Chapter 1

A Survey of Parallelization Techniques for Multigrid Solvers

Edmond Chow, Robert D. Falgout, Jonathan J. Hu, Raymond S. Tuminaro and Ulrike Meier Yang

This paper surveys the techniques that are necessary for constructing computationally efficient parallel multigrid solvers. Both geometric and algebraic methods are considered. We first cover the sources of parallelism, including traditional spatial partitioning and more novel additive multilevel methods. We then cover the parallelism issues that must be addressed: parallel smoothing and coarsening, operator complexity, and parallelization of the coarsest grid solve.

1.1 Introduction

The multigrid algorithm is a fast and efficient method for solving a wide class of integral and partial differential equations. The algorithm requires a series of problems be "solved" on a hierarchy of grids with different mesh sizes. For many problems, it is possible to prove that its execution time is asymptotically optimal. The niche of multigrid algorithms is large-scale problems where this asymptotic performance is critical. The need for high-resolution PDE simulations has motivated the parallelization of multigrid algorithms. It is our goal in this paper to provide a brief but structured account of this field of research. Earlier comprehensive treatments of parallel multigrid methods can be found in [26, 61, 71, 52] and Chapter 6 of [75].

1.2 Sources of Parallelism

1.2.1 Partitioning

Most simulations based on partial differential equations (PDEs) are parallelized by dividing the domain of interest into subdomains (one for each processor). Each processor is then responsible for updating the unknowns associated within its subdomain only. For logically rectangular meshes, partitioning into boxes or cubes is straight-forward. For unstructured meshes there are several tools to automate the subdivision of domains [55, 50, 34]. The general goal is to assign each processor an equal amount of work and to minimize the amount of communication between processors by essentially minimizing the surface area of the subdomains.

Parallelization of standard multigrid algorithms follows in a similar fashion. In particular, V- or W-cycle computations within a mesh are performed in parallel but each mesh in the hierarchy is addressed one at a time as in standard multigrid (i.e., the fine mesh is processed and then the next coarser mesh is processed, etc.). For partitioning, the finest grid mesh is usually subdivided ignoring the coarse meshes[1]. While coarse mesh partitioning can also be done in this fashion, it is desirable that the coarse and fine mesh partitions "match" in some way so that inter-processor communication during grid transfers is minimized. This is usually done by deriving coarse mesh partitions from the fine mesh partition. For example, when the coarse mesh points are a subset of fine mesh points, it is natural to simply use the fine mesh partitioning on the coarse mesh. If the coarse mesh is derived by agglomerating elements and the fine mesh is partitioned by elements, the same idea holds. In cases without a simple correspondence between coarse and fine meshes, it is often easy and natural to enforce a similar condition that coarse grid points reside on the same processors that contain most of the fine grid points that they interpolate [60]. For simulations that are to run on many processors (i.e., much greater than 100) or on networks with relatively high communication latencies, repartitioning the coarsest meshes may be advantageous. There are two such approaches worth noting. The first serializes computations by mapping the entire problem to a single processor at some coarse level in the multigrid cycle [66]. The second replicates computations by combining each processor's coarse mesh data with data from neighboring processors [49, 84]. Both approaches reduce communication costs. Repartitioning meshes is also important in cases where the original fine mesh is not well-balanced, leading to significant imbalances on coarse meshes. Another challenge is when the discretization stencil grows on coarse meshes, as is common with algebraic multigrid. Here, a subdomain on a coarse mesh may need to communicate with a large number of processors to perform its updates, leading to significant overhead.

[1] When uniform refinement is used to generate fine grid meshes, it is more natural to partition the coarse mesh first. When adaptive refinement is used, it is useful to consider all meshes during partitioning.

1.2.2 Specialized Parallel Multigrid Methods

Parallelization of standard multigrid methods yields highly efficient schemes so long as there is sufficient work per processor on the finest mesh. When this is not the case, however, the parallel efficiency of a multigrid algorithm can degrade noticeably due to inefficiencies associated with coarse grid computations. In particular, the number of communication messages on coarse meshes is often nearly the same as that on fine meshes (although messages lengths are much shorter). On most parallel architectures, communication latencies are high compared to current processor speeds and so coarse grid calculations can be dominated by communication. Further, machines with many processors can eventually reach situations where the number of processors exceeds the number of coarse mesh points implying that some processors are idle during these computations. To address these concerns, specialized parallel multigrid-like methods have been considered. Most of these highly parallel multigrid methods fit into four broad categories: concurrent iterations, multiple coarse corrections, full domain partitioning, and block factorizations. The concurrent iteration approach reduces the time per multigrid iteration by performing relaxation sweeps on all grids simultaneously. The multiple coarse grid methods accelerate convergence by projecting the fine grid system onto several different coarse grid spaces. Full domain partitioning reduces the number of communication messages per iteration by only requiring processors to exchange information on the finest and coarsest mesh. Block factorizations use a special selection of coarse and fine points to reveal parallelism.

Concurrent Iterations

The principal element of concurrent iteration methods is the distribution of the original problem over a grid hierarchy so that simultaneous processing of the grids can occur. In this way an iteration can be performed more quickly. Methods which fall into this family include any kind of additive multilevel method such as additive two level domain decomposition schemes [71] as well as additive hierarchical basis type methods [90, 86]. In addition, to these well-known preconditioning schemes, special multigrid algorithms have been proposed in [42, 43, 73, 78, 40]. All of these methods divide the computation over meshes so that the individual problems do not greatly interfere with each other. The general idea is to focus fine grid relaxations on high frequency errors and coarse grid relaxations on low frequency errors. This can be done, for example, by first splitting the residual into oscillatory and smooth parts. Then, the oscillatory part is used with the fine grid relaxation while the smooth part is projected onto coarser meshes. Another way to reduce interference between fine and coarse grid computations is to enforce some condition (e.g., an orthogonality condition) when individual solutions are recombined. Unfortunately, while these methods are interesting, convergence rates can suffer, and the efficient mapping of the grid pyramid onto separate processors is a non-trivial task. A more complete discussion of additive multigrid methods can be found in [12]. Theoretical aspects of additive multigrid are established in [17].

Multiple Coarse Grids

Multiple correction methods employ additional coarse grid corrections to further improve convergence rates. The key to success here is that these additional corrections must do beneficial work without interfering with each other. To illustrate the idea, consider the simple case of one grid point assigned to each processor for a three dimensional simulation. The number of mesh points is reduced by eight within a standard hierarchy and so most processors are inactive even on the first coarse mesh. However, if each time the current mesh spawns eight coarse grid correction equations then all the processors are kept active throughout the V-cycle. While this situation of one grid point per processor is academic, communication can so dominate time on coarse meshes within realistic computations that additional subproblems can be formulated at little extra cost.

The most well-known algorithms in this family are due to Frederickson and McBryan [41] and Hackbusch [47, 48]. Each of these algorithms was originally formulated for structured mesh computations in two dimensions. In the Fredrickson and McBryan method the same fairly standard interpolation and projection operators are used for all four subproblems. The stencils, however, for the different problems are shifted, i.e., coarse points for the first subproblem are exactly aligned with fine mesh points that correspond to the intersection of even numbered mesh lines in both the x and y directions. Coarse points for the second subproblem coincide with the intersection of even numbered mesh lines in the x direction and odd numbered mesh lines in the y direction. The third and fourth subproblems are defined in a similar fashion. The similarities in the four subproblems makes for a relatively easy algorithm to implement and analyze. The major benefit of the three additional subproblems is that the combined coarse grid corrections essentially contain no aliasing error[2]. This is due to a beneficial cancellation of aliasing error on the separate grids so that it does not reappear on the fine grid [25]. Further extensions and analysis of this algorithm is pursued in [85]. This work is closely related to methods based on the use of symmetries [35].

