently, in either a segregated [4, 5] or coupled [6, 7] manner, yielded more robust and more efficient algorithms [8]. Work on the acceleration of the inner loop has led to the development of geometric multigrid methods [9–11], and later to algebraic multigrid methods (AMGs) [12, 13]. The AMGs extended the main idea of geometric multigrid to a purely algebraic setting, yielding robustness and algorithmic simplicity. AMGs can be used to build highly efficient and robust linear solvers [14–18] by combining iterative solvers efficient in damping high-frequency errors with multilevel grids that transform low-frequency errors on fine grids to high-frequency errors on coarser grids. This dual approach has been quite successful in tackling relatively large CFD simulations, but reaches its limits with large-scale simulations, for which the use of parallel processing becomes essential.

Parallel computing has undergone a small revolution of its own over the last decade. The continuously improving floating-point performance of the last few generations of microprocessors, and the availability of continuously cheaper high-speed interconnection networks, has meant that PC clusters (distributed memory) are increasingly being adopted as a cost-effective alternative to classical parallel supercomputers (shared memory) for running large-scale numerical simulations [19–21].

Parallel CFD codes can be written in several ways, depending on the algorithmic implicitness/explicitness, the type of grid used (structured/unstructured, adaptive/nonadaptive), and the degree of coupling that exists or is incorporated into the implemented physical models. Many paradigms can be used; however, when the aim of the parallelization is to use clusters of interconnected processors [22–25], the domain decomposition approach is demonstrably the most effective.

In domain decomposition, the parallelization is enforced by dividing (partitioning) the domain of interest into a number of subdomains or partitions (usually one

for each processor). Each processor is then responsible for solving the computational problem within its subdomain or partition. This is followed by a synchronization phase in which neighboring subdomains swap solution information to ensure consistency in the global solution of the original domain. This is equivalent to performing an interdomain coupling at the outer loop; however, it is not sufficient if high scalability and robustness in performance are to be achieved. To overcome this shortfall, it is essential that an interdomain coupling also be performed at the inner loop or linear solver level.

In the context of parallel multigrid solvers, a number of strategies can be followed to perform suitable synchronizations. At one extreme, synchronization can be performed at the finest mesh only, with the multigrid solver mainly playing the role of its sequential counterpart over the subdomain or partition. At the other extreme, synchronization can be performed at each multigrid level, with the multigrid solver playing an additional role of smoother across partitions but at the expense of additional communication cost between partitions. Other options include synchronization during the restriction phase or in both the restriction/prolongation phases. It is also worth noting that the number of communication messages on coarse meshes is often nearly the same as that on fine meshes, although message lengths are much shorter. However, since most parallel architectures have high communication latencies as compared to current processor speeds, these can end up dominating coarse-grid computations. Furthermore, as the number of partitions increases, the treatment of the coarse level becomes essential from both a robustness and a scalability point of view. If the coarse level has a minimum of 5 elements, a 20-partition parallel run mesh will have 100 elements on the coarsest level as compared to 5 on the sequential run.

This article compares different strategies in building a highly scalable and robust pAMG solver. Different synchronization options are tested, and the large-partitions coarse-grid problem is addressed by migrating the coarsest level in the slave partitions to the master partition for solution using a direct or iterative solver and injecting the results back to the different partitions. Results from solving advection and diffusion test problems of various mesh sizes are presented. The problems were selected to test the scalability and robustness of the pAMG solver.

In the remainder of this article the algebraic multigrid method is presented, with some emphasis on the agglomeration or coarsening algorithm. The domain decomposition (DD) method and interdomain synchronization strategies are then outlined in the context of the AMG. Finally, the different options implemented in the pAMG are outlined, and the algorithms used is described. Results and conclusions drawn across a range of computational meshes and partitions are presented.

THE FINITE-VOLUME METHOD

The finite-volume method (FVM) is a numerical technique aimed at the solution of partial differential equations (PDEs) and especially tuned to those arising in fluid, heat, and mass transfer problems. The general PDE governing the transport of a conserved passive scalar has the following form:

$$\underbrace{\frac{\partial(\rho\phi)}{\partial t}}_{\text{Transient Term}} + \underbrace{\nabla\cdot(\rho\mathbf{u}\phi)}_{\text{Convection Term}} = \underbrace{\nabla\cdot(\Gamma\nabla\phi)}_{\text{Diffusion Term}} + \underbrace{S^{\phi}}_{\text{Source Term}} \qquad (1)$$

where ρ is the density, t is the time, $\mathbf{u}$ is the velocity field, Γ is the diffusivity, and ϕ is the unknown scalar for which a solution is sought. Upon discretizing Eq. (1), our arbitrary control volume, an algebraic equation of the following form is obtained:

$$a_i \phi_i + \sum_{n=\mathrm{nb}(i)} a_i^n \phi_j = b_i \tag{2}$$

where the coefficients depend on the specific schemes used in the discretization process. An iterative solver is then used to solve these equations.

MULTIGRID METHODS

While the standard iterative techniques are quite efficient at solving small and medium-sized meshes, their rate of convergence deteriorates for larger-sized meshes. Brandt [10] studied the intricacies associated with solving a large set of equations and, after decomposing the error in the approximate solution into Fourier components, he found that while many of the iterative solvers are efficient in eliminating high-frequency or oscillatory components of the error, they are inefficient in reducing the remaining low-frequency smooth components of the error. These solvers are said to possess the smoothing property, and are referred to as smoothers.

Multigrid algorithms were originally introduced independently by Federenko [26, 27] (geometric multigrid) and Poussin [28] (algebraic multigrid) in the 1960s, and later gained popularity with the work of Brandt [10, 29]. They are considered to be among the most efficient techniques for the numerical solution of PDEs, at least for sequential computers. While standard iterative solvers (Jacobi, Gauss-Siedel, successive overrelaxation, ILU) are efficient in removing high-frequency errors, they are incompetent in removing the remaining low-frequency or smooth errors [30]. Multigrid methods overcome the decay in the convergence rate by using a hierarchy of coarse grids in addition to the one on which the solution is sought. The fundamental idea is that by restricting the problem to a coarser grid, the lower-frequency errors now appear more oscillatory, i.e., as higher-frequency errors. Applying this restriction recursively, a grid is eventually reached that is coarse enough for the problem to be solved efficiently.

The application of multigrids (MGs) to a given problem follows two stages. In the first stage, the coarse grids and their connectivities are set up using an agglomeration or coarsening algorithm [31–34]. In the second stage, a multigrid cycling procedure is used with a smoother to yield the solution at the finest grid.

Given a system of equations

$$A\Phi = b \tag{3}$$

where A and b are respectively the left-hand side (LHS) and right-hand side (RHS) coefficients defined at the finest grid, and Φ is the exact solution of the system. The error after a number of smoother iterations, i.e., after the high-frequency components of the initial guess are reduced, can be expressed as

$$e^{(1)} = \Phi^{(1)} - \phi^{(1)} \tag{4}$$

where $\phi^{(1)}$ is the iterative solution of Eq. (3), and $e^{(1)}$ is the error of the algebraic approximation (not to be confused with the discretization error). The superscript (1) denotes the finest grid level. Equation (4) can be rewritten as

$$A^{(1)}e^{(1)} = r^{(1)} \tag{5}$$

where

$$r^{(1)} = b^{(1)} - A^{(1)}\phi^{(1)} \tag{6}$$

is the residual of $\phi^{(1)}$. The error $e^{(1)}$ can be computed by continuously iterating on the linear system of grid level (1), but this is computationally very expensive. Instead, the error $e^{(2)}$, an approximation of $e^{(1)}$ defined on a coarser mesh $\Omega^{(2)}$, can be found. $\Omega^{(k)}$ stands for the coarse mesh of level k obtained by some agglomeration of the elements of $\Omega^{(k-1)}$. Hence a new system of equations is derived:

$$A^{(k)}e^{(k)} = b^{(k)} \quad k = 2, \ldots, m \tag{7}$$

where

$$b^{(k)} = I_{k-1}^{(k)} r^{(k-1)} \quad k = 2, \ldots, m \tag{8}$$

$A^{(k)}$ is the approximation to $A^{(k-1)}$ over the coarser mesh $\Omega^{(k)}$, and $I_{k-1}^{(k)}$ is the fine-to-coarse mesh restriction operator.

When Eq. (7) is solved, a correction to $\phi^{(k-1)}$ is obtained as

$$\phi^{(k-1)} \leftarrow \phi^{(k-1)} + I_k^{(k-1)} e^{(k)} \tag{9}$$

where $I_k^{(k-1)}$ is a coarse-to-fine grid prolongation operator. Different types of restriction and prolongation operators can be used. After the correction, starting with $\phi^{(1)}$, ν_2 iterations of the relaxation method are performed. This step is called the postrelaxation (postsmoothing), while the first before the correction is called the prerelaxation (presmoothing). Each juncture of the MG algorithm consists of this three-steps process (prerelaxation, correction, and postrelaxation).

