BoomerAMG: a Parallel Algebraic Multigrid Solver and Preconditioner

Van Emden Henson and Ulrike Meier Yang

Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA, 94550. Email: {vhenson, umyang}@llnl.gov

DEDICATED TO THE MEMORY OF RÜDIGER WEISS

Abstract

Driven by the need to solve linear systems arising from problems posed on extremely large, unstructured grids, there has been a recent resurgence of interest in algebraic multigrid (AMG). AMG is attractive in that it holds out the possibility of multigrid-like performance on unstructured grids. The sheer size of many modern physics and simulation problems has led to the development of massively parallel computers, and has sparked much research into developing algorithms for them. Parallelizing AMG is a difficult task, however. While much of the AMG method parallelizes readily, the process of coarse-grid selection, in particular, is fundamentally sequential in nature.

We have previously introduced a parallel algorithm [7] for the selection of coarse-grid points, based on modifications of certain parallel independent set algorithms and the application of heuristics designed to insure the quality of the coarse grids, and shown results from a prototype serial version of the algorithm.

In this paper we describe an implementation of a parallel AMG code, using the algorithm of [7] as well as other approaches to parallelizing the coarse-grid selection. We consider three basic coarsening schemes and certain modifications to the basic schemes, designed to address specific performance issues. We present numerical results for a broad range of problem sizes and descriptions, and draw conclusions regarding the efficacy of the method. Finally, we indicate the current directions of the research.

Key words: algebraic multigrid; parallel computing

[1] 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.

1 Introduction

Algebraic multigrid (AMG) was introduced in the 1980's [3,1,2,4,18,14,16,15] and immediately attracted the attention of scientists needing to solve large problems posed on unstructured grids. In the past several years there has been a major surge of interest in AMG. Much of the current research focuses either on improving the standard AMG algorithm [9,8] or on dramatic new algebraic approaches [19,5].

Currently, there is great interest in finding effective ways of applying AMG to extremely large problems, involving millions or even billions of unknowns. Necessarily, this implies finding ways of implementing AMG on massively parallel computers with distributed memory hierarchies. Of particular concern is the *scalability* of the method. Roughly speaking, a method is scalable if the time required to produce the solution remains essentially constant as both the problem size and the computing resources increase. Iterative methods are generally scalable *per iteration step*, but for most iterative methods we observe that the number of iterations required to solve a system is dependent on the size of the system. Many geometric multigrid methods are known to be scalable, both in the per-iteration sense and in the number of iterations required to solve the system. Much of the interest in AMG comes from the hope that the scalability of geometric multigrid methods can be obtained for large unstructured grid problems if an effective parallel AMG can be devised.

To date, however, relatively little research has been done on parallelizing AMG [12]. Techniques for parallelizing *geometric* multigrid methods have been known for some time [10]. Fortuitously, most of the AMG algorithm can be parallelized using this existing technology. But the heart of the AMG setup phase includes the coarse-grid selection process. In the classic AMG development [15] this process is inherently sequential in nature.

In this paper we describe the parallel AMG code *BoomerAMG*. This code has been implemented on several massively parallel machines and has been tested on problems involving tens of millions of unknowns and over a thousand processors. As noted above, most of the code can be constructed using a straightforward modification of existing parallel technology. We describe in detail various parallel algorithms for selecting the coarse-grid points. One such algorithm is based on modifications of certain parallel *independent set* algorithms. Also, we employ heuristics designed to insure the quality of the coarse grids.

In Section 2, we outline the basic principles of AMG. We describe the classical (sequential) coarsening algorithm in Section 3, and give two basic approaches to the parallel coarse-grid selection in Section 4, along with some important modifications to these methods. In Section 5 we describe the interpolation we use. Results of numerical experiments are presented and discussed in Section 6. In Section 7, we make concluding remarks and indicate directions for future research.

2 Algebraic Multigrid

We begin by outlining the basic principles and techniques that comprise AMG. Detailed explanations may be found in [15]. Consider a problem of the form

$$A\mathbf{u} = \mathbf{f}, \tag{1}$$

where A is an $n \times n$ matrix with entries a_{ij}. For convenience, the indices are identified with grid points, so that u_i denotes the value of $\mathbf{u}$ at point i, and the grid is denoted by $\Omega = \{1, 2, \ldots, n\}$. In any multigrid method, the central idea is that "smooth error," $\mathbf{e}$, that is not eliminated by relaxation must be removed by coarse-grid correction. This is done by solving the residual equation $A\mathbf{e} = \mathbf{r}$ on a coarser grid, then interpolating the error back to the fine grid and using it to correct the fine-grid approximation by $\mathbf{u} \leftarrow \mathbf{u} + \mathbf{e}$.