For a two dimensional problem, Hackbusch's parallel multigrid method also uses four projections for two dimensional problems. The stencils for the four different restriction operators are given by

$$\begin{aligned} R^{(1)} &= \frac{1}{8}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}, & R^{(2)} &= \frac{1}{8}\begin{pmatrix} -1 & 2 & -1 \\ -2 & 4 & -2 \\ -1 & 2 & -1 \end{pmatrix}, \\ R^{(3)} &= \frac{1}{8}\begin{pmatrix} -1 & -2 & -1 \\ 2 & 4 & 2 \\ -1 & -2 & -1 \end{pmatrix}, & R^{(4)} &= \frac{1}{8}\begin{pmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{pmatrix} \end{aligned} \tag{1.1}$$

and interpolation is taken as the transpose of restriction. The idea is to project the fine grid problem into spaces associated with both high and low frequencies in the x and y directions, i.e., $R^{(1)}$ projects into low frequency spaces in both the x

[2] Aliasing error arises when high frequency components are projected onto coarser grids. These frequencies get mapped to low frequency and are often amplified on the coarse grid before returning to the fine mesh.

and y directions, and $R^{(2)}$ projects into a space corresponding to low frequency in the y direction but high frequency in the x direction. These projections into both high and low frequency spaces can be useful when it is difficult to smooth in certain directions (e.g., anisotropic problems) or they can be used to reduce the number of smoothing iterations on the fine grid (as the high frequency subproblems can be used to reduce oscillatory error). It was shown in [32] how to modify this method to be more robust for discontinuous coefficient problems and in [31] how to apply this method with line relaxation on ocean modeling problems. Further extensions and improvements are discussed in [7].

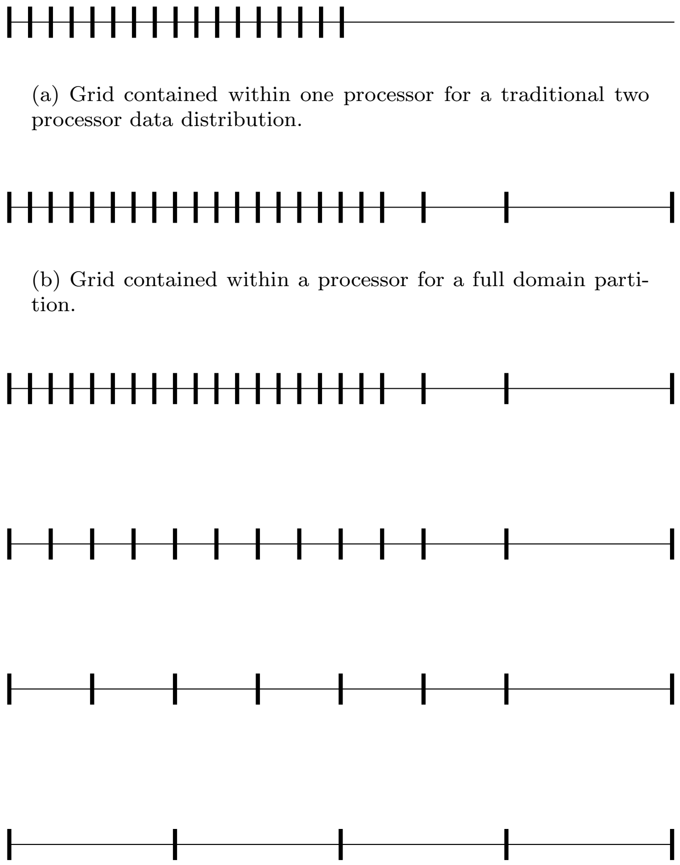
Another interesting and related idea is due to Mulder [64] which also uses multiple coarse grid corrections to address anisotropic problems. In Mulder's method, different coarse grid corrections are built by applying semicoarsening[3] in different directions. That is, one coarse mesh is built only coarsening in the x direction while another coarsens in the y direction. The idea is similar to Hackbusch's method in that the idea is to improve robustness by having multiple spaces which are intended to address problems where smoothing is difficult in some direction. Improvements to this idea have been presented in [65].

Full Domain Partitioning

Full domain partitioning takes a different approach that is intended to reduce the number of messages that are sent during each multigrid cycle. The idea was motivated by adaptive grid refinement and is described in the context of hierarchical basis methods [62, 63]. Here, we will give the general flavor of the method in a more traditional multigrid setting.

To illustrate full domain partitioning consider a one-dimensional PDE discretized on a uniform mesh. Instead of subdividing the mesh and assigning each piece to a different processor as shown in Figure 1.1(a), an auxiliary mesh (with a corresponding discrete PDE operator) for each processor spans the entire domain. A sample is shown for one particular processor in Figure 1.1(b). While each processor's mesh spans the entire domain, only a subregion of the processor's mesh actually corresponds to the fine grid mesh. Additionally, the resolution within a processor's mesh decreases the further we are from the subregion. The basic idea is that each processor performs multigrid on its adaptive grid using a mesh hierarchy suitable for adaptive meshes such as the one shown in Figure 1.1(c). The multigrid cycle of each processor is almost completely independent of other processors except for communication on the finest level and coarsest level. This greatly improves the ratio of computation to communication within each multigrid cycle. In [62], convergences rates comparable to standard multigrid are obtained at much higher efficiencies using this full domain partition approach. In [9, 10], these ideas are expanded upon to easily adapt and load balance an existing serial code PLTMG [8] to a parallel environment in an efficient way. Most recently, [11, 58] have developed new parallel algebraic multigrid solvers motivated by these ideas.

[3] Semi-coarsening refers to coarsening the mesh only in some subset of coordinate directions. The idea is to not coarsen in directions where smoothing is ineffective.



(a) Grid contained within one processor for a traditional two processor data distribution.

(b) Grid contained within a processor for a full domain partition.

(c) Full multigrid hierarchy for a single processor using full domain partition.

Figure 1.1. *Full Domain Partitioning Example.*

Parallel Multilevel Block LU Factorizations

Parallel multilevel algorithms have also been developed in the context of approximate block LU factorizations. To illustrate the method with two levels, the variables in a matrix A is partitioned into fine (f) and coarse (c) sets, and the approximate block LU factorization is

$$\begin{pmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{pmatrix} \approx \begin{pmatrix} A_{ff} & 0 \\ A_{cf} & S \end{pmatrix} \begin{pmatrix} I & -P \\ 0 & I \end{pmatrix}$$

where S is the Schur complement and P is an approximation to $-A_{ff}^{-1}A_{fc}$. The similarity to multigrid methods is evident when $[P, I]^T$ is viewed as a prolongation operator, an approximation to A_{ff} is viewed as a smoother for the fine variables, and S is a suitable coarse grid matrix.

Parallel multilevel versions of this algorithm were first developed by choosing an *independent set* of fine grid variables, i.e., A_{ff} is diagonal, although actual parallel implementations were not tested [69, 15, 16]. Practical parallel versions were then developed by using a domain decomposition ordering of A, where A_{ff} is block diagonal, possibly with multiple blocks per processor; see [57] and the references therein. Each block represents a small aggregate, and the boundary between the aggregates fall into the coarse grid. If A_{ff} has general form, parallelism can be recovered by using a sparse approximate inverse to approximate the inverse of A_{ff}. See, for example, [28]. This is equivalent to using a sparse approximate inverse smoother, to be discussed later.