Algebraic MG Correction Equations

The AMG is an efficient method to implement the multigrid algorithm. In the AMG the correction equations [13, 35] are assembled directly from the original equations. This process begins with the expansion of the correction for $\phi^{(k)}$,

$$\phi^{(k)} \leftarrow \phi^{(k)} + I_{k+1}^{(k)} e^{(k+1)} \quad k = 1, \ldots, m-1 \tag{10}$$

where m is the number of multigrid levels. In this case, we do not assume that $e^{(k+1)}$ is the exact solution but only a generic vector with $N^{(k+1)}$ components (while $\phi^{(k)}$ has $N^{(k)}$ components). The goal is to find a suitable expression for $e^{(k+1)}$. Starting with

Eq. (7) for $k = 2$, we have

$$A^{(2)}e^{(2)} = b^{(2)} \tag{11}$$

The correction Eq. (9) thus becomes

$$\phi^{(1)} \leftarrow \phi^{(1)} + I_2^1 e^{(2)} \tag{12}$$

yielding a residual of the form

$$r^{(1)} = b^{(1)} - A^{(1)}\phi^{(1)} - A^{(1)}I_2^1 e^{(2)} \tag{13}$$

To generate the $N^{(2)}$ equations needed to solve for $e^{(2)}$, we require that the restriction of $r^{(1)}$ be null, i.e.,

$$I_1^2 r^{(1)} = 0 \tag{14}$$

That is, the sum of the residual errors over each coarse element is zero. Substituting Eq. (13) into Eq. (14), we obtain

$$I_1^2(b^{(1)} - A^{(1)}\phi^{(1)} - A^{(1)}I_2^1 e^{(2)}) = 0 \tag{15}$$

or

$$I_1^2 A^{(1)} I_2^1 e^{(2)} = I_1^2(b^{(1)} - A^{(1)}\phi^{(1)}) \tag{16}$$

which is equivalent to

$$A^{(2)}e^{(2)} = I_1^2 r^{(1)} \tag{17}$$

where now the coefficient of $A^{(2)}$ is computed as

$$A^{(2)} = I_1^2 A^{(1)} I_2^1 \tag{18}$$

This process is repeated for the other $k = 3$ to m.

Element Agglomeration

Three different approaches can be adopted for the agglomeration or coarsening algorithm. The first approach begins with a coarse mesh definition and generates finer grids by refinement [36, 37]. The main advantage of this approach is that the inter-grid operators become simple because of grid nesting. Another advantage is the possibility of utilizing this setup in an adaptive procedure in which the fine meshes are formed by adaptively refining the coarse meshes [36, 38]. The principal disadvantage is the dependence of the fine grid distribution on the coarse levels. The second approach uses non-nested grids, either with a subset of fine grid points comprising the coarse meshes or with completely independent coarse and fine meshes [39]. In this case the inter-grid information-transfer operators become very expensive

to construct. Furthermore, for both of the above-outlined approaches, generating coarse grids that truly represent complex geometries can be a difficult proposition. In the third approach, essential for the AMG, the coarse grids are generated through agglomeration of the fine-grid control volumes [40, 41]. The agglomeration procedure can be based on a geometric relation between the elements of the grid or on a conditional relation between the mutual coefficients of elements. In the current implementation we follow the latter approach.

To generate the coarse grid sequence, an agglomeration algorithm is used to selectively fuse fine grid elements to form agglomerated or coarse grid elements. This process is repeated until all fine grid elements are fused into coarse elements. The agglomeration process is heuristic, and a number of algorithms can be used. In general, fine grid elements are visited one by one. For a given seed element, a maximum $k_{\max}-1$ of its adjacent elements are fused if they satisfy the agglomeration criterion. If the number of fused elements is less than $k_{\max}-1$, then the neighbors of the fused element are evaluated for fusing until the $k_{\max}-1$ elements have been fused to the seed element, giving rise to a coarse grid element. If a seed element fails to form a coarse element, it is added to the less populated coarse element among its neighbors.

In this work, the directional agglomeration (DA) algorithm of Mavriplis [42] is used. The agglomeration proceeds by fusing available fine grid elements according to the connectivity strength of the elements adjacent to the seed element, and the connectivity strength is governed by the geometry of the elements. This procedure is performed only once at the start of the run.

Directional coarsening. The agglomeration criteria should account for the stretching of the elements in order to form coarser grids that are less stretched than the fine grid ones. The DA algorithm of Mavriplis enforces coarsening in a direction normal to the local element stretching. A weight ω_i^j is assigned to each element i face and coarsening is only applied across faces for which

$$\omega_i^n > \beta\omega_i^{\max} \tag{19}$$

where $\omega_i^{\max}$ is the maximum face weight about the seed element i, and β is typically between 0 and 1, in this case a value of 0.5 was chosen. The face weight recommended by Mavriplis is proportional to the magnitude of the face normal $\mathbf{S}_i^n$. This produces grids that are directionally coarsened in areas of element stretching, and isotropically coarsened in regions of uniform element shape. A second condition was added ensure two-way strong coupling, namely,

$$\omega_i^n > \beta\omega_n^{\max} \tag{20}$$

where now $\omega_n^{\max}$ is the maximum face weight about the seed element n.

Coarsening algorithm. For each Partition,

1. Select a nonagglomerated element on the fine grid as a seed element and set it as the first element in a new coarse element.
2. Visit its neighboring elements. If the neighboring element is not part of a coarse element, check whether Eqs. (19) and (20) are satisfied.

3. If Eqs. (19) and (20) are satisfied, mark the element as part of the current seed coarse element.
4. If the number of elements in the seed coarse element is less than the minimum allowed, visit the neighbors of the newly marked element. If the neighboring element is not agglomerated, check whether Eqs. (19) and (20) are satisfied.
5. If Eqs. (19) and (20) are satisfied, agglomerate the element into the coarse element.
6. Loop over all the fine elements if there are still nonagglomerated elements; try to merge them to any neighboring coarse element if they satisfy Eq. (20).
7. Loop over the fine elements and agglomerate any remaining element to a neighboring coarse element.

The Coarse-Level Coefficients

The coefficients of the coarse element are obtained by summing up the appropriate coefficients of their constituting fine elements. Recalling that the linear equations after discretization have the form

$$a_i \phi_i + \sum_{j=\mathrm{nb}(i)} a_i^j \phi_j = b_i \tag{21}$$

where nb (i) represents the neighbors of element i. This equality is initially not satisfied and gives rise to a residual error defined as

$$r_i = b_i - \left(a_i \phi_i + \sum_{j=\mathrm{nb}(i)} a_i^j \phi_j \right) \tag{22}$$

Assume that the solution on the coarse mesh element $\boldsymbol{I}$ that is parent to the fine mesh element $\boldsymbol{i}$ is $\phi_{I/i}$. The correction on the fine mesh from the coarse mesh is

$$\tilde{e}_i = \phi_{I/i} - \phi_i \tag{23}$$

In order to enforce the residual sum in i to zero,

$$\sum_{i \mathrm{in} I} r_i = 0 \tag{24}$$

The equation on the coarse mesh becomes

$$\sum_{i \mathrm{in} I} a_i \phi_{I/i} + \sum_{i \mathrm{in} I} \sum_{n=\mathrm{nb}(i)} a_i^n \phi_{N/n} = \sum_{i \mathrm{in} I} r_i \tag{25}$$

Rewriting the equation on the coarse mesh yields

$$A_I \phi_I + \sum_{N=\mathrm{NB}(I)} A_I^N \phi_N = B_I \tag{26}$$

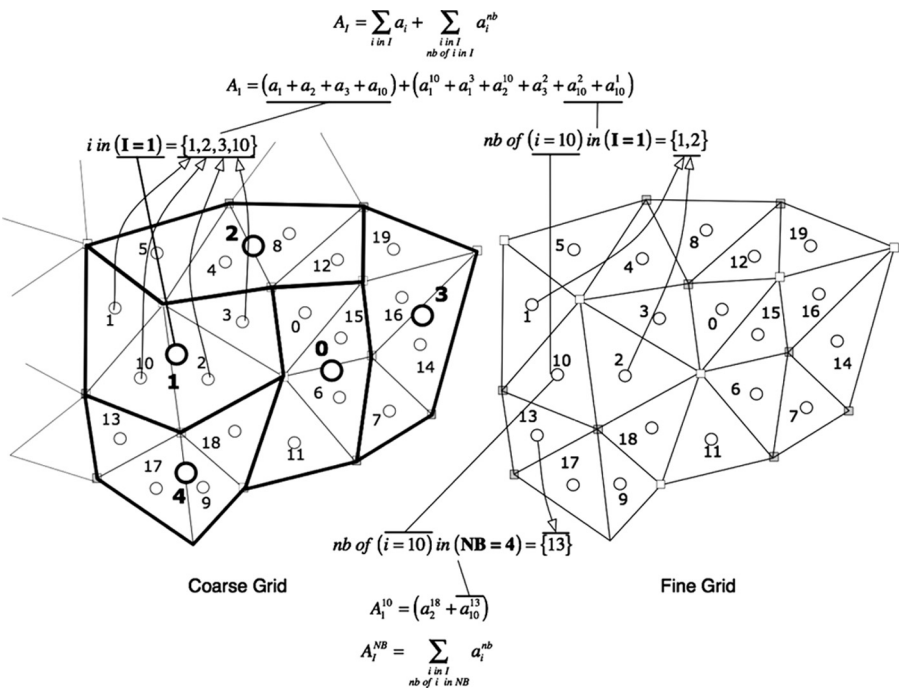


Figure 1. Elements agglomeration.

Then A_I, A_I^N, and B_I are derived directly from fine grid coefficients as

$$A_I = \sum_{i \text{in} I} a_i + \sum_{\substack{i \text{in} I \\ n=\text{nb}(i) in I}} a_i^n$$

$$A_I^N = \sum_{\substack{i \text{in} I \\ n=\text{nb}(i) in N}} a_i^j \tag{27}$$

$$B_I = \sum_{i \text{in} I} r_i \tag{28}$$

The assembly of the coarse-level coefficients is illustrated in Figure 1. The chosen prolongation operator is of order zero, thus the corrections of the coarse elements are injected directly into their constituting finer elements.