Using superscripts to indicate level number, where 1 denotes the finest level so that $A^1 = A$ and $\Omega^1 = \Omega$, the components that AMG needs are as follows:

(1) "Grids" $\Omega^1 \supset \Omega^2 \supset \ldots \supset \Omega^M$.
(2) Grid operators $A^1, A^2, \ldots, A^M$.
(3) Grid transfer operators:
    Interpolation $I_{k+1}^k, k = 1, 2, \ldots M-1$,
    Restriction $I_k^{k+1}, k = 1, 2, \ldots M-1$.
(4) Relaxation scheme for each level.

Once these components are defined, the recursively defined cycle is as follows:

```
Algorithm: $MV^k(\mathbf{u}^k, \mathbf{f}^k)$. The $(\mu_1, \mu_2)$ V-cycle.
   If $k = M$, set $\mathbf{u}^M = (A^M)^{-1}\mathbf{f}^M$.
   Otherwise:
      Relax $\mu_1$ times on $A^k\mathbf{u}^k = \mathbf{f}^k$ .
      Perform coarse grid correction:
         Set $\mathbf{u}^{k+1} = 0, \mathbf{f}^{k+1} = I_k^{k+1}(\mathbf{f}^k - A^k\mathbf{u}^k)$.
         "Solve" on level $k+1$ with $MV^{k+1}(\mathbf{u}^{k+1}, \mathbf{f}^{k+1})$.
         Correct the solution by $\mathbf{u}^k \leftarrow \mathbf{u}^k + I_{k+1}^k\mathbf{u}^{k+1}$.
      Relax $\nu_2$ times on $A^k\mathbf{u}^k = \mathbf{f}^k$.
```

The choice of components in AMG is done in a separate preprocessing step, known as the *setup phase.*

AMG Setup Phase:

(1) Set $k = 1$.
(2) Partition Ω^k into disjoint sets C^k and F^k.
  (a) Set $\Omega^{k+1} = C^k$.
  (b) Define interpolation I_{k+1}^k.
(3) Set $I_k^{k+1} = \left(I_{k+1}^k\right)^T$ and $A^{k+1} = I_k^{k+1} A^k I_{k+1}^k$.
(4) If Ω^{k+1} is small enough, set $M = k + 1$ and stop. Otherwise, set $k = k + 1$ and go to step 2.

3 Coarse-Grid Selection

Coarse-grid selection in the classical algorithm comprises two stages. In the first, we make an initial partitioning of the grid points into C- and F-points, based on an indicator of a point's suitability to be a C-point. Then, as the interpolation operator is constructed, we make adjustments to this partitioning, changing points initially chosen as F-points to be C-points in order to insure that the partitioning conforms to certain heuristic rules.

Since the focus here is on coarsening a particular level, k, such superscripts are omitted here. The goal of the setup phase is to choose the set C of coarse-grid points and, for each fine-grid point $i \in F \equiv \Omega - C$, a small set $C_i \subset C$ of interpolating points.

To select the coarse-grid points, we seek those unknowns u_i which can be used to represent the values of nearby unknowns u_j. This gives rise to the concepts of *dependence* and *influence*. We say that the point i depends on the point j if the value of the unknown u_j is important in determining the value of u_i from the ith equation. In that case we also say that j influences i. We denote by S_i the set of of points on which a point i depends. The definition used in classical AMG is

$$S_i \equiv \left\{ j \neq i : -a_{ij} \geq \alpha \max_{k \neq i} (-a_{ik}) \right\}, \tag{2}$$

with α typically set to be 0.25. We also define the set $S_i^T \equiv \{j : i \in S_j\}$, that is, the set of points j that are influenced by i.

We try to adhere to two criteria while choosing C and F:

C1: For each $i \in F$, each $j \in S_i$ is either in C or $S_j \cap C_i \neq \emptyset$.

C2: C should be a maximal subset with the property that no point in C depends on another point in C.

C1 is designed to insure that the value of u_j is represented in the interpolation formula for u_i if i strongly depends on j, even when j is not a C-point. **C2** is designed to strike a balance on the size of the coarse grid. If the coarse grid is a large fraction of the total points, then the interpolation of smooth errors is likely to be very accurate, which, in turn, generally produces better convergence factors. However, relatively large coarse grids generally mean a prohibitively large amount of work in doing V-cycles.

It is not always possible to enforce both **C1** and **C2**. Because the classical definition of interpolation depends on **C1** being satisfied, we choose to enforce **C1** rigorously, while using **C2** as a guide. While this choice may lead to larger coarse grids than necessary, experience shows that this trade-off between accuracy and expense is generally worthwhile.