1.3 Parallel Computation Issues

The remainder of this paper primarily considers parallelization of standard multigrid algorithms (as opposed to those considered in the previous subsection). The main steps in developing multigrid methods include: coarsening the fine grid (or fine matrix graph), choosing grid transfer operators to move between meshes, determining the coarse mesh discretization matrices[4], and finally developing appropriate smoothers. Developing effective multigrid methods often boils down to striking a good balance between setup times, convergence rates, and cost per iteration. These features in turn depend on operator complexity, coarsening rates, and smoother effectiveness.

1.3.1 Complexity

Complexity Issues in Geometric Solvers

On sequential computers, complexity is not typically a concern for geometric multigrid methods. In parallel, however, implementation issues can lead to large complexities, even for algorithms that exhibit adequate parallelism.

As an illustrative example, consider the 3D SMG semi-coarsening multigrid method described in [70]. This SMG method uses a combination of semi-coarsening and plane relaxation to achieve a high degree of robustness. It is recursive, employing one V-cycle of a 2D SMG method to effect the plane solves. The computational complexity of the method is larger than standard geometric methods, but it is still optimal.

The storage costs for relaxation can be kept to $O(N^2)$ in the sequential code, which is small relative to the $O(N^3)$ original problem size, where N is the problem size along one dimension. Alternatively, a faster solution time can be achieved by saving the coarse grid information for each of the plane solves, but at the cost of $O(N^3)$ storage for relaxation. In parallel, there is little choice. The solution of one

[4] The coarse grid discretization matrix in an algebraic multigrid method is usually generated by a Galerkin process—the coarsening and grid transfer operators determine the coarse discretization operators.

plane at a time in relaxation would incur a communication overhead that is too great and that depends on N. To achieve reasonable scalability, all planes must be solved simultaneously, which means an additional $O(N^3)$ storage requirement for relaxation that more than doubles the memory (see [24, 37]).

Another parallel implementation issue in SMG that exacerbates the storage cost problem is the use of *ghost zones*, which is simply the extra "layer" of data needed from another processor to complete a processor's computation. For parallel geometric methods, the use of ghost zones is natural and widely used. It simplifies both implementation and maintainability, and leads to more efficient computational kernels. However, because of the recursive nature of SMG and the need to store all coarse-grid information in the plane solves, the ghost zone memory overhead is quite large and depends logarithmically on N (see [37]).

Complexity Issues in Algebraic Solvers

For algebraic multigrid solvers, there are two types of complexities that need to be considered: the *operator complexity* and the *stencil size*. The operator complexity is defined as the quotient of the sum of the numbers of nonzeros of the matrices on all levels, $A^i, i = 1, \ldots, M$, (M levels) divided by the number of nonzeros of the original matrix $A^1 = A$. This measure indicates how much memory is needed. If memory usage is a concern, it is important to keep this number small. It also affects the number of operations per cycle in the solve phase. Small operator complexities lead to small cycle times. The stencil size of a matrix A is the average number of coefficients per row of A. While stencil sizes of the original matrix are often small, it is possible to get very large stencil sizes on coarser levels. Large stencil sizes can lead to large setup times, even if the operator complexity is small, since various components, particularly coarsening and to some degree interpolation, require that neighbors of neighbors are visited and so one might observe superlinear or even quadratic growth in the number of operations when evaluating the coarse grid or the interpolation matrix. Large stencil sizes can also increase parallel communication cost, since they might require the exchange of larger sets of data.

Both convergence factors and complexities need to be considered when defining the coarsening and interpolation procedures, as they often affect each other; increasing complexities can improve convergence, and small complexities lead to a degradation in convergence. The user needs therefore to decide his/her priority. Note that often a degradation in convergence due to low complexity can be overcome or diminished by using the multigrid solver as a preconditioner for a Krylov method.

1.3.2 Coarsening

The parallelization of the coarsening procedure for geometric multigrid methods and block-structured problems in the algebraic case are fairly straight-forward. On the other hand, the standard coarsening algorithms for unstructured problems in algebraic multigrid are highly recursive and not suitable for parallel computing. We first describe some issues for coarsening block-structured problems, and then move on to unstructured problems.

Coarsening for Block-Structured Problems

Geometric multigrid methods have traditionally been discussed in the context of rectangular structured grids, i.e., Cartesian grids on a square in 2D or a cube in 3D (see, e.g., [33, 6, 7]). In this setting, computing coarse grids in parallel is a trivial matter, and only the solution phase of the algorithm is of interest. However, in the more general setting where grids are composed of arbitrary unions of rectangular *boxes* such as those that arise in structured adaptive mesh refinement applications, parallel algorithms for coarsening are important [37]. Here, a box is defined by a pair of indexes in the 3D index-space (there is an obvious analogue for 2D), $\mathcal{I} = \{(i, j, k) : i, j, k \text{ integers}\}$. That is, a box represents the "lower" and "upper" corner points of a subgrid via the indices $(i_l, j_l, k_l) \in \mathcal{I}$ and $(i_u, j_u, k_u) \in \mathcal{I}$.

In the general setting of a parallel library of sparse linear solvers, the problem data has already been distributed and is given to the solver library in its distributed form. On each processor, the full description of each grid's distribution is not needed, only the description of the subgrids that it owns and their "nearest" neighboring subgrids. However, to compute this on all grid levels requires that at least one of the processors—one containing a nonempty subgrid of the coarsest grid—has information about the coarsening of every other subgrid. In other words, computing the full set of coarse grids in the V-cycle requires global information.

Assume that we have already determined some appropriate neighbor information (at a cost of $\log(P)$ communications), and consider the following two basic algorithms for coarsening, denoted by A1 and A2. In A1, each processor coarsens the subgrids that it owns and receives neighbor information from other processors. This requires $O(1)$ computations and $O(\log(N))$ communications. In A2, the coarsening procedure is replicated on all processors, which requires $O(P)$ computations and no communications. This latter approach works well for moderate numbers of processors, but becomes prohibitive for large P. In particular, the latter approach also requires $O(P)$ storage, which may not be practical for machines with upwards of 100K processors such as BlueGene/L.

The performance of these two basic algorithms is discussed in more detail in [37], and results are also presented that support the analysis. Algorithm A1 is much harder to implement than A2 because of the complexity of determining new nearest neighbor information on coarser grid levels while storing only $O(1)$ grid boxes.

Sequential Coarsening Strategies for Unstructured Problems

Before describing any parallel coarsening schemes, we will describe various sequential coarsening schemes, since most parallel schemes build on these. There are basically two different ways of choosing a coarse grid: "classical" coarsening [18, 68, 72], and coarsening by agglomeration [82].

Classical coarsening strives to separate all points i into either coarse points (C-points), which are taken to the next level, and fine points (F-points), which are interpolated by the C-points. Since most if not all matrix coefficients are equally important for the determination of the coarse grids, one should only consider those matrix entries which are sufficiently large. Therefore only *strong connections* are considered. A point i depends strongly on j, or conversely, j strongly influences i if

$$-a_{ij} \geq \theta \max_{k \neq i}(-a_{ik}) \tag{1.2}$$

where θ is a small constant. In the classical coarsening process (which we will denote Ruge-Stüben or RS coarsening) an attempt is made to fulfill the following two conditions. In order to restrict the size of the coarse grid, condition (C1) should be fulfilled: the C-points should be a maximal independent subset of all points, i.e. no two C-points are connected to each other, and if another C-point is added then independence is lost. To ensure the quality of interpolation, a second condition (C2) needs to be fulfilled: For each point j that strongly influences an F-point i, j is either a C-point or strongly depends on a C-point k that also strongly influences i.