Parallelization adds another level of complexity to the agglomeration process, since the agglomeration process in each partition needs to produce coarse elements with associated halo elements to enable synchronization across partitions.

AMG Cycles

A multigrid cycle refers to the way by which coarse grids are visited during the solution process. Two categories of cycling methods in multigrid can be defined: fixed and flexible cycles [43]. While three fixed V, F, and W cycles were implemented,

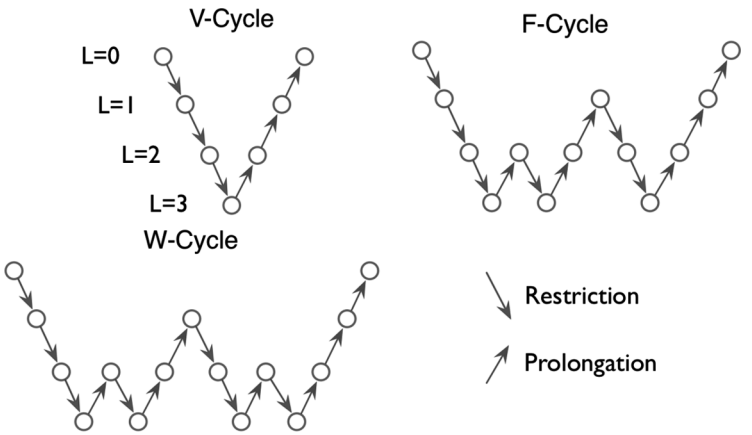


Figure 2. AMG cycles.

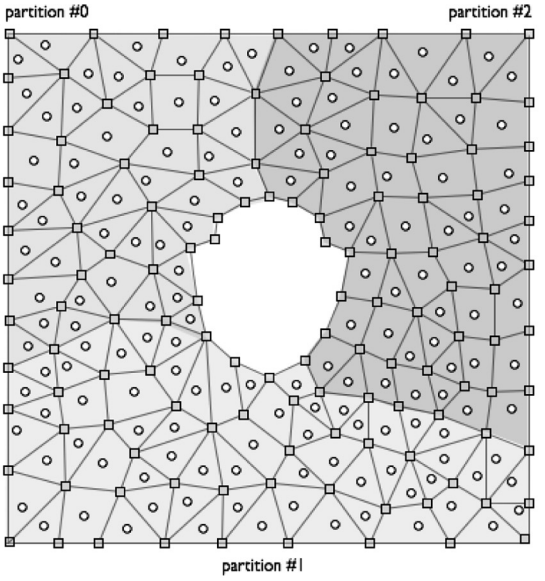
the results discussed in this article were obtained using a V cycle (see Figure 2). For all three cycles, each of the coarse grids is visited successively. During restriction, the solver performs two sweeps on fine levels and 10 sweeps on the coarsest level during prolongation; the solver performs only one sweep on all visited levels.

Parallel CFD

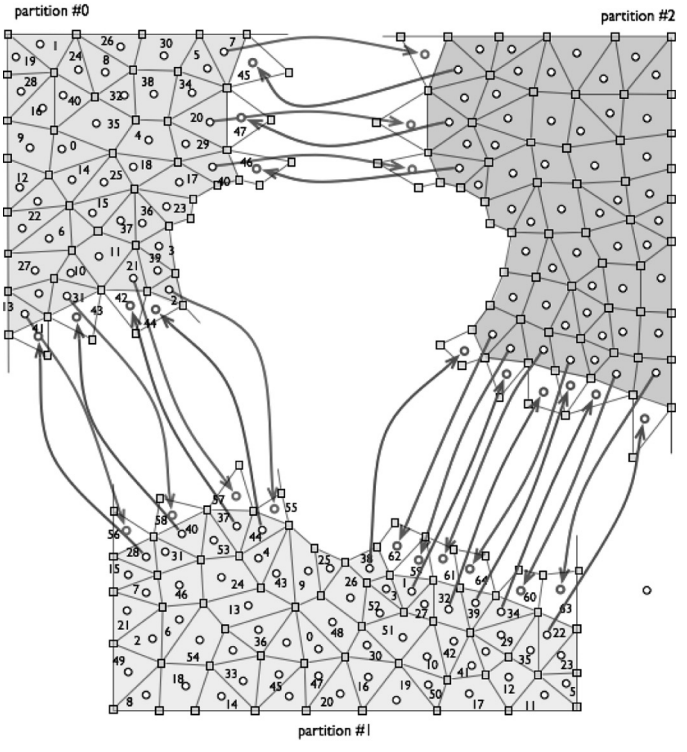
Domain decomposition following the SPMD (Single Program Multiple Data) [44] programming model present a natural parallelization strategy for CFD codes. This approach, usually called *coarse grain* parallelization, is more suitable when using a cluster of personal computers for parallel computations [45, 46]. In this approach the computational domain (Figure 3*a*) is decomposed into a set of subdomains or partitions (Figure 3*b*). Each of these partitions is distributed along with the problem definition to the different processors where the same program is run in a sequential manner but works on a different part of the original data, while allowing for synchronization and data exchange across partitions as needed. The process thus begins with the partitioning of the computational domain, followed by an iteration of discretization and solution phases on the respective processors, along with inter-partition synchronization. Finally, the solution fields on the different partitions are sent to the main processor for reconstruction on the original mesh and saved for further manipulation. The main steps are detailed below.

Mesh Partitioning

METIS [47] was used for the partitioning of the computational mesh because it combines flexibility and good load-balancing properties, in addition to optimizing communication across partitions by minimizing partition boundaries [48]. The output from METIS is a list of all elements with their assigned partition number. These elements form the basis of the computational mesh of the partitions and are denoted



(*a*)



(*b*)

Figure 3. (*a*) Mesh partitions. (*b*) Partitions renumbering.

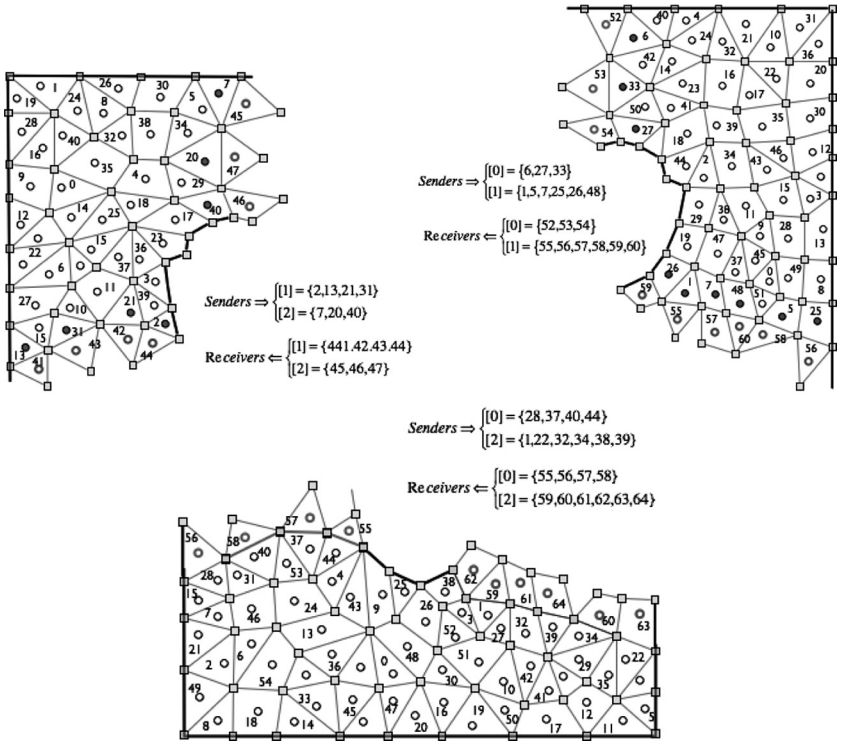


Figure 4. Senders and receivers lists.

as core elements. Core elements at the boundary of a partition can have neighboring elements that are part of another partition; these neighboring elements are denoted as shadow elements and are added to the list of elements in each partition. A low shadow-to-core element count in each partition is essential to minimize interpartition communication during the solution phase.

In the process of renumbering the partition entities, and in order to keep the coupling (and therefore enforce conservation) between the various partitions, shadow elements at the interface of each partition have to be correctly defined as depicted in Figure 4. The shadow elements are numbered sequentially according to their core partition and are placed after the core elements. To expedite interpartition synchronization, a list of shadow and sender elements is stored within each partition (Figure 4). Sender elements are defined as the elements that update the values of shadow elements in neighboring partitions. For example, as Figure 4 shows, element 5 in partition #2 (see 2nd element in senders list of partition #2 to partition #1) updates shadow element 60 in partition #1 (see receivers list in partition #1 from partition #2) . The values of the shadow elements are updated from their associated core elements and stored in the partition for later use. This is essential in order to avoid excessive interpartition communication during the discretization phase. This is somewhat similar to treating these shadow elements as a continuously updated Dirichlet condition.

Core and shadow numbering algorithm. For each partition,

1. Form a list of core elements.
2. Loop over neighbors of all core elements and check whether they belong to the current partition. If not, add element number to the shadow list with its partition number.
3. Loop over shadow elements and renumber them so that shadow elements belonging to the same partition are arranged consecutively.
4. Form the senders and receivers list, and send them to their respective partitions.

Partition Agglomeration Strategies

If the agglomerated grids are not synchronized during the solution process (across partitions), the benefits of multigridding are lost with the increase in number of partitions, as the solution will proceed from partition to partition in an explicit manner. A better approach is to synchronize at the multigrid levels.