The first pass begins by assigning to each point i the number, λ_i, of other points strongly influenced by i. We then select a point with maximal λ as the first point in C. The points that depend strongly on i then become F-points. Since all other points that strongly influence these new F-points are potential C-points, for each new F-point j in S_i^T, we increment λ_k of each unassigned member $k \in S_j$. The process is then repeated until all points are assigned to C or F. Fig. 1 shows the coarse grid selection for the 9-point Laplacian operator with Dirichlet boundary conditions on a uniform grid.

While coarse-grid selection is complete after the first pass in this example, it is not difficult to concoct an example that does not work so well. Fig. 2 shows the result of the first pass of the coarse-grid selection for the nine-point Laplacian stencil on a uniform grid with periodic boundary conditions. The final coloring (left in figure) violates **C1**, with a large number of F-F dependencies between points not sharing a C-point.

To alleviate this a second pass for the coarsening algorithm is done, in which each of the F-points is examined in turn; if there are F-F dependencies with points not depending on a common C-point, one of the two F-points is tentatively changed into a C-point.

The diagram on the right of Figure 2 displays the coarsening produced for the periodic nine-point Laplacian after the second pass of the coloring algorithm. The extra C-points are shaded black with a rim of gray.

Two important measures are used in judging the quality of a coarsening algorithm. One obvious measure is the asymptotic convergence factor (per V-cycle) that results. The second measure is *operator complexity*, defined as the ratio of the total number of nonzero entries in the operator matrices A^k, for all

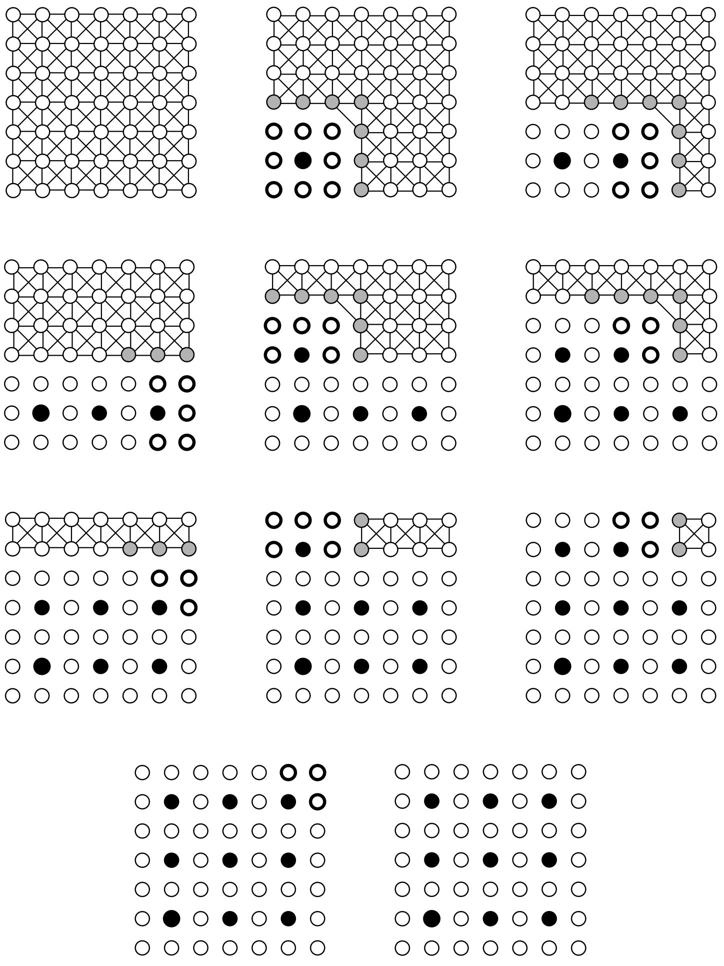

Fig. 1. *Sequence of coloring steps for the nine-point Laplacian on a uniform grid. The upper left diagram is the original grid, the lower right the final coloring.*

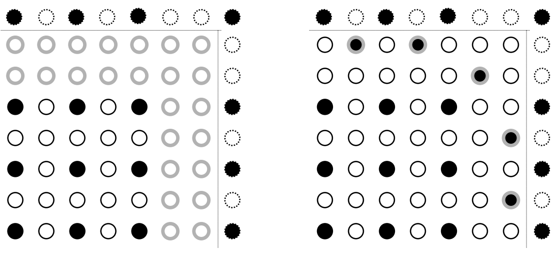

Fig. 2. Left: *Result of first coloring pass for the nine-point Laplacian problem with periodic boundary conditions.* Right: *Additional points added in second coloring pass. The added C-points are shown in the final coloring as black dots with thick gray outlines.*

levels k, to the number of nonzeros in the fine-grid operator $A = A^0$. Operator complexity gives a measure of the storage cost, and in addition measures the cost of the V-cycle in terms of operation count, as the most expensive work, relaxation, is proportional to the number of nonzero entries in the operator matrices. These measures are also influenced heavily by other parts of the AMG algorithm; the effectiveness of relaxation and the quality of the interpolation heavily influence the convergence factor, while the choice of interpolation operator affects operator complexity.