RS coarsening consists of two passes. In the first pass, which consists of a maximal independent set algorithm, each point i is assigned a measure λ_i, which equals the number of points that are strongly influenced by i. Then a point with a maximal λ_i (there usually will be several) is selected as the first coarse point. Now all points that strongly depend on i become F-points. For all points that strongly influence these new F-points, λ_j is incremented by the number of new F-points that j strongly influences in order to increase j's chances of becoming a C-point. This process is repeated until all points are either C- or F-points. Since this first pass does not guarantee that condition (C2) is satisfied, it is followed by a second pass, which examines all strong F-F connections for common coarse neighbors. If (C2) is not satisfied new C-points are added.

Experience has shown that often the second pass generates too many C-points, causing large complexities and inefficiency [72]. Therefore condition (C1) has been modified to condition (C1$'$): Each F-point i needs to strongly depend on at least one C-point j. Now just the first pass of the RS coarsening fulfills this requirement. This method leads to better complexities, but worse convergence. Even though this approach often decreases complexities significantly, complexities can still be quite high and require more memory than desired. Allowing C-points to be even further apart leads to *aggressive coarsening*. This is achieved by the following new definition of strength: A variable i is *strongly n-connected along a path of length l* to a variable j if there exists a sequence of variables $i_0, i_1, \ldots, i_l$ with $i = i_0$, $j = i_l$ and i_k strongly connected (as previously defined) to i_{k+1} for $k = 0, 1, \ldots, l-1$. A variable i is *strongly n-connected w.r.t. (p,l)* to a variable j if at least p paths of lengths $\leq l$ exist such that i is strongly n-connected to j along each of these paths. For further details see [72].

Coarsening by aggregation accumulates aggregates, which are the coarse "points" for the next level. For the aggregation scheme, a matrix coefficient a_{ij} is dropped if the following condition is fulfilled:

$$|a_{ij}| \leq \theta \sqrt{|a_{ii} a_{jj}|}. \tag{1.3}$$

An aggregate is defined by a root point i and its neighborhood, i.e. all points j, for which a_{ij} fulfills (1.3). The basic aggregation procedure consists of the following two phases. In the first pass, a root point is picked that is not adjacent to any existing aggregate. This procedure is repeated until all unaggregated points are adjacent to an aggregate. In the second pass, all remaining unaggregated points are either integrated into already existing aggregates or used to form new aggregates. Since root points are connected by paths of length at least 3, this approach leads to a fast coarsening and small complexities. While aggregation is fundamentally different from classical coarsening, many of the same concerns arise. In particular, considerable care must be exercised in choosing root points to limit the number of unaggregated points after the first pass. Further care must be exercised within the second pass when deciding to create new aggregates and when determining what points should be placed within which existing aggregate. If too many aggregates are created in this phase, complexities grow. If aggregates are enlarged too much or have highly irregular shapes, convergence rates suffer.

Parallel Coarsening Strategies for Unstructured Problems

The most obvious approach to parallelize any of the coarsening schemes described in the previous section is to partition all variables into subdomains, assign each processor a subdomain, coarsen the variables on each subdomain using any of the methods described above, and find a way of dealing with the variables that are located on the processor boundaries.

The easiest option, a decoupled coarsening scheme, i.e., just ignoring the processor boundaries, is the most efficient one, since it requires no communication, but will most likely not produce a good coarse grid. For the RS coarsening, it generally violates condition (C1) by generating strong F-F connections without common coarse neighbors and leads to poor convergence [51]. While in practice this approach might lead to fairly good results for coarsening by aggregation [79], it can produce many aggregates near processor boundaries that are either smaller or larger than an ideal aggregate and so lead to larger complexities or have a negative effect on convergence. Another disadvantage of this approach is that it cannot have fewer coarse points or aggregates than processors, which can lead to a large coarsest grid.

There are various ways of dealing with the variables on the boundaries. One possible way of treating this problem—after one has performed both passes on each processor independently—is to perform a third pass only on the processor boundary points which will add further C-points and thus ensure that condition (C1) is fulfilled. This approach is called RS3 [51]. One of the disadvantages of this approach is that this can generate C-point clusters on the boundaries, thus increasing stencil sizes at the boundaries where in fact one would like to avoid C-points in order to keep communication costs low.

Another parallel approach is subdomain blocking [56]. Here, coarsening starts with the processor boundaries, and one then proceeds to coarsen the inside of the domains. Full subdomain blocking is performed by making all boundary points coarse and then coarsening into the interior of the subdomain by using any coarsening scheme, such as one pass of RS coarsening or any of the aggressive coarsening schemes. Like RS3 coarsening, this scheme generates too many C-points on the boundary. A method which avoids this problem is minimum subdomain blocking. This approach uses standard coarsening on the boundaries and then coarsens the interior of the subdomains.

In the coupled aggregation method, aggregates are first built on the boundary. This step is not completely parallel. When there are no more unaggregated points adjacent to an aggregate on the processor boundaries, one can proceed to choose aggregates in the processor interiors, which can be done in parallel. In the third phase, unaggregated points on the boundaries and in the interior are swept into local aggregates. Finally, if there are any remaining points, new local aggregates are formed. This process yields significantly better aggregates and does not limit the coarseness of grids to the number of processors, see [79]. Another aggregation scheme suggested in [79] is based on a parallel maximally independent set algorithm, since the goal is to find an initial set of aggregates with as many points as possible with the restriction that no root point can be adjacent to an existing aggregate. Maximizing the number of aggregates is equivalent to finding the largest number of root points such that the distance between any two root points is at least three. This can be accomplished by applying a parallel maximally independent set (MIS) algorithm, e.g., the asynchronous distributed memory algorithm ADMMA [3], to the square of the matrix in the first phase of the coupled aggregation scheme.

A parallel approach that is independent on the number of processors is suggested in [29, 51] for classical coarsening. It is based on parallel independent set algorithms as described by Luby [59] and Jones and Plassmann [53]. This algorithm, called CLJP coarsening, begins by generating global measures as in RS coarsening, and then adding a random number between 0 and 1 to each measure, thus making them distinct. It is now possible to find unique local maxima. The algorithm proceeds as follows: If i is a local maximum, make i a C-point, eliminate the connections to all points j that influence i and decrement j's measure. (Thus rather than immediately turning C-point neighbors into F-points, we increase their likelihood of becoming F-points. This combines the two passes of RS coarsening into one pass.) Further, for any point j that depends on i, remove its connection to i and examine all points k that depend on j to see whether they also depend on i. If i is a common neighbor for both k and j decrement the measure of j and remove the edge connecting k and j from the graph. If a measure gets smaller than 1, the point associated with it becomes an F-point. This procedure does not require the existence of a coarse point in each processor as the coarsening schemes above and thus coarsening does not slow down on the coarser levels. While this approach works fairly well on truly unstructured grids, it often leads to C-point clusters and fairly high complexities on structured grids. These appear to be caused by fulfilling condition (C1). To reduce operator complexities, while keeping the property of being independent of the number of processors, a new algorithm, the PMIS coarsening [30], has been developed that is more comparable to using one pass of the RS coarsening. While it does not fulfill condition (C1), it fulfills condition (C1$'$). PMIS coarsening begins just as the CLJP algorithm with distinct global measures, and sets local maxima to be C-points. Then points that are influenced by C-points are made into F-points, and are eliminated from the graph. This procedure will continue until all points are either C- or F-points.