Two approaches can be followed for the parallelization of the multigrid coarsening scheme described previously, and these result in a degradation of performance with increased partitions, thus decreasing any potential for scalability. In the first approach, the coarsening takes place on the original mesh. The agglomerated grids are constructed, then the fine grid is partitioned and the agglomerated elements are associated with the respective partitions and distributed to the different processors. The problem in this approach occurs when elements are agglomerated across partitions; in this case, special treatment is needed for these elements that can degrade the performance of the multigrid solver. The agglomeration can be forced to occur on a partition basis, but then, in this case, following the second approach becomes more appropriate. In the second approach, the agglomeration is performed on the core elements of each partition and starts after the partitioning step, and is performed in parallel in the different partitions. This is repeated until the coarsest grid level is reached [49]. At each level, partitions exchange information at the interface regarding the agglomerated shadow and sender elements. This information is determined from the agglomerated core elements.

Coupled coarsening algorithm. For each partition,

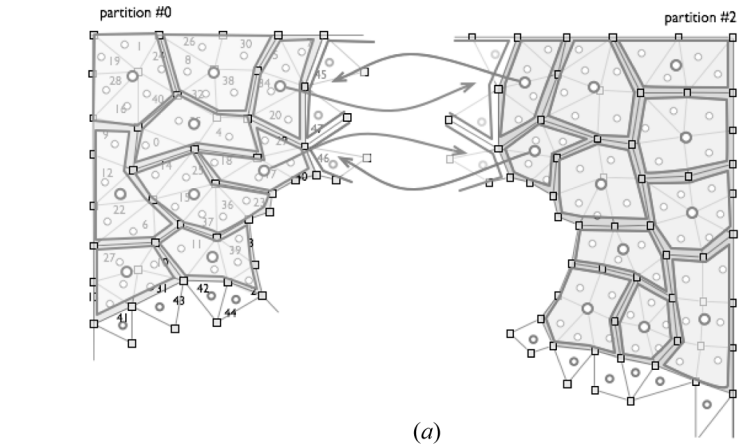
1. Agglomerate core elements and form coarser grid.
2. Send list of sender elements parents to adjacent partitions.
3. Receive shadow element parents list from adjacent partitions.
4. Complete the agglomeration of shadow elements based on their core parents.
5. Send number of elements in coarse grid to master and receive control info on whether to continue or stop agglomeration.

Multigrid Synchronization

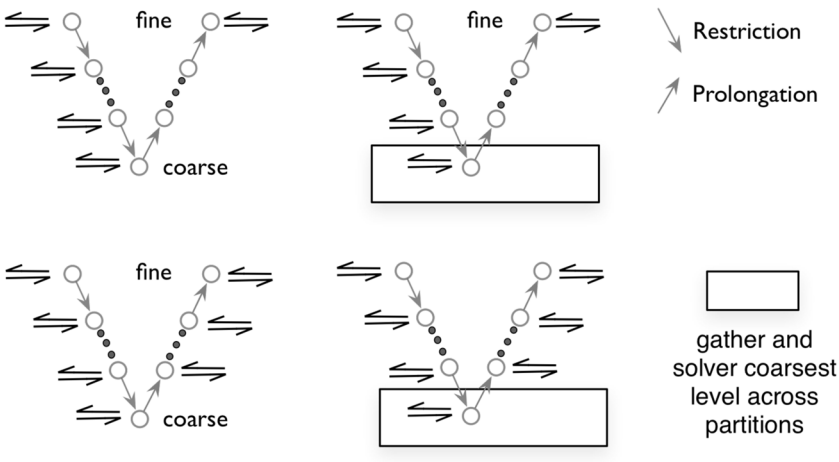
At each multigrid level, the shadow elements are synchronized with their core elements as illustrated in Figure 5*a*. It is important to remember that if the benefits of synchronization do not offset its cost, then the performance of the pAMG will suffer. Two strategies were tested. In the first, the synchronization is performed during the

restriction step. This is essential to ensure that the equations solved at each level represent the whole problem across the different partitions. The synchronization can also be performed during the prolongation steps; this will ensure that some nonlinearities are accounted for during the solution procedure. The two approaches are illustrated in Figure 5*b*.

Coarsest grid synchronization. The solver on the coarsest grid can limit the ultimate speed-up that can be achieved in a parallel computation for two related reasons. First, the linear system at this level is generally small and the time required for communication may be higher than the time required to solve the system on a single processor. Second, the coarsest grid may couple all pieces of the global problem, and



(*a*)



(*b*)

Figure 5. (*a*) Synchronization across partitions. (*b*) Synchronization strategies.

thus an accurate solution at this level is important, as is the global communication of the right-hand side.

If the coarse grid is small enough, instead of solving in parallel, the coarsest grid problem may factored and solved on a single processor with the right-hand side gathered and solution scattered to the other processors. This is illustrated in Figure 6, where coefficients from the various partitions are gathered and solved on the master partition. Another option would be to solve the coarsest grid problem on all processors. This redundant form of the calculation does not require communication to distribute the results [50], at the cost of gathering all coefficients in all partitions.

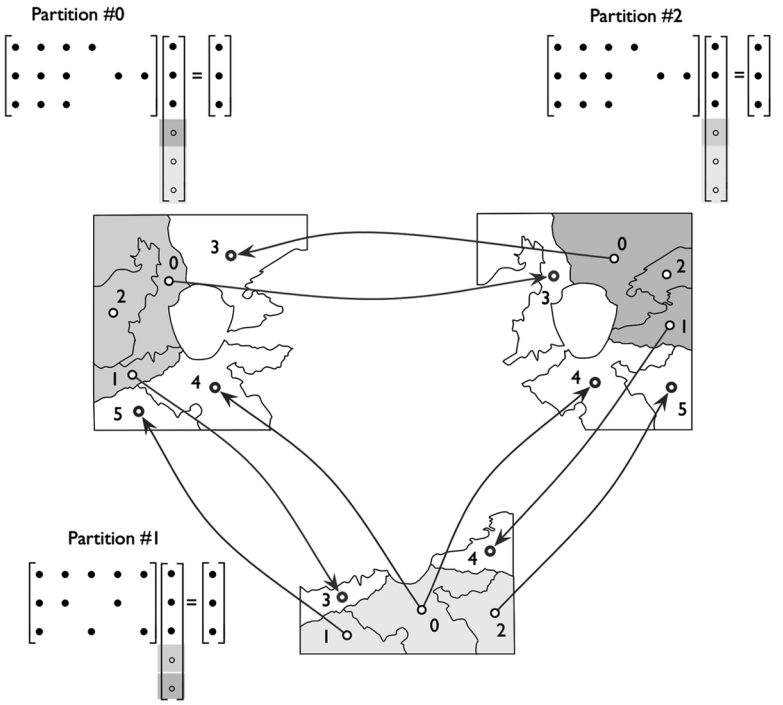


Figure 6. (*a*) Coefficients in partitions. (*b*) Gathered coefficients.

Table 1. Algorithms for interpartition synchronization

Algorithm A	Synch multigrid levels during restriction (2 iterations) and prolongation (1 iteration) For coarsest levels, assemble all coefficients on master node for interpartition solution using direct solver
Algorithm B	Synch multigrid levels during restriction (2 iterations) and prolongation (1 iteration) For coarsest levels, 5 iterations
Algorithm C	Synch during restriction only For coarsest levels 5 iterations

Gathering top-level grids across the partitions algorithm. For each partition,

1. Proceed as in the coupled coarsening algorithm, then, during each iterative solution, ...
2. Send top-level grid coefficients with senders and receivers lists to master partition.
3. Renumber core elements for all the gathered partitions in the same order as the partition number.
4. Replace shadow elements with their respective core elements.
5. Solve top-level grid on master partition.
6. Send core elements solution to respective partitions.
7. Update shadow elements across partitions.

To evaluate the various strategies presented for synching across partitions, three algorithms presented in Table 1 were tested. In algorithm A, synching across partitions takes place during both the restriction and prolongation steps, and the coarsest level is solved on the master partition using a direct solver. In algorithm B, the synching strategy adopted is similar to algorithm A except that no special treatment is applied at the coarsest level except that the number of iterations of the smoother is increased to 5. For algorithm C, synching across partitions takes place only during restriction. This last algorithm was found to be relatively instable in the second test problem and was not used.

TEST PROBLEMS

In all tests, parallel runs on 2, 4, 6, 8, 10, 12, 14, 16, 18, and 20 partitions were performed for all grids on a cluster of 10 G5 dual-processor machines running at 2.0 GHz each with a memory of 512 MB per node, except for the master node with 2 GB.

The interconnection network is a 100-MBits Ethernet switch. The network is not a dedicated one but rather part of a local area network (LAN), which led to some variations in the different runs.

The runs were made until the residual was reduced to a value of 10^{-6}, where the residual is defined as

$$r_P = \frac{b_P - \sum_{n=\mathrm{NB}(P)} a_n \phi_n}{a_P} \tag{29}$$

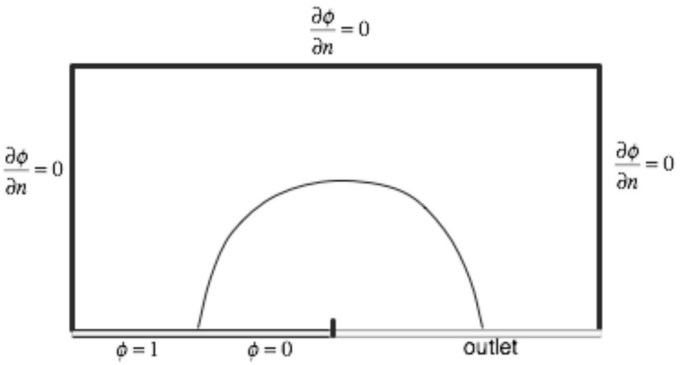


Figure 7. Test 1, advection of a step profile in a rotational flow field.