The classical coarsening algorithm tends to produce very good coarsening when applied to standard geometric problems [15,8]. In particular, it has the property that it tends to coarsen in the direction of dependence even in cases of anisotropic operators and discontinuous coefficients [6]. In the standard model problems it produces the same coarse grids, operator and grid complexities, and convergence factors that are typically seen in geometric methods [6], see for example Figure 4a., which shows the coarsening for a 5-point Laplacian problem on a regular grid, where C-points are gray and F-points are black.

4 Parallel Selection of Coarse Grids

Several strategies can be employed to parallelize coarse-grid selection for algebraic multigrid; the choice of parallelization strategy depends largely on the specific goals of the user. For example, it may be desired to select a coarsening that leads to maximal cycling efficiency as measured by convergence factor. This may lead to choices that make the coarse-grid selection expensive both in terms of time and operator complexity. Another choice might be to minimize grid and operator complexity because of storage constraints. This may lead to algorithms that are less efficient in terms of convergence factor (and hence, time to convergence). Some users require that the same sequence of coarse grids (and convergence results) be produced independent of the number of processors employed.

We have devised and implemented several different parallel coarsening schemes to meet some of the goals described above. In the next several subsections we describe the coarsening schemes, and main modifications to them.

4.1 The CLJP Algorithm

We describe first the Cleary-Luby-Jones-Plassman (CLJP) parallel coarsening algorithm. This coarsening was proposed by Cleary [7], and is based on parallel graph partitioning algorithms introduced by Luby [13] and developed by Jones and Plassman [11].

We begin by defining S, the auxiliary *influence matrix*:

$$S_{ij} = \begin{cases} 1 & \text{if } \ j \in S_i, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

That is, $S_{ij} = 1$ only if i depends on j. The ith row of S gives S_i, the set of dependencies of i, while the ith column of S gives S_i^T, the influences of i. We can then form the directed graph of S, and observe that a directed edge from vertex i to vertex j exists only if $S_{ij} \neq 0$.

To each point i we define a measure $w(i) = |S_i^T| + \sigma(i)$, the number of points influenced by the point i plus a random number in $(0, 1)$. The random number is used as a mechanism for breaking ties between points with the same number of influences. We then select a set D of points where the point i is placed in the set D if $w(i) > w(k)$ for all $k \in S_i \cap S_i^T$ (all points that either influence or depend on i). By construction, this set will be independent.

Once the independent set D is chosen, we modify the graph according to the following pair of heuristics, which are designed (like **C1** and **C2**) to ensure the quality of the coarse-grid while controlling its size.

H1: Values at C-points are not interpolated; hence, neighbors that influence a C-point are less valuable as potential C-points themselves.

H2: If k and j both depend on i, a given C-point, and j influences k, then j is less valuable as a potential C-point, since k can be interpolated from i.

We implement these heuristics as follows:

```
for each $i \in D$,
    for each $j$ that influences $i$,
        decrement $w(j)$
        set $S_{ij} \leftarrow 0$ (remove edge $ij$ from the graph)

    for each $j$ that depends on $i$,
        set $S_{ji} \leftarrow 0$ (remove edge $ji$ from the graph)
        for each $k$ that depends on $j$,
        if $k$ depends on $i$,
            decrement $w(j)$
            set $S_{kj} \leftarrow 0$ (remove edge $kj$ from the graph)
```

The heuristics have the effect of lowering the measure w for a set of neighbors of each point in D. As these measures are lowered, edges of the graph of S are removed to indicate that certain influences have already been taken into account. Whenever the decrementing of $w(j)$ results in $w(j) < 1$, point j is flagged as an F-point.

Once the heuristics have been applied for all the points in D, the points in D are designated as C-points and a global communication step is required, so that each processor has updated w values for all their "ghost" neighbors. The entire process is then repeated, by picking a new set D from the vertices of the modified graph of S and performing the heuristics on the new set D. The process repeats until all points have either been selected as C- or F-points. An example of the application of the CLJP coarsening, applied to the 9-point Laplacian operator on a regular grid, is displayed in Figure 3.