An approach which has shown to work quite well for structured problems is the following combination of the RS and the CLJP coarsening which is based on an idea by Falgout [51]. This coarsening starts out as the decoupled RS coarsening. It then uses the C-points that have been generated in this first step and are located in the interior of each processor as the first independent set (i.e., they will all remain C-points) and feeds them into the CLJP-algorithm. The resulting coarsening, which satisfies condition (C1), fills the boundaries with further C-points and possibly adds a few in the interior of the subdomains. A more aggressive scheme, which satisfies condition (C1'), and uses the same idea, is the HMIS coarsening [30]. It performs only the first pass of the RS coarsening to generate the first independent set, which then is used by the PMIS algorithm.

Another approach is to color the processors so that subdomains of the same color are not connected to each other. Then all these subdomains can be coarsened independently. This approach can be very inefficient since it might lead to many idle processors. An efficient implementation that builds on this approach can be found in [54]. Here the number of colors is restricted to n_c, i.e., processors with color numbers higher than n_c are assigned the color n_c. Good results were achieved using only two colors on the finest level, but allowing more colors on coarser levels.

1.3.3 Smoothers

Except for damped Jacobi smoothing, traditional smoothers such as Gauß-Seidel are inherently sequential. In this section, we describe some alternatives that have been developed that have better parallel properties.

Multicolor Gauß-Seidel

One of the most popular smoother choices for multigrid is Gauß-Seidel relaxation, which is a special case of successive over relaxation (SOR) [89]. Although Gauß-Seidel is apparently sequential in nature, one method for exposing parallelism is to use multicoloring. In this approach, the unknown indices are partitioned into disjoint sets $U_1, \ldots, U_k$. Each set is thought of as having a distinct color. Let $A = (a_{ij})$. Each set U_l has the property that if $i, j \in U_l$, then $a_{ij} = a_{ji} = 0$, i.e,. the equation for unknown i does not involve unknown j, and vice-versa. Unknowns in the same set can be updated independently of each another. Hence, the unknowns of single color can be updated in parallel. In addition to imparting parallelism, reordering the unknowns changes the effectiveness of Gauß-Seidel as a smoother or as a solver. We note that an appropriate ordering depends on the underlying problem.

Much of the literature approaches multicoloring from the viewpoint of using Gauß-Seidel as either a preconditioner to a Krylov method or as the main solver. The underlying ideas, however, are applicable in the context of smoothing. Multicoloring to achieve parallelism for compact stencils on structured grids has been studied extensively. Perhaps the best known instance of multicolor Gauß-Seidel is the use of two colors for the 5-point Laplace stencil, i.e., red-black Gauß-Seidel [36]. For a rigorous analysis of red-black Gauß-Seidel as a smoother, see for instance

[88, 83]. For the 9-point Laplacian, four colors are sufficient to expose parallelism. See [1, 2], for example. Adams and Jordan [1] analyze multicolor SOR and show that for certain colorings the iteration matrix has the same convergence rate as the iteration matrix associated with the natural lexicographic ordering.

Multicoloring can also be extended to block or line smoothing. Multiple unknowns in a line or block of the computational grid are treated as one unknown and updated simultaneously. Each block is assigned a color in such a way that all the blocks of one color have no dependencies on one another. Because multiple unknowns are updated simultaneously, parallel block smoothing tends to have less interprocessor communication than an exact point Gauß-Seidel method. The convergence rate of multigrid using multicolor block Gauß-Seidel, however, depends on the underlying problem. For a problem without strongly varying coefficients, the convergence rate will tend to be worse than point Gauß-Seidel. For strongly anisotropic problems, however, line smoothing may be necessary for acceptable multigrid convergence.

Block et al. [14], among others, discusses multicolor block Gauß-Seidel as a solver, and provides numerical evidence that the communication overhead is lower for multicolor block Gauß-Seidel. O'Leary [67] shows that for stencils that rely only on eight or fewer nearest neighbors, block colorings exist such that the convergence rate is at least as good as lexicographic ordering. Two color line Gauß-Seidel as a smoother is analyzed in [83].

For unstructured grids, multicoloring can be problematic, as potentially many more colors may be necessary. Adams [4] has implemented a parallel true Gauß-Seidel (i.e., no stale off-processor values) and has shown it to be effective on large 3D unstructured elasticity problems.

Hybrid Gauß-Seidel with Relaxation Weights

The easiest way to implement any smoother in parallel is to just use it independently on each processor, exchanging boundary information after each iteration. We will call such a smoother a *hybrid smoother*. If we use the following terminology for our relaxation scheme:

$$u_{n+1} = u_n + Q^{-1}(f - Au_n), \tag{1.4}$$

Q would be a block diagonal matrix with p diagonal blocks $Q_k, k = 1, ..., p$ for a computer with p processors. For example, if one applies this approach to Gauß-Seidel, Q_k are lower triangular matrices (we call this particular smoother hybrid Gauß-Seidel; it has also been referred to as Processor Block Gauß-Seidel [5]). While this approach is easy to implement, it has the disadvantage of being more similar to a block Jacobi method, albeit worse, since the block systems are not solved exactly. Block Jacobi methods can converge poorly or even diverge unless used with a suitable damping parameter. Additionally, this approach is not scalable, since the number of blocks increases with the number of processors and with it the number of iterations increases. In spite of this, good results can be achieved by setting $Q = (1/\omega)\,\tilde{Q}$ and choosing a suitable relaxation parameter ω. Finding good parameters is not easy and is made even harder by the fact that in a multilevel scheme one deals with a new system on each level, requiring new parameters. It is therefore important to find an automatic procedure to evaluate these parameters. Such a procedure has been developed for symmetric positive problems and smoothers in [87] using convergence theory for regular splittings. A good smoothing parameter for a positive symmetric matrix A is $\omega = 1/\lambda_{max}(\tilde{Q}^{-1}A)$, where $\lambda_{max}(M)$ denotes the maximal eigenvalue of M. A good estimate for this value can be obtained by using a few relaxation steps of Lanczos or conjugate gradient preconditioned with $\tilde{Q}$. In [19] this procedure was applied to hybrid symmetric Gauß-Seidel smoothers within smoothed aggregation. Using the resulting preconditioner to solve several structural mechanics problems led to scalable convergence.

This automatic procedure can also be applied to determine smoothing parameters for any symmetric positive definite hybrid smoother, such as hybrid symmetric Gauß-Seidel, Jacobi, Schwarz smoothers or symmetric positive definite variants of sparse approximate inverse or incomplete Cholesky smoothers.

Polynomial Smoothing

While polynomials have long been used as preconditioners, they have not been as widely used as smoothers in multigrid. The computational kernel of a polynomial is the matrix-vector multiply, which means its effectiveness as a smoother does not degrade as the number of processors increases.

One of the major problems associated with traditional polynomial iterative methods and preconditioners is that it is necessary to have the extremal eigenvalues of the system available. While an estimate of the largest eigenvalue is easily available via either Gershgorin's theorem or a few Lanczos steps, estimating the smallest eigenvalue is more problematic. However, when polynomial methods are used as smoothers this smallest eigenvalue is not really necessary, as only high energy error needs to be damped. Thus, it is often sufficient to take the smallest eigenvalue as a fraction of the largest eigenvalue. Experience has shown that this fraction can be chosen to be the coarsening rate (the ratio of the number of coarse grid unknowns to fine grid unknowns), meaning more aggressive coarsening requires the smoother to address a larger range of high frequencies [5].

Brezina analyzes the use of a polynomial smoother, called MLS smoothing, in the context of smoothed aggregation [20]. This smoother is essentially a combination of two transformed Chebychev polynomials, which are constructed so as to complement one another on the high energy error components [21]. Further analysis can be found in [80, 81].