Test 1: Advection of a Step Profile in a Rotational Flow Field

The first test, illustrated in Figure 7, is the well-known Smith and Hutton test problem [51], in which a step profile is advected in a rotational flow field. A first-order upwind scheme was used to solve the problem on five computational grids with triangular elements. The relation between the computational meshes, the number of partitions, and the number of multigrid levels (MGL) reached for each mesh partition combination and the total number of coarsest grid elements across partitions (CGE) is show in Table 2, while the shadow-to-core elements ratio is shown in Table 3 for the various partitions of the computational meshes.

Figure 8 shows the speed up obtained for the four meshes using up to 16 processors. For the 99,454-elements mesh, the scalability starts deteriorating as of 4 processors. For the 199,222-elements mesh, the behavior is linear up to 14

Table 2. Multigrid levels (MGL) and coarsest grid elements (CGE) for test 1 meshes

	Computational mesh									
	100,000		200,000		400,000		600,000		800,000	
Partitions	MGL	CGE	MGL	CGE	MGL	CGE	MGL	CGE	MGL	CGE
1	8	2	8	4	9	2	9	3	9	3
2	8	2	8	4	9	3	9	3	9	2
4	7	11	8	4	8	11	8	13	9	9
6	7	8	7	16	9	11	8	16	9	11
8	7	9	7	16	8	12	8	13	8	20
10	7	17	7	19	8	14	8	17	8	22
12	7	19	7	18	8	17	8	19	8	33
14	6	34	7	15	7	37	**8**	**26**	8	19
16	6	37	7	20	7	40	**8**	**26**	8	25
18	6	33	7	21	7	35	**7**	**52**	8	33
20	6	35	7	23	7	47	**7**	**53**		

Table 3. Shadow-to-core elements ratios for test 1 meshes and partitions

	Number of control volumes				
Partitions	100,000 (%)	200,000 (%)	400,000 (%)	600,000 (%)	800,000 (%)
1	0.34	0.26	0.18	0.14	0.12
2	1.13	1.00	0.69	0.48	0.48
4	1.96	1.45	0.87	0.65	0.69
6	2.23	1.46	1.16	0.90	0.80
8	2.90	2.05	1.45	1.33	1.04
10	3.30	2.46	1.60	1.43	1.22
12	3.55	2.80	1.76	1.59	1.29
14	4.09	2.66	2.11	1.62	1.37
16	3.87	2.95	2.03	1.71	1.48
18	4.13	3.16	2.33	1.97	1.63
20	0.34	0.26	0.18	0.14	0.12

processors. The scalability starts to deteriorate when the ratio of the number of shadow elements to the number of core elements (shadow-to-core ratio hereafter) exceeds 0.02. This is shown in Figure 9, which shows the efficiency versus the shadow-to-core ratio. For the 298,154-elements mesh, it is noticed that the behavior is superlinear up to 16 processors at which the shadow-to-core ratio remains almost the same but the communication cost increases. The 397,393-elements mesh displays the same behavior as the latter mesh.

The reason why large meshes have better scalability is that the shadow-to-core ratio is less than that for smaller meshes, leading to a lower communication-to-computational time ratio, which accounts for the improved performance. The number of iterations to convergence for the various partitions and computational meshes is shown in Table 4. It is shown to increase with the mesh size, but for all three algorithms the number does not increase with the number of partitions.

Test 2: Diffusion of a Scalar with Various Diffusion Coefficient Ratios

In the second test, three diffusion problem illustrated in Figure 10 are solved. The diffusion coefficient ratio varies for cases (i) 1, (ii) 10, and (iii) 100 for the three problems. The importance of interpartition coupling increases with the diffusion ratios, so the first test is used to evaluate the performance of the coefficient assembly techniques. The three problems were solved on grids with 100,000, 300,000, 500,00, and 800,000 elements (control volumes). Only algorithms A and B were used in the solution, as algorithm C was found to be very unstable.

Figure 11 shows the speed-up obtained for the four meshes using up to 20 processors. For the 100,000- and 300,000-elements meshes, the scalability degrades after 6–8 partitions, partly due to the large shadow-to-core ratios, shown in Table 5. The performance seems to improve as we move from case (i) to case (ii) to case (iii) with algorithm A, due to the increased computational time needed to solve the problem sequentially as the diffusion ratio increases. With algorithm B we also see an improvement in the performance of the solver, except for the fours runs in cases

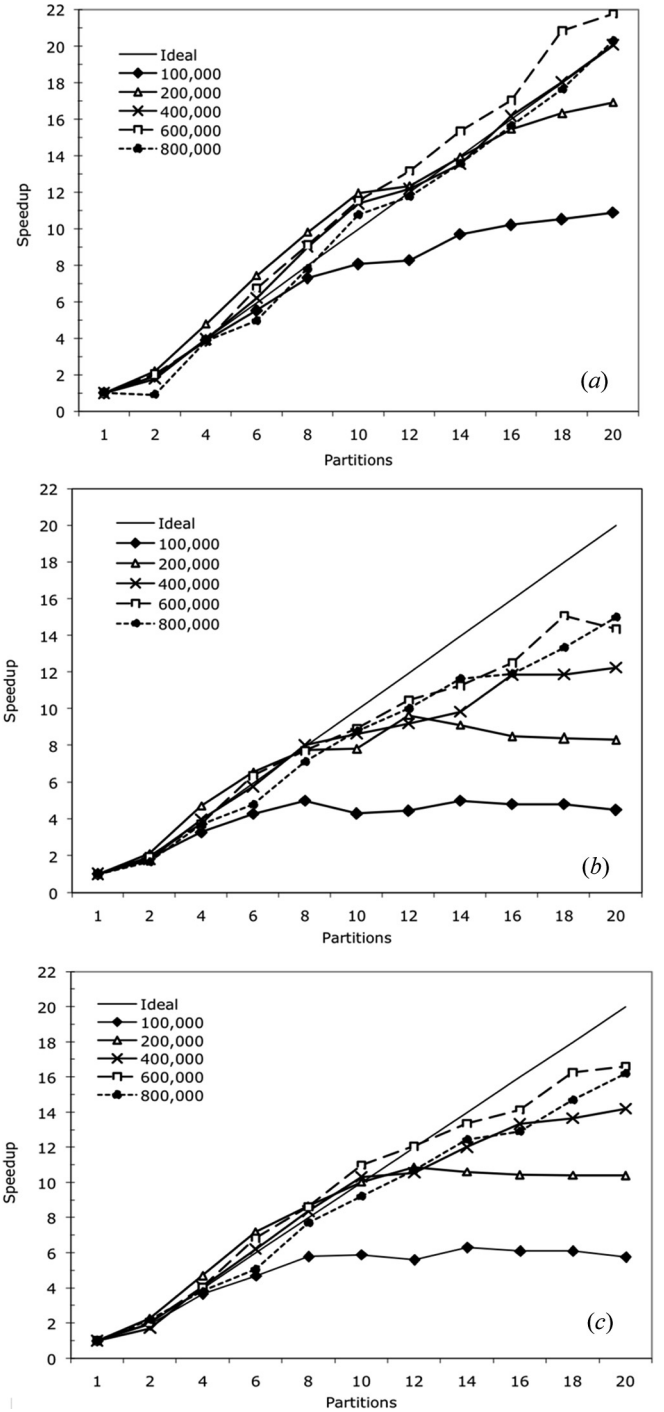


Figure 8. Speed-up for test 1 using (*a*) algorithm A, (*b*) algorithm B, and (*c*) algorithm C.

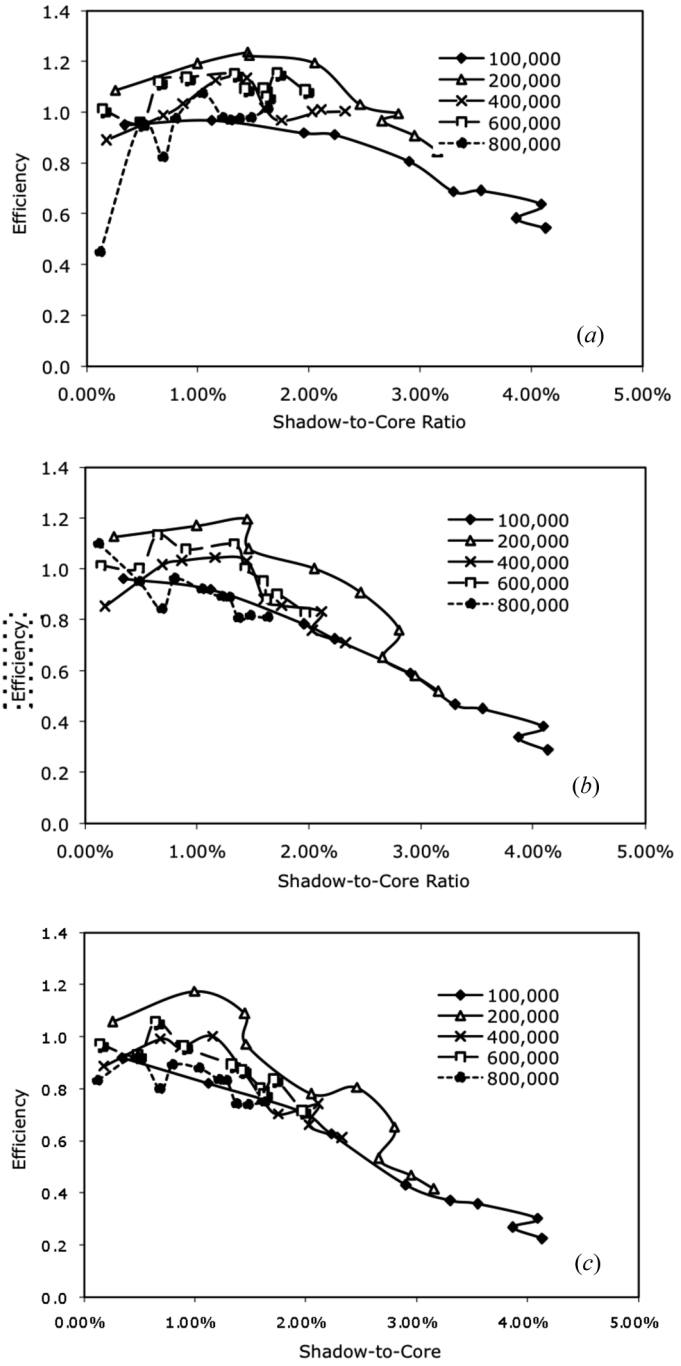


Figure 9. Efficiency versus shadow-to-core ratio for test 1 using (*a*) algorithm A, (*b*) algorithm B, and (*c*) algorithm C.