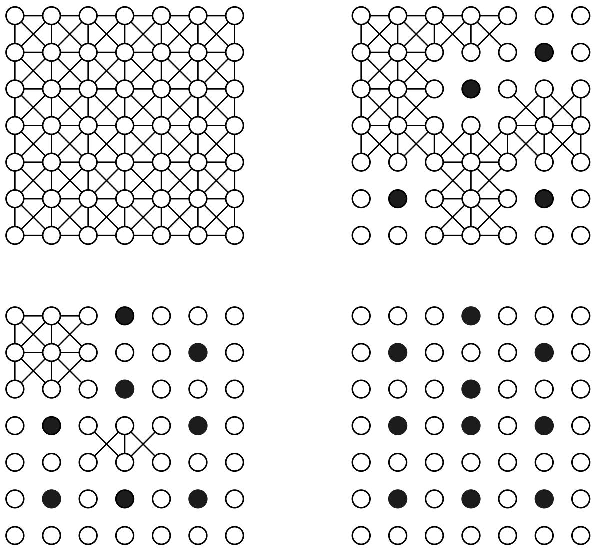

Fig. 3. *Sequence of coloring steps using the CLJP algorithm for the nine-point Laplacian on a uniform grid.* Top Left: *the original dependence graph;* Top Right: *the first independent set and the graph remaining after the application of the heuristics.* Lower Left: *the graph after the second independent has been chosen and the heuristics performed;* Lower Right: *the final coloring.*

The main advantage of the CLJP coarsening is that it is entirely parallel, and results in the same selection of coarse-grid points (given the same global set of random numbers) regardless of the number of processors involved. It also behaves well as the grids become increasingly coarse; processors can "shut down" when no points remain to be coarsened in their domain without necessitating extra communication or special treatment. Our experience, which will be detailed in Section 6, is that the CLJP coarsening tends to pick coarse grids with more points than are necessary, resulting in large complexities. This is illustrated in Figure 4b. where CLJP is applied to a 5-point Laplacian on a 10x10 grid and generates 58 C-points compared to 50 C-points for the standard coarsening which is shown in part a. of Figure 4.

4.2 Parallel Ruge-Stüben Coarsening

A relatively natural approach to take is to use the standard Ruge-Stüben (RS) approach described in the previous section. In this case each processor employs the RS method on all of the points it holds. Choices must be made regarding

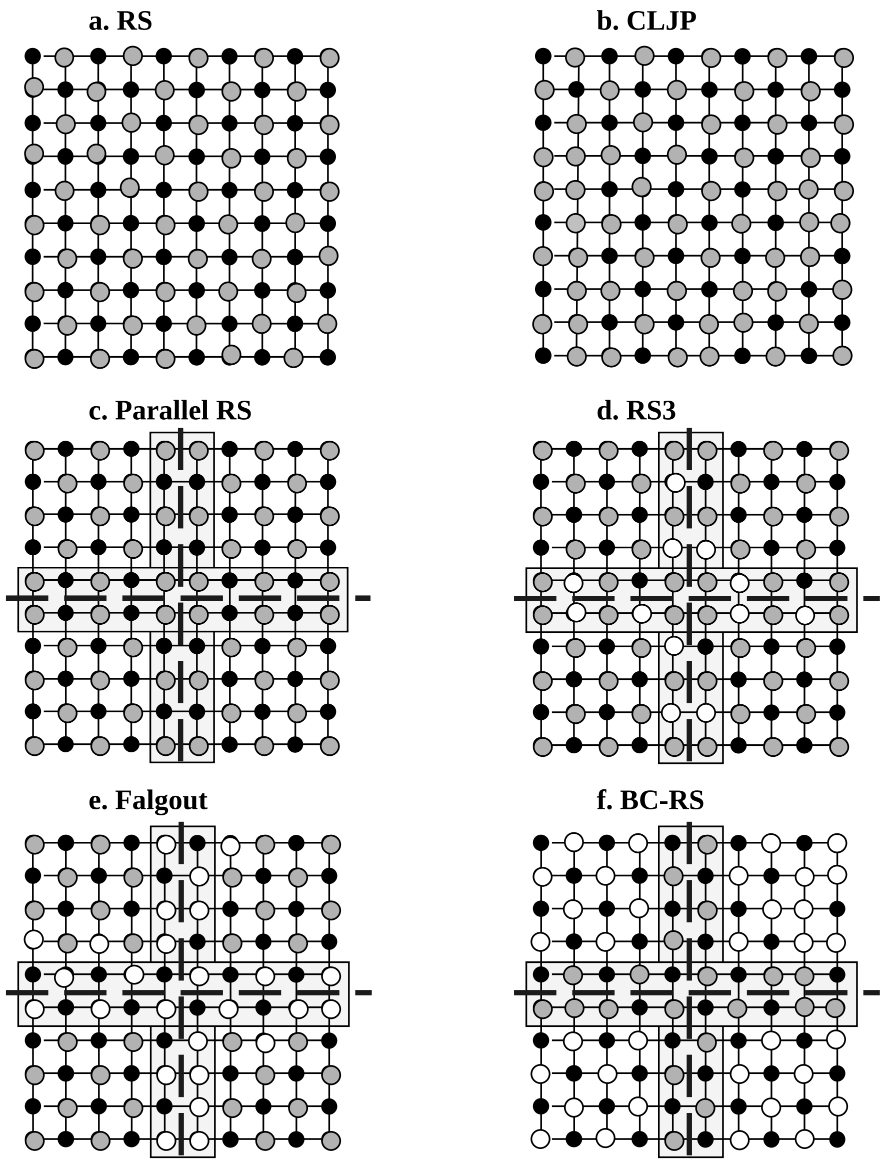

Fig. 4. *Various coarsenings demonstrated for the five-point Laplacian on a uniform* 10×10 *grid using 4 processors for coarsenings c.-f.*