Adams et al. propose the use of Chebychev polynomials as smoothers in [5]. They show that such smoothers can often be competitive with Gauß-Seidel on serial architectures. These results are different from earlier experiences with Gauß-Seidel and polynomial methods. These differences arise from unstructured mesh considerations, cache effects, and carefully taking advantage of zero initial guesses[5]. In their parallel experiments, better timings were achieved with polynomial smoothers than with basic hybrid Gauß-Seidel smoothers.

[5] The initial guess on the coarse grids is typically zero within a V-cycle. Further, when multigrid is used as a preconditioner, the initial guess is identically zero on the finest mesh.

Sparse Approximate Inverse Smoothing

A sparse approximate inverse is a sparse approximation to the inverse of a matrix. A sparse approximate inverse can be used as a smoother, and can be applied easily in parallel as a sparse matrix-vector product, rather than a triangular solve, for instance. Sparse approximate inverses only have local couplings, making them suitable as smoothers. Other advantages are that more accurate sparse approximate inverse smoothers can be used for more difficult problems, and their performance is not dependent on the ordering of the variables. A drawback of these methods is the relatively high cost of constructing sparse approximate inverses in the general case, compared to the almost negligible cost of setting up Gauß-Seidel. Most studies have focused on very sparse versions that are cheaper to construct.

One common form of the sparse approximate inverse can also be computed easily in parallel. To compute a sparse approximate inverse M for the matrix A, this form minimizes the Frobenius norm of the residual matrix $(I - MA)$. This can be accomplished in parallel because the objective function can be decoupled as the sum of the squares of the 2-norms of the individual rows

$$\|I - MA\|_F^2 = \sum_{i=1}^{n} \|e_i^T - m_i^T A\|_2^2$$

in which e_i^T and m_i^T are the ith rows of the identity matrix and of the matrix M, respectively. Thus, minimizing the above expression is equivalent to minimizing the individual functions

$$\|e_i^T - m_i^T A\|_2, \quad i = 1, 2, \ldots, n.$$

If no restriction is placed on M, the exact inverse will be found. To find an economical sparse approximation, each row in M is constrained to be sparse. A right approximate inverse may be computed by minimizing $\|I - AM\|_F^2$. The left approximate inverse described above, however, is amenable to the common distribution of parallel matrices by rows.

Sparse approximate inverse smoothers were first proposed by Benson [13] in 1990. Tang and Wan [74] discuss the choice of sparsity pattern and least-squares problem to solve to reduce the cost of the smoother. They also analyzed the smoothing factor for constant coefficient PDEs on a two-dimensional regular grid. Some additional theoretical results are given in [23], including for a diagonal approximate inverse smoother, which may be preferable over damped Jacobi. Experimental results in the algebraic multigrid context are given in [22]. Although none of these studies used parallel implementations, parallel implementations of sparse approximate inverses are available [27].

1.3.4 Coarse Grid Parallelism

The solver on the coarsest grid can limit the ultimate speedup that can be attained in a parallel computation, for two related reasons. First, the operator at this level is generally small, and the time required for communication may be higher than the time required to perform the solve on a single processor. Second, the coarsest grid operator may couple all pieces of the global problem (i.e., it is dense, or nearly dense), and thus global communication of the right-hand side or other data may be necessary. For these reasons, parallel multigrid solvers often minimize the time spent on coarse grids, i.e., W-cycles and FMG are avoided.

The coarse grid solver may be a direct solver, an iterative solver, or a multiplication with the full inverse. These will be covered briefly in this subsection. Generally, the parallel performance of the setup or factorization stage of the solver is unimportant, since this phase will be amortized over several solves.

A direct solver is perhaps most often used for the coarsest grid problem. However, the solves with the triangular factors are well-known to be very sequential. If the problem is small enough, instead of solving in parallel, the coarsest grid problem may be factored and solved on a single processor, with the right-hand side gathered, and the solution scattered to the other processors. The other processors may do useful work during this computation. If the other processors have no work and would remain idle, a better option is to solve the coarsest grid problem on all processors. This redundant form of the calculation does not require communication to distribute the result. For an analysis, see [44].

If the coarsest grid problem is too large to fit on a single processor, then there is no choice but to do a parallel computation. However, the communication complexity can be reduced by solving with only a subset of the processors. Solving redundantly with a subset of the processors is again an option. Related are the techniques of [66, 49, 84], where a smaller set of the processors may be used at coarser levels. We note that for difficult problems the final coarse grid may be chosen to be very large, as the error to be reduced becomes more poorly represented on the coarser grids.

Iterative methods for the coarsest grid solve are less sequential, requiring matrix-vector products for each solve. However, since the matrices are quite dense, it is important that very few iterations are required, or the accumulation of communication costs can become very high. To this end, preconditioning may be used, especially since the cost of constructing the preconditioner will be amortized. Similarly, it is advantageous to exploit previous solves with the same matrix, e.g., Krylov subspace vectors from previous solves may be used as an initial approximation space, e.g., [38].

If the coarse grid problems are small enough, another solution strategy is to first compute the inverse of the coarse grid matrix [46, 45]. Each processor stores a portion of the inverse and computes the portion of the solution it requires. The communication in the solution phase requires the right-hand side to be collected at each processor. However, since the inverses are dense, storage will limit the applicability of this method. To alleviate this problem, Fischer [39] has proposed computing a sparse factorization of the inverse of the coarse grid matrix, $A_c^{-1} = XX^T$. The factor X is computed via an A-orthogonalization process, and remains sparse if the order of orthogonalization is chosen according to a nested-dissection ordering. Parallel results with this method were reported in [76, 77].

1.4 Concluding Remarks

We have considered a few of the main research topics associated with the parallelization of multigrid algorithms. These include traditional sources of parallelism such as spatial partitioning as well as non-traditional means of increasing parallelism via multiple coarse grids, concurrent smoothing iterations, and full domain partitioning. We have discussed parallel coarsening and operator complexity issues that arise in both classical algebraic multigrid and agglomeration approaches. Finally, we have discussed parallel smoothers and the coarsest grid solution strategy.

Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Bibliography

[1] L. M. Adams and H. F. Jordan. Is SOR color-blind? *SIAM J. Sci. Statist. Comput.*, 7(2):490–506, 1986.

[2] L. M. Adams, R. J. Leveque, and D. M. Young. Analysis of the SOR iteration for the 9-point Laplacian. *SIAM J. Numer. Anal.*, 25(5):1156–1180, October 1988.

[3] M. F. Adams. A parallel maximal independent set algorithm. In *Proceedings of the 5th Copper Mountain Conference on Iterative Methods*, 1998.

[4] M. F. Adams. A distributed memory unstructured Gauss-Seidel algorithm for multigrid smoothers. In *ACM/IEEE Proceedings of SC2001: High Performance Networking and Computing*, 2001.

[5] M. F. Adams, M. Brezina, J. Hu, and R. Tuminaro. Parallel multigrid smoothing: polynomial versus Gauss-Seidel. *Journal of Computational Physics*, 188:593–610, 2003.

[6] S. F. Ashby and R. D. Falgout. A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations. *Nuclear Science and Engineering*, 124(1):145–159, September 1996.

[7] V. Bandy, J. Dendy, and W. Spangenberg. Some multigrid algorithms for elliptic problems on data parallel machines. *SIAM J. Sci. Stat. Comp.*, 19(1):74–86, 1998.