Table 4. Number of iterations to convergence for test 1 meshes

	Computational mesh														
	100,000			200,000			400,000			600,000			800,000		
Partitions	S1	S2	S3	S1	S2	S3	S1	S2	S3	S1	S2	S3	S1	S2	S3
1	39	39	39	51	51	51	52	52	52	62	62	62	67	67	67
2	39	39	39	51	51	51	52	52	52	62	62	62	67	NC	67
4	39	39	39	51	51	51	52	52	52	60	60	62	67	67	67
6	39	39	39	51	51	51	52	52	52	62	62	62	67	67	67
8	39	39	39	51	51	51	52	52	52	62	62	62	67	67	67
10	39	39	39	51	51	51	52	52	52	62	62	62	67	NC	67
12	39	39	39	51	51	51	NC	52	52	62	62	62	67	67	67
14	39	39	39	NC	NC	NC	52	NC	52	62	62	62	67	67	67
16	39	39	39	51	49	51	52	52	52	62	62	62	67	67	67
18	39	39	39	51	51	NC	52	52	52	62	62	62	67	67	67
20	39	39	39	51	51	51	52	52	52	62	62	62	NC	67	67

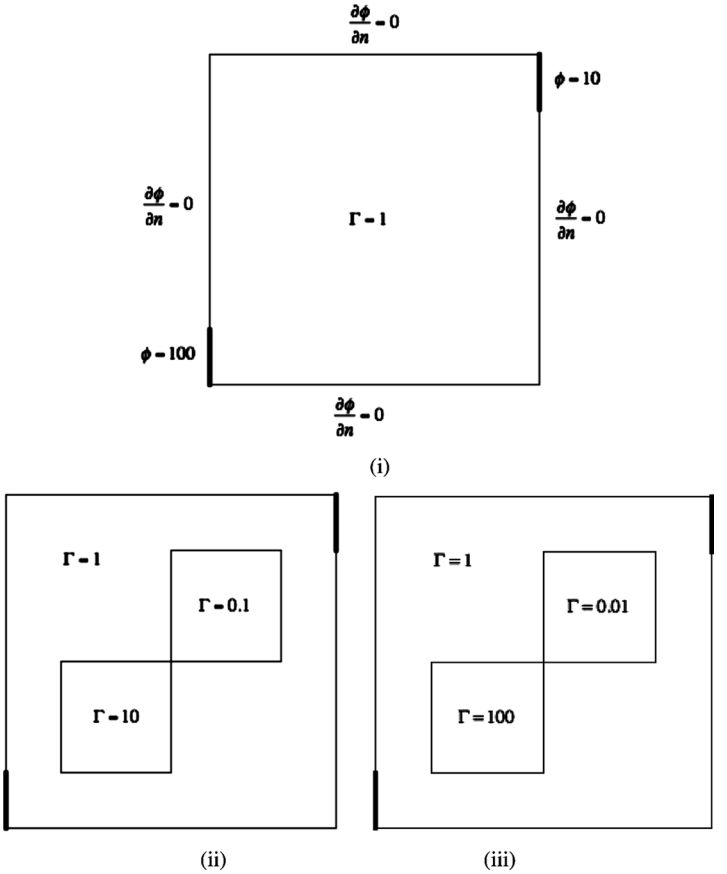


Figure 10. Test 2 cases, diffusion of a scalar field with (i) uniform diffusion coefficients, (ii) coefficients ratio of 10, and (iii) coefficients ratio of 100.

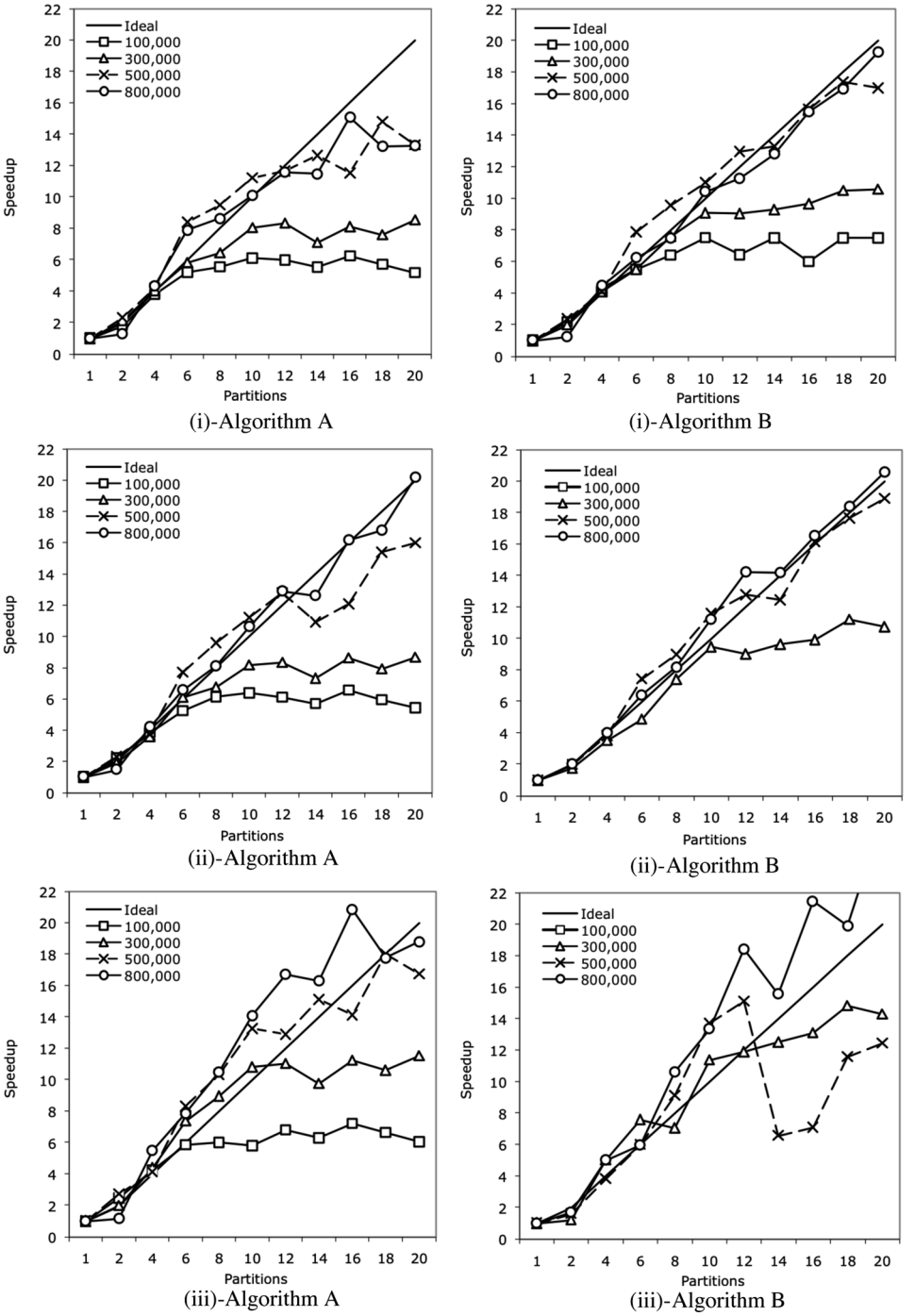


Figure 11. Speed-up graphs for test 2 cases (i), (ii), and (iii) with algorithm A and algorithm B.

Table 5. Shadow-to-core elements ratios for test 2 meshes and partitions

	Number of control volumes			
Partitions	100,000 (%)	300,000 (%)	500,000 (%)	800,000 (%)
1	0.00	0.00	0.00	0.00
2	0.33	0.39	0.31	0.26
4	1.34	0.85	0.70	0.55
6	2.13	1.63	1.43	0.99
8	**2.33**	2.03	1.78	1.37
10	2.94	**2.66**	1.87	1.56
12	3.24	2.72	2.25	1.73
14	3.88	3.08	2.32	1.88
16	4.21	3.33	**2.57**	1.98
18	4.00	3.40	2.72	2.25
20	4.36	4.02	2.78	**2.26**

(iii) for mesh 500,000 and partitions 14, 16, 18, and 20. A look at Table 6 explains this behavior. The table lists for each computational mesh and partition number the multigrid level (MGL) used across the partitions and the coarse grid elements (CGE), which is the total number of core elements across all the partitions; this last number is related to the number of MGL reached for the respective meshes and partitions. This number is computed based on the agglomeration algorithm. For the 500,000 mesh and partitions 14, 16, 18, and 20, the total number of core elements at the coarsest level is around 40, since only 7 MGL are used, in contrast to the 800,000 computational mesh, where an extra multigrid level is used to lower the CGE number. As the CGE number increases, we expect a degradation of performance with algorithm B. This is clear in Table 7 where for algorithm B the number of iterations to convergence increases with partition number for various meshes and cases.