how to treat the coarsening at the processor boundaries. There are several variations that can be employed. One feature common to all variations is that the resulting coarsening will depend on the number of processors; hence, it is not possible for the same coarsening to be achieved with differing numbers of processors.

The simplest such method is to have each processor perform both the first and second pass of the RS algorithm on their local data without any special treatment at all on the processor boundaries. It is highly probable that this method will result in F-F dependencies across processor boundaries where the points in question do not share a common C-point, violating **C1** (for an example see Figure 4c. where this approach is applied to a 5-point Laplacian on a 10×10 grid using 4 processors).

When employing the RS algorithm the measure $w(i)$, for points i that are adjacent to the processor boundary, can be computed either locally or globally. Local measures comprise only those connections to points residing on the processor holding i, while global measures also account for off-processor connections. The "local measures only" method we denote as "RS" while, if global measures are used, we denote the method "RSgm." For our small example, the 5-point Laplacian on a 10×10 grid using 4 processors RSgm generates the same coarsening as the classical RS algorithm on one processor, thus preserving **C1**, however it is not hard to see that choosing a different grid size (e.g. 12×12) will reverse this picture, and now parallel RS would lead to the same coarsening as the classical RS algorithm, while RSgm would lead to a coarsening similar to the one shown in Figure 4c., which violates **C1**.

A natural modification to RS coarsening is to perform some kind of boundary "fix-up" to correct for the problems entailed by having F-F dependencies across processor boundaries where the points involved do not depend on a mutual C-point. Again, several methods can be used to correct the initial coarsening. Our experience favors an approach we denote as RS3, standing for Ruge-Stüben third pass coarsening. Essentially, after each processor independently performs a first and second pass of RS coarsening, each processor performs a third pass on its boundary points and the associated ghost points. The so-called third pass is actually a second application of the "second pass" heuristic, but only the boundary and ghost points are involved. After the extra pass we still may have conflicting coarsenings proposed by the processors on either side of the boundary. Some strategy must be employed to resolve the conflict. For example, one might choose to make a C-point out of any point proposed as a C-point by any of the processors involved. This leads to densely packed C-points along processor boundaries. At the other extreme is to select as a C-point only those points nominated by all processors having them as points or ghost points. This choice tends to underselect, leaving F-F dependencies without common C-points, violating **C1**. A third choice could be to use the C-point selection of the highest numbered processor involved. This however would still leave untreated F-F dependencies. In our codes, a combination of the first and the third choice was used, i.e. each processor accepts the C-point choice made by processors with higher numbers while keeping its own C-points. This process leads to a smaller number of C-points than the first choice while not violating **C1**. One potential problem of this choice is that this can lead to load imbalance, since it will generate a denser C-point packing on the boundaries of processors with a lower number. Nevertheless, it appeared to be the best choice in our experiments.

Figure 4d. displays the effect of the RS3 process. The gray points are C-points generated in the first two passes, while the white points are C-points that are added during the third pass. The effect of the RS3 coarsening is to produce more C-points along processor boundaries. This increases the complexity, and, as the grids become coarser, tends to increase the "surface-to-volume ratio" of boundary points to total points on each processor. However, for many problems RS3 also leads to noticeably better convergence factors (see Section 6).

A drawback to both the RS and RS3 coarsenings is that coarsening each processor independently cannot proceed once each processor has attained a "coarsest" possible grid, i.e., a single gridpoint. However, if the problem is run using thousands or tens of thousands of processors this implies that the global coarse grid will contain thousands or tens of thousands of points. In such cases a direct solution of the coarse-grid problem can require an exhorbitant amount of time, memory, communication, and computation.

One solution to this problem is to collect the coarsest-grid points onto a subset of the processors (or a single processor), continue coarsening on those processors, solve the coarse-grid problem, and propagate the solution back out to all the remaining processors. A related approach is to copy the coarsest-grid points onto *all* processors, continue coarsening until a sufficiently small coarsest grid is reached, solve this coarse-grid problem, and propagate the solution to all processors. While this latter approach still requires a global communication step when the coarse-grid information is assembled on the processors, it eliminates a global communication required when the coarse-grid solution is propagated to finer grid levels.