[8] R. Bank. *PLTMG:A Software Package for Solving Elliptic Partial Differential Equations:User's Guide, 8.0.* SIAM, Philadelphia, 1998.

[9] R. Bank and M. Holst. A new paradigm for parallel adaptive meshing algorithms. *SIAM J. Sci. Stat. Comp.*, 22:1411–1443, 2000.

[10] R. Bank and M. Holst. A new paradigm for parallel adaptive meshing algorithms. *SIAM Review*, 45:291–323, 2003.

[11] R. Bank, S. Lu, C. Tong, and P. Vassilevski. Scalable parallel algebraic multigrid solvers. Technical report, University of California at San Diego, 2004.

[12] P. Bastian, W. Hackbusch, and G. Wittum. Additive and multiplicative multigrid – a comparison. *Computing*, 60:345–364, 1998.

[13] M. W. Benson. Frequency domain behavior of a set of parallel multigrid smoothing operators. *International Journal of Computer Mathematics*, 36:77–88, 1990.

[14] U. Block, A. Frommer, and G. Mayer. Block colouring schemes for the SOR method on local memory parallel computers. *Parallel Computing*, 14:61–75, 1990.

[15] E. F. F. Botta, A. van der Ploeg, and F. W. Wubs. Nested grids ILU-decomposition (NGILU). *J. Comput. Appl. Math.*, 66:515–526, 1996.

[16] E. F. F. Botta and F. W. Wubs. Matrix Renumbering ILU: An effective algebraic multilevel ILU preconditioner for sparse matrices. *SIAM J. Matrix Anal. Appl.*, 20:1007–1026, 1999.

[17] J. Bramble, J. Pasciak, and J. Xu. Parallel multilevel preconditioners. *Math. Comp.*, 55:1–22, 1990.

[18] A. Brandt, S. F. McCormick, and J. W. Ruge. Algebraic multigrid (AMG) for sparse matrix equations. In D. J. Evans, editor, *Sparsity and Its Applications*. Cambridge University Press, 1984.

[19] M. Brezina, , C. Tong, and R. Becker. Parallel algebraic multigrids for structural mechanics. *SIAM Journal of Scientific Computing*, submitted, 2004.

[20] M. Brezina. *Robust iterative solvers on unstructured meshes.* PhD thesis, University of Colorado at Denver, Denver, CO, USA, 1997.

[21] M. Brezina, C. Heberton, J. Mandel, and P. Vaněk. An iterative method with convergence rate chosen a priori. Technical Report UCD/CCM Report 140, University of Colorado at Denver, 1999.

[22] O. Bröker and M. J. Grote. Sparse approximate inverse smoothers for geometric and algebraic multigrid. *Applied Numerical Mathematics*, 41:61–80, 2002.

[23] O. Bröker, M. J. Grote, C. Mayer, and A. Reusken. Robust parallel smoothing for multigrid via sparse approximate inverses. *SIAM Journal on Scientific Computing*, 23:1396–1417, 2001.

[24] P. N. Brown, R. D. Falgout, and J. E. Jones. Semicoarsening multigrid on distributed memory machines. *SIAM J. Sci. Comput.*, 21(5):1823–1834, 2000.

[25] T. Chan and R. Tuminaro. Analysis of a parallel multigrid algorithm. In S. McCormick, editor, *Proceedings of the Fourth Copper Mountain Conference on Multigrid Methods*, NY, 1987. Marcel Dekker.

[26] T. Chan and R. Tuminaro. A survey of parallel multigrid algorithms. In A. Noor, editor, *Proceedings of the ASME Syposium on Parallel Computations and their Impact on Mechanics*, volume AMD-Vol. 86, pages 155–170. The American Society of Mechanical Engineers, 1987.

[27] E. Chow. Parallel implementation and practical use of sparse approximate inverses with a priori sparsity patterns. *Intl. J. High Perf. Comput. Appl.*, 15:56–74, 2001.

[28] E. Chow and P. S. Vassilevski. Multilevel block factorizations in generalized hierarachical bases. *Num. Lin. Alg. Appl.*, 10:105–127, 2003.

[29] A. J. Cleary, R. D. Falgout, V. E. Henson, and J. E. Jones. Coarse-grid selection for parallel algebraic multigrid. In *Proc. of the Fifth International Symposium on: Solving Irregularly Structured Problems in Parallel*, volume 1457 of *Lecture Notes in Computer Science*, pages 104–115, New York, 1998. Springer–Verlag.

[30] H. De Sterck, U. M. Yang, and J. Heys. Reducing complexity in parallel algebraic multigrid preconditioners. *SIAM Journal on Matrix Analysis and Applications*, submitted, 2004.

[31] J. Dendy. Revenge of the semicoarsening frequency decomposition method. *SIAM J. Sci. Stat. Comp.*, 18:430–440, 1997.

[32] J. Dendy and C. Tazartes. Grandchild of the frequency decomposition method. *SIAM J. Sci. Stat. Comp.*, 16:307–319, 1995.

[33] J. E. Dendy, M. P. Ida, and J. M. Rutledge. A semicoarsening multigrid algorithm for SIMD machines. *SIAM J. Sci. Stat. Comput.*, 13:1460–1469, 1992.

[34] K. Devine, B. Hendrickson, E. Boman, M. St. John, and C. Vaughan. Zoltan: A dynamic load-balancing library for parallel applications; user's guide. Technical Report SAND99-1377, Sandia National Laboratories, 1999.

[35] C. Douglas and W. Miranker. Constructive interference in parallel algorithms. *SIAM Journal on Numerical Analysis*, 25:376–398, 1988.

[36] D. J. Evans. Parallel S.O.R. iterative methods. *Parallel Computing*, 1:3–18, 1984.

[37] R. D. Falgout and J. E. Jones. Multigrid on massively parallel architectures. In E. Dick, K. Riemslagh, and J. Vierendeels, editors, *Multigrid Methods VI*, volume 14 of *Lecture Notes in Computational Science and Engineering*, pages 101–107. Springer–Verlag, 2000.

[38] C. Farhat and P. S. Chen. Tailoring domain decomposition methods for efficient parallel coarse grid solution and for systems with many right hand sides. *Contemporary Mathematics*, 180:401–406, 1994.

[39] P. F. Fischer. Parallel multi-level solvers for spectral element methods. In R. Scott, editor, *Proceedings of International Conference on Spectral and High Order Methods '95*, pages 595–604, 1996.

[40] L. Fournier and S. Lanteri. Multiplicative and additive parallel multigrid algorithms for the acceleration of compressible flow computations on unstructured meshes. *Applied Numerical Mathematics*, 36(4):401–426, 2001.

[41] P. Frederickson and O. McBryan. Parallel superconvergent multigrid. In S. McCormick, editor, *Proceedings of the Third Copper Mountain Conference on Multigrid Methods*, pages 195–210, NY, 1987. Marcel Dekker.

[42] D. Gannon and J. Van Rosendale. On the structure of parallelism in a highly concurrent PDE solver. *Journal of Parallel and Distributed Computing*, 3:106–135, 1986.

[43] A. Greenbaum. A multigrid method for multiprocessors. In S. McCormick, editor, *Proceedings of the Second Copper Mountain Conference on Multigrid Methods*, volume 19 of *Appl. Math and Computation*, pages 75–88, 1986.

[44] 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, editors, *Fifth Conference on Domain Decomposition Methods for Partial Differential Equations*, pages 349–362. SIAM, 1992.

[45] W. D. Gropp and D. E. Keyes. Domain decomposition methods in computational fluid dynamics. *International Journal for Numerical Methods in Fluids*, 14:147–165, 1992.