On the other hand, since a direct solver is used in algorithm A at the master node for the assembled coefficients, a large number of coarsest grid elements leads

Table 6. Multigrid levels (MGL) and coarsest grid elements (CGE) for test 2 meshes

	Computational mesh							
	100,000		300,000		500,000		800,000	
Partitions	MGL	CGE	MGL	CGE	MGL	CGE	MGL	CGE
1	8	5	8	5	9	3	9	4
2	8	4	8	5	9	3	9	4
4	8	7	8	8	8	10	9	4
6	8	9	8	6	8	11	8	14
8	8	18	7	23	8	11	8	18
10	7	26	7	21	8	11	8	16
12	7	25	7	24	8	12	8	13
14	7	26	7	24	**7**	**41**	8	16
16	7	37	7	23	**7**	**39**	8	17
18	7	24	7	23	**7**	**39**	8	19
20	7	30	7	21	**7**	**40**	8	20

to a performance penalty. This could be resolved by using an iterative solver for the assembled coefficients.

Figure 12 shows the graph of efficiency versus shadow-to-core elements ratio, where it is clear that beyond a certain shadow-to-core ratio, the performance of the

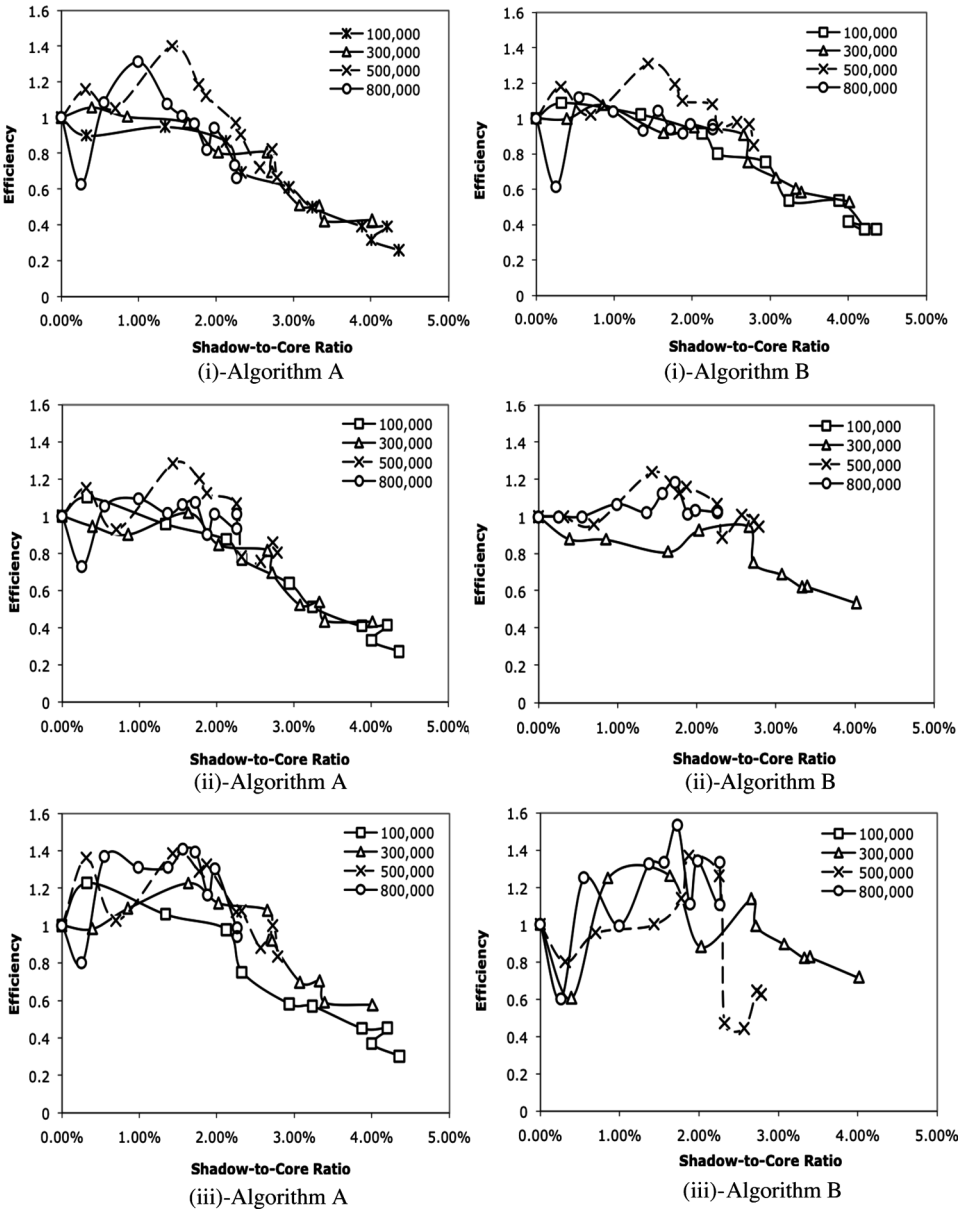


Figure 12. Efficiency versus shadow-to-core elements ratio for test 2 cases (i), (ii), and (iii) with algorithm A and algorithm B.

Table 7. Number of iterations to convergence for test 2 cases and meshes

	Number of control volumes							
	100,000		300,000		500,000		800,000	
Partitions	Algo. A	Algo. B	Algo. A	Algo. B	Algo. A	Algo. B	Algo. A	Algo. B
3 Case (i)								
1	14	14	14	14	15	15	16	16
2	14	14	11	12	14	14	15	15
4	14	14	10	10	13	14	13	13
6	14	50	10	11	10	11	10	13
8	14	50	10	10	10	11	11	14
10	14	14	10	10	10	NC	10	11
12	14	16	10	10	10	10	10	12
14	14	14	10	10	10	11	11	12
16	14	20	10	10	10	10	10	10
18	14	50	10	10	10	10	10	10
20	14	NC	10	10	10	10	10	10
Case (ii)								
1	15	16	16	20	17	17	17	18
2	14	70	11	12	14	70	17	17
4	14	70	11	12	16	16	14	70
6	14	70	10	13	11	12	12	13
8	14	70	10	10	10	12	12	13
10	14	70	10	10	10	10	10	11
12	14	NC	10	10	10	10	10	10
14	14	14	10	10	10	10	10	11
16	14	70	10	10	10	10	10	10
18	14	70	10	10	10	10	10	10
20	14	70	10	10	10	10	10	10
Case (iii)								
1	17	33	21	71	20	22	22	23
2	14	100	14	23	14	NC	18	41
4	14	100	12	NC	17	19	14	NC
6	14	100	11	11	12	NC	13	18
8	33	100	10	14	11	14	12	13
10	17	100	10	11	10	10	10	12
12	14	NC	10	10	10	10	10	10
14	14	14	10	NC	10	27	10	13
16	14	NC	10	10	10	27	10	10
18	14	19	10	10	10	18	10	12
20	14	14	10	10	10	18	10	10

parallel solver degrades. This is to be expected, as the communication cost increases with the number of shadow elements.

It can also be noted that for large meshes, there is a drop that occurs in performance for partition number of 2; this was linked to memory constraints on the slave nodes. For a two-partition, 800,000 mesh, each partition will have nearly 400,000 elements, which strains the memory of the slave nodes, leading to excessive use of virtual memory.

Using algorithm C resulted in divergence in all of the problems of test 2.

CONCLUSION

In this article, various algorithms that are the basis of parallel AMG solvers were explained, and the performance of one implementation pAMG solver used in a finite-volume element-centered unstructured CFD code was presented. The performance of the parallel solver was shown to scale linearly for relatively large meshes with a shadow-to-core ratio above 2.33%; this was clearly illustrated using a graph of efficiency versus shadow-to-core. A technique for assembling the coarsest grid coefficients at the main node was shown to generally improve the robustness of the solver. The cost of the assembly is relatively high, and therefore it should be used only for larger-mesh problems. The solver behaves extremely well for a range of processors. Future work will focus on optimizing the assembly process and solution process of coarsest-grid-level coefficients.