Our approach to alleviating this difficulty is to switch from the RS (or RS3) coarsening scheme to the CLJP coarsening, when coarsening slows down. Effectively, this implies that the first several coarse grids are selected using an RS approach, but that the last few levels are selected using CLJP. Since the CLJP method can coarsen all the way to a single coarse-grid point, we are free to decide at what grid size to stop coarsening.

4.3 Falgout Coarsening

We propose another coarsening scheme, which is a hybrid RS/CLJP scheme, the so-called *Falgout coarsening.* The RS or RS3 scheme described in the previous paragraph allows coarsening to proceed to a very coarse grid, but still faces the processor-boundary difficulties of C-point packing and, in the case of RS, of untreated F-F dependencies. Both of these phenomena are avoided in CLJP coarsening; however, CLJP tends to give somewhat poorer coarsenings (in general) on the interiors of the processor domains. The concept behind Falgout coarsening is relatively simple – use RS coarsening on the interior of the processor domains while using CLJP coarsening near the processor boundaries.

The method proceeds as follows. Each processor performs RS coarsening on the points it holds. The C-points selected in this process, for all points *except* those adjacent to a processor boundary, are used as the first independent set (D) in the CLJP algorithm. The remainder of the coarsening is done simply by continuing the CLJP algorithm. Hence, the method differs from CLJP coarsening only in the manner in which the first independent set D, on each grid level, is selected. Since the bulk of the C-point selection occurs in the processor domain interiors, it stands to reason that the resulting coarsening will be more like the RS coarsening than the CLJP scheme. However, because the boundary points are not selected in the initial set D, treatment of processor-boundary points is done entirely using the CLJP scheme, which avoids untreated F-F dependencies and diminishes the boundary-packing problems.

Figure 4e. shows how the Falgout coarsening treats the 5-point Laplacian on a 10×10 grid using 4 processors. The gray points denote C-points that were generated during the RS phase of the algorithm, while the white C-points were added during the CLJP phase. Note that here, different than in the RS3 coarsening, addition of C-points is not restricted to the processor boundaries, since CLJP is applied to the whole grid. Consequently, a more even distribution of C-points is possible. The Falgout coarsening generates here 58 C-points, clearly more than the classical algorithm, but significantly less than the 64 C-points generated by the RS3 coarsening.

4.4 BC-RS Coarsening

Another potential coarsening scheme that combines RS and CLJP coarsenings is to first treat the processor boundaries using the CLJP coarsening and then filling out the interior of the domains using the RS coarsening. We call this strategy the *BC-RS coarsening.*

Figure 4f. shows the gray C-points on the boundaries that were generated first using the CLJP coarsening scheme. Note the irregular coarsening on the boundaries. Even though for this particular example the total number of C-points achieved is only 52, the irregularities on the boundaries can lead to irregular coarsenings with groups of densely packed C-points in the interior of the processor domains. This effect is to some degree already apparent when investigating the upper right corner of Figure 4f., but becomes much worse for larger test problems. Experiments with this scheme show (see also Section 6) that this approach leads to worse convergence and complexities than the Falgout or the RS3 coarsenings.

5 AMG Interpolation

In this section, we consider the construction of the interpolation operator. The interpolation of the error at the F-point i takes the form

$$e_i = \sum_{j \in C_i} w_{i,j} e_j \tag{4}$$

where $w_{i,j}$ is an interpolation weight determining the contribution of the value e_j in the final value e_i, and C_i is the subset of C-points whose values will be used to interpolate a value at i.

In classical AMG the underlying assumption is that (algebraically) smooth error corresponds to having very small residuals; that is, the error is smooth when the residual $r \equiv f - Au \approx 0$. Since the error, e, and the residual are related by $Ae = r$, smooth error has the property $Ae \approx 0$. Let i be an F-point, to which we wish to interpolate. From this the *ith* equation becomes

$$\sum_{j=1}^{N} a_{i,j} e_j = 0. \tag{5}$$

The points, to which i is connected, comprise three sets: the set C_i, the set D_i^s of points that influence i but are not coarse interpolatory points, and the set D_i^w of points connected to, but not influencing, i. Hence, (5) can be written as

$$\sum_{j \in C_i} a_{i,j} e_j + \sum_{j \in D_i^s} a_{i,j} e_j + \sum_{j \in D_i^w} a_{i,j} e_j. \tag{6}$$

The points $k \in D_i^w$ are weakly connected to i, and the substitution $e_k \approx e_i$ is made. The values e_k for the points in D_i^w are "distributed" to points in C_i that are connected to e_k (hence the **C1** requirement). This yields the formula for the interpolation weights,

$$w_{i,j} = -\frac{1}{a_{i,j} + \sum_{k \in D_i^w} a_{i,k}} \left(a_{i,j} + \sum_{k \in D_i^s} \frac{a_{i,k} a_{k,j}}{\sum_{m \in C_i} a_{k,m}} \right). \tag{7}$$

AMG theory was developed originally [3,15] under the assumption that the operator matrix A is an M-matrix, and that the non-zero off-diagonal entries are all of the opposite sign as the diagonal entries (which all have the same sign). While this assumption is necessary for convergence proof, AMG works quite well on matrices that do not stray too far from this ideal. Our experience, however, indicates that in situations where there are off-diagonal entries of the "wrong" sign (i.e., the same sign as the diagonal) the presence of these coefficients in the denominator in (7) can lead to extremely large interpolation weights and can result in non-convergence or even divergence. Interestingly, while this phenomenon does occur occasionally on sequential problems, we see it much more frequently on large parallel runs. Two reasons account for this. First, the extremely large size of the problems in the parallel run makes it more likely that the catastrophic cancellation will occur (many more weights are calculated). Second, we have observed that Gauss-Seidel relaxation tends to reduce the effect of the large interpolation weights. However, Gauss-Seidel cannot be employed in parallel because communication would be required with every update involving a ghost point. Instead, we employ a hybrid relaxation, Gauss-Seidel on the interior points of the processor domain, followed by a Jacobi sweep along the processor boundaries. If the bad interpolation weights occur at processor boundaries the mitigating effect of Gauss-Seidel is not present, and we see non-convergence or divergence.

We have devised a simple modification to the classical interpolation formula that eliminates this problem. Essentially, we use the coefficient value in the denominator only if it is the "correct" sign; otherwise the coefficient is ignored:

$$w_{i,j} = -\frac{1}{a_{i,j} + \sum\limits_{k \in D_i^w} a_{i,k}} \left(a_{i,j} + \sum_{k \in D_i^s} \frac{a_{i,k} \hat{a}_{k,j}}{\sum\limits_{m \in C_i} \hat{a}_{k,m}} \right), \tag{8}$$

where

$$\hat{a}_{i,j} = \begin{cases} 0 & \text{if } \operatorname{sign}(a_{i,j}) = \operatorname{sign}(a_{i,i}) \\ a_{i,j} & \text{otherwise.} \end{cases}$$

The modified interpolation formula effectively eliminates the situations leading to non-convergence. By itself, however, it leads to a significant increase in the setup time, particularly for 3-dimensional problems. We find that this latter complication can be controlled by judicious selection of the strength threshold α. $\alpha = 0.5$ is often a good choice for 3-dimensional problems.

6 Numerical Experiments

We now present numerical results from a series of experiments designed to test the various coarsening and interpolation schemes described above. The experiments are performed using the Blue Pacific parallel processor at Lawrence Livermore National Laboratory. Varying numbers of processors, as well as a variety of test problems, are presented. The AMG V-cycle uses a pointwise Gauss-Seidel hybrid smoother, in which one relaxation sweep is performed over the C-points, followed by one sweep over the F-points. On points interior to each processor the relaxation is pointwise Gauss-Seidel, using new values as they become available. Updates of points on the processor boundaries are performed after each interior sweep over the C- or F-points, and are Jacobi-like, in that "old" values at the neighboring points are used throughout the sweep over the boundary points. Unless stated differently, a random right hand side is used. In the tables, p denotes the number of processors and N the size of the linear system.

6.1 Laplace Operator

We first apply *BoomerAMG* to the Laplace equation

$$-\Delta u = f, \tag{9}$$

with homogeneous Dirichlet boundary conditions, posed on a regular Cartesian grid on the unit square (or unit cube). The matrix is generated using a finite difference discretization. For the 2-dimensional problem a 9-point stencil is used, while a 7-point stencil is used for the 3-dimensional problem. The purpose of this experiment is to investigate the scalability of the code, as well as its behavior on very large linear systems using many processors.

Tables 1 - 3 display the asymptotic convergence factors, setup times, and solution times for the 2-dimensional problem. A strength threshold $\alpha = 0.25$ is used. Scalability is tested by holding the number of points per processor constant at 122,500 (350×350) while allowing the number of processors to grow. Operator complexities are very uniform for the various coarsenings. The CLJP-coarsening has the highest complexities, with an operator complexity of 2.0. The BC-RS coarsening exhibits an operator complexity of 1.5. All other coarsenings result in an operator complexity of 1.3.

Table 2 reveals that the best scalability of the setup phase is obtained using the RS coarsening with local measures. Since this is the only coarsening requiring no communication between the processors, this result is to be expected. However, RS coarsening also results in the poorest convergence factors (Table 1), indicating that some communication between processors during the coarsening process is essential for algorithmic efficiency