[46] W. D. Gropp and D. E. Keyes. Domain decomposition with local mesh refinement. *SIAM Journal on Scientific and Statistical Computing*, 15:967–993, 1992.

[47] W. Hackbusch. A new approach to robust multi-grid methods. In *First International Conference on Industrial and Applied Mathematics*, Paris, 1987.

[48] W. Hackbusch. The frequency decomposition multigrid method, part I: Application to anisotropic equaitons. *Numer. Math.*, 56:229–245, 1989.

[49] R. Hempel and A. Schüller. Experiments with parallel multigrid algorithms using the SUPRENUM communications subroutine library. Technical Report 141, GMD, Germany, 1988.

[50] B. Hendrickson and R. Leland. A user's guide to Chaco, Version 1.0. Technical Report SAND93-2339, Sandia National Laboratories, 1993.

[51] V. E. Henson and U. M. Yang. BoomerAMG: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41:155–177, 2002.

[52] J. E. Jones and S. F. McCormick. Parallel multigrid methods. In Keyes, Sameh, and Venkatakrishnan, editors, *Parallel Numerical Algorithms*, pages 203–224. Kluwer Academic, 1997.

[53] M. Jones and P. Plassmann. A parallel graph coloring heuristic. *SIAM J. Sci. Comput.*, 14:654–669, 1993.

[54] W. Joubert and J. Cullum. Scalable algebraic multigrid on 3500 processors. *Electronic Transactions on Numerical Analysis*, submitted, 2003. Los Alamos National Laboartory Technical Report No. LAUR03-568.

[55] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. Technical Report 95-064, Army HPC Research Center Technical Report, 1995.

[56] A. Krechel and K. Stüben. Parallel algebraic multigrid based on subdomain blocking. *Parallel Computing*, 27:1009–1031, 2001.

[57] Z. Li, Y. Saad, and M. Sosonkina. pARMS: a parallel version of the algebraic recursive multilevel solver. *Numerical Linear Algebra with Applications*, 10:485–509, 2003.

[58] S. Lu. *Scalable Parallel Multilevel Algorithms for Solving Partial Differential Equations*. PhD thesis, University of California at San Diego, 2004.

[59] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. on Computing*, 15:1036–1053, 1986.

[60] D. J. Mavriplis. Parallel performance investigations of an unstructured mesh Navier-Stokes solver. *Intl. J. High Perf. Comput. Appl.*, 16:395–407, 2002.

[61] O. A. McBryan, P. O. Frederickson, J. Linden, A. Schüller, K. Stüben, C.-A. Thole, and U. Trottenberg. Multigrid methods on parallel computers—a survey of recent developments. *Impact of Computing in Science and Engineering*, 3:1–75, 1991.

[62] W. Mitchell. A parallel multigrid method using the full domain partition. *Electron. Trans. Numer. Anal.*, 6:224–233, 1998.

[63] W. Mitchell. Parallel adaptive multilevel methods with full domain partitions. *App. Num. Anal. and Comp. Math.*, 1:36–48, 2004.

[64] W. Mulder. A new multigrid approach to convection problems. *J. Comput. Phys.*, 83:303–329, 1989.

[65] N. Naik and J. Van Rosendale. The improved robustness of multigrid solvers based on multiple semicoarsened grids. *SIAM J. Numer. Anal.*, 30:215–229, 1993.

[66] V. K. Naik and S. Ta'asan. Performance studies of multigrid algorithms implemented on hypercube multiprocessor systems. In M. Heath, editor, *Proceedings of the Second Conference on Hypercube Multiprocessors*, pages 720–729, Philadelphia, 1987. SIAM.

[67] D. P. O'Leary. Ordering schemes for parallel processing of certain mesh problems. *SIAM J. Sci. Stat. Comp.*, 5:620–632, 1984.

[68] J. W. Ruge and K. Stüben. Algebraic multigrid (AMG). In S. F. McCormick, editor, *Multigrid Methods*, volume 3 of *Frontiers in Applied Mathematics*, pages 73–130. SIAM, Philadelphia, PA, 1987.

[69] Y. Saad. ILUM: A multi-elimination ILU preconditioner for general sparse matrices. *SIAM J. Sci. Comput.*, 17:830–847, 1996.

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

[71] B. Smith, P. Bjorstad, and W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

[72] K. Stüben. Algebraic multigrid (AMG): an introduction with applications. In A. Schüller U. Trottenberg, C. Oosterlee, editor, *Multigrid*. Academic Press, 2000.

[73] J. Swisshelm, G. Johnson, and S. Kumar. Parallel computation of Euler and Navier-Stokes flows. In S. McCormick, editor, *Proceedings of the Second Copper Mountain Conference on Multigrid Methods*, volume 19 of *Appl. Math. and Computation*, pages 321–331, 1986.

[74] W.-P. Tang and W. L. Wan. Sparse approximate inverse smoother for multigrid. *SIAM Journal on Matrix Analysis and Applications*, 21:1236–1252, 2000.

[75] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2000.

[76] H. M. Tufo and P. F. Fischer. Terascale spectral element algorithms and implementations. In *Proceedings of SC99*, 1999.

[77] H. M. Tufo and P. F. Fischer. Fast parallel direct solvers for coarse grid problems. *J. Par. & Dist. Computing*, 61:151–177, 2001.

[78] R. Tuminaro. A highly parallel multigrid-like algorithm for the Euler equations. *SIAM J. Sci. Comput.*, 13(1), 1992.

[79] R. Tuminaro and C. Tong. Parallel smoothed aggregation multigrid: aggregation strategies on massively parallel machines. In J. Donnelley, editor, *Supercomputing 2000 Proceedings*, 2000.

[80] P. Vaněk, M. Brezina, and J. Mandel. Convergence of algebraic multigrid based on smoothed aggregation. *Numerische Mathematik*, 88:559–579, 2001.

[81] P. Vaněk, M. Brezina, and R. Tezaur. Two-grid method for linear elasticity on unstructured meshes. *SIAM J. Sci. Comp.*, 21:900–923, 1999.

[82] P. Vaněk, J. Mandel, and M. Brezina. Algebraic multigrid based on smoothed aggregation for second and fourth order problems. *Computing*, 56:179–196, 1996.

[83] P. Wesseling. *An Introduction to Multigrid Methods*. John Wiley & Sons, Chichester, 1992. Reprinted by R.T. Edwards, Inc., 2004.

[84] D. E. Womble and B. C. Young. A model and implementation of multigrid for massively parallel computers. *Int. J. High Speed Computing*, 2(3):239–256, September 1990.

[85] S. Xiao and D. Young. Multiple coarse grid multigrid methods for solving elliptic problems. In N. Melson, T. Manteuffel, and S. McCormick C. Douglas, editors, *Proceedings of the Seventh Copper Mountain Conference on Multigrid Methods*, volume 3339 of *NASA Conference Publication*, pages 771–791, 1996.

[86] J. Xu. *Theory of Multilevel Methods*. PhD thesis, Cornell University, 1987.

[87] U. M. Yang. On the use of relaxation parameters in hybrid smoothers. *Numerical Linear Algebra with Applications*, 11:155–172, 2004.

[88] I. Yavneh. On red-black SOR smoothing in multigrid. *SIAM J. Sci. Comp.*, 17:180–192, 1996.

[89] D. M. Young. *Iterative Methods for Solving Partial Difference Equations of Elliptic Type*. PhD thesis, Harvard University, Cambridge, MA, USA, May 1950.

[90] H. Yserentant. On the multi-level splitting of finite element spaces. *Numer. Math.*, 49:379–412, 1986.