REFERENCES

1. H. Martin Bucker, Iteratively Solving Large Sparse Linear Systems on Parallel Computers, *NIC Services, John Von Neumann Institute for Computing, Julich*, vol. 10, pp. 521–548, 2002.
2. S. V. Patankar, *Numerical Heat Transfer and Fluid Flow*, Hemisphere, McGraw-Hill, Washington, DC, New York, 1980.
3. J. H. Ferziger and M. Peric, *Computational Methods for Fluid Dynamics*, Springer-Verlag, Berlin, New York, 1999.
4. D. S. Jang, R. Jetli, and S. Acharya, Comparison of the PISO, SIMPLER, AND SIMPLEC Algorithms for the Treatment of the Pressure-Velocity Coupling in Steady Flow Problems, *Numer. Heat Transfer.*, vol. 10, pp. 209–228, 1986.
5. J. P. Van Doormaal and G. D. Raithby, An Evaluation of the Segregated Approach for Predicting Incompressible Fluid Flows, ASME Paper 85-HT-9, National Heat Transfer Conference, Denver, CO, August 4–7, 1985.
6. B. R. Hutchinson, P. F. Galpin, and G. D. Raithby, Application of Additive Correction Multigrid to the Coupled Fluid Flow Equations, *Numer. Heat Transfer.*, vol. 13, pp. 133–147, 1988.
7. R. Webster, Performance of Algebraic Multi-grid Solvers Based on Unsmoothed and Smoothed Aggregation Schemes, *Int. J. Numer Meth. Fluids*, vol. 36, pp. 743–773, 2001.
8. F. Moukalled and M. Darwish, A Unified Formulation of the Segregated Class of Algorithms for Fluid Flow at All Speed, *Numer. Heat Transfer B*, vol. 37, pp. 103–139, 2000.
9. A. Brandt, Multi-level Adaptive Technique (MLAT) for Fast Numerical Solution to Boundary Value Problems, in H. Cabannes and R. Teman (eds.), *Proc. 3rd Int. Conf. on Numerical Methods in Fluid Mechanics*, University of Paris, Springer-Verlag, New York, Berlin, Heidelberg, 1973.
10. A. Brandt, Multi-level Adaptive Solutions to Boundary Value Problems, *Math. Comput.*, vol. 31, pp. 333–390, 1977.
11. W. Hackbusch, A Fast Numerical Method for Elliptic Boundary Value Problems with Variable Coefficients, in E. H. Hirschel and W. Geller (eds.), *2nd GAMM-Conf. on Numerical Methods in Fluid Mechanics*, pp. 50–57, Köln, 1977.
12. A. Brandt, Algebraic Multigrid Theory: The Symmetric Case, *Appl. Math. Comput.*, vol. 19, pp. 24–56, 1986.
13. B. R. Hutchinson and G. D. Raithby, A Multigrid Method Based on the Additive Correction Strategy, *Numer. Heat Transfer.*, vol. 9, pp. 511–537, 1986.

14. Y. H. Hwang, Unstructured Additive Correction Multigrid Method for the Solution of Matrix Equations, *Numer. Heat Transfer B*, vol. 27, pp. 195–212, 1995.
15. B. Shome, An Enhanced Additive Correction Multigrid Method, *Numer. Heat Transfer B*, vol. 49, pp. 395–407, 2006.
16. R. E. Phillips and F. W. Schmidt, Multigrid Techniques for the Solution of the Passive Scalar Advection-Diffusion Equation, *Numer. Heat Transfer*, vol. 8, pp. 25–43, 1985.
17. J. W. Ruge and K. Stüben, Algebraic Multigrid (AMG), in S. F. McCormick (ed.), *Multigrid Methods*, Frontiers in Applied Mathematics, vol. 3, pp. 73–130, SIAM, Philadelphia, 1987.
18. K. Stüben, Algebraic Multigrid (AMG): Experiences and Comparisons, *Appl. Math. Comput.*, vol. 13, pp. 419–452, 1983.
19. W. D. Gropp, D. K. Kauchik, D. E. Keyes, and B. F. Smith, High Performance Parallel Implicit CFD, *Parallel Comput.*, vol. 27, pp. 337–362, 2001.
20. R. Kanapady, K. K. Tamma, and A. Mark, Highly Scalable Parallel Computational Models for Large-Scale RTM Process Modeling Simulations. I. Theoretical Formulations and Generic Design, *Numer. Heat Transfer B*, vol. 36, pp. 265–284, 1999.
21. S. P. Burns and M. A. Christon, Spatial Domain-Based Parallelism in Large-Scale Radiative Transport Applications, *Numer. Heat Transfer B*, vol. 31, pp. 401–421, 1997.
22. V. Dolean and S. Lanteri, Parallel Multigrid Methods for the Calculation of Unsteady Flows on Unstructured Grids: Algorithmic Aspects and Parallel Performances on Clusters of PCs, *Parallel Comput.*, vol. 30, pp. 503–525, 2004.
23. Z. H. Yan, Parallel Computation of Turbulent Combustion and Flame Spread in Fires, *Numer. Heat Transfer B*, vol. 41, pp. 191–208, 2002.
24. K. Sungmo and K. Yongmo, Parallel Unstructured-Grid Finite-Volume Method for Turbulent Nonpremixed Flames Using the Flamelet Model, *Numer. Heat Transfer B*, vol. 43, pp. 525–547, 2003.
25. W. Mingyu and J. G. Georgiadis, Parallel Computation of Forced Convection Using Domain Decomposition, *Numer Heat Transfer B*, vol. 20, pp. 41–59, 1991.
26. R. P. Fedorenko, A Relaxation Method for Solving Elliptic Difference Equations, *Comput. Math. Math. Phys.*, vol. 4, pp. 1092–1096, 1964.
27. R. P. Fedorenko, The Speed of Convergence of the Iterative Process, *Comput. Math. Math. Phys.*, vol. 4, pp. 227–235, 1964.
28. F. V. Poussin, An Accelerated Relaxation Algorithm for Iterative Solution of Elliptic Equations, *SIAM J. Numer. Anal.*, vol. 5, pp. 340–351, 1968.
29. A. Brandt, *Multigrid Techniques, Guide with Applications to Fluid Dynamics*, von Karman Institute for Fluid Dynamics, Belgium, 1984.
30. W. L. Briggs, *A Multigrid Tutorial*, Society of Industrial and Applied Mathematics, Philadelphia, PA, 1987.
31. S. R. Elias, G. D. Stubley, and G. D. Raithby, An Adaptive Agglomeration Method for Additive Correction Multigrid, *Int. J. Numer. Meth. Eng.*, vol. 40, pp. 887–903, 1997.
32. D. Mavriplis and V. Venkatakfrishnan, Agglomeration Multigrid for Viscous Turbulent Flows, AIAA paper 94-2332, 1994.
33. R. E. Phillips and F. W. Schmidt, A Multilevel-Multigrid Technique for Recirculating Flows, *Numer. Heat Transfer*, vol. 8, pp. 573–594, 1985.
34. R. D. Lonsdale, An Algebraic Multigrid Scheme for Solving the Navier-Stokes Equations on Unstructured Meshes, in C. Taylor, J. H. Chin, and G. M. Homsy (eds.), *Numerical Methods in Laminar and Turbulent Flow*, vol. 7, part 2, pp. 1432–1442, Pineridge Press, Swansea, UK, 1991.
35. M. S. Guerrero, Parallel Multigrid Algorithms for Computational Fluid Dynamics and Heat Transfer, Doctoral thesis, Department de Maquines I Motots Trmics, Universitat Politecnica de Catalunya, 2000.

36. E. Perez, A 3D Finite Element Multigrid Solver for the Euler Equations, INRIA Rep. 442, September 1985.
37. S. D. Connell and D. G. Braaten, A 3D Unstructured Adaptive Multigrid Scheme for the Euler Equations, *AIAA J.*, vol. 32, pp. 1626–1632, 1994.
38. V. Parthasarathy and Y. Kallinderis, New Multigrid Approach for Three-Dimensional Unstructured Adaptive Grids, *AIAA J.*, vol. 32, pp. 956–963, 1994.
39. D. J. Mavriplis, Three-Dimensional Multigrid Reynolds-Averaged Navier-Stokes Solver for Unstructured Meshes, *AIAA J.*, vol. 33, No. 12, pp. 445–453, 1995.
40. W. Qinghua and J. Yogendra, Algebraic Multigrid Preconditioned Krylov Subspace Methods for Fluid Flow and Heat Transfer on Unstructured Meshes, *Numer. Heat Transfer B*, vol. 49, pp. 197–221, 2006.
41. M. Lallemand, H. Steve, and A. Dervieux, Unstructured Multigridding by Volume Agglomeration: Current Status, *Comput. and Fluids*, vol. 21, pp. 397–433, 1992.
42. D. J. Mavriplis, Directional Agglomeration Multigrid Techniques for High Reynolds Number Viscous Flow Solvers, *AIAA J.*, vol. 37, pp. 393–415, 1999.
43. U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, London, 2001.
44. B. Wilkinson and C. M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice Hall, Upper Saddle River, NJ, 1999.
45. L. C. Dutto, W. G. Habashi, and M. Fortin, An Algebraic Multilevel Parallelizable Pre-conditioner for Large-Scale CFD Problems, *Comput. Meth. Appl. Mech. Eng.*, vol. 149, pp. 303–318, 1997.
46. D. Kwak, C. Kiris, and C. S. Kim, Computational Challenges of Viscous Incompressible Flows, *Comput. Fluids*, vol. 34, pp. 283–299, 2005.
47. G. Karypis and V. Kumar, Multilevel k-Way Partitioning Scheme for Irregular Graphs, *J. Parallel Distrib. Comput.*, vol. 48, pp. 96–129, 1998.
48. K. McManus, A Strategy for Mapping Unstructured Mesh Computational Mechanics Programs Onto Distributed Memory Parallel Architectures, *School of Computing and Mathematics Science, University of Greenwich, London, UK*, vol. 1, pp. 1–213, 1995.
49. A. Krechel and K. Stüben, Parallel Algebraic Multigrid Based on Subdomain Blocking, *Parallel Comput.*, vol. 27, pp. 1009–1031, 2001.
50. W. D. Gropp, Parallel Computing and Domain Decomposition, in T. F. Chan, D. E. Keyes, G. A. Meurant, J. S. Scroggs, and R. G. Voigt (eds.), *Fifth Conf. on Domain Decomposition Methods for Partial Differentia Equations*, pp. 349–362, SIAM, Philadelphia, PA, 1992.
51. R. M. Smith and A. G. Hutton, The Numerical Treatment of Advection: A Performance Comparison of Current Methods, *Numer. Heat Transfer*, vol. 5, pp. 439–461, 1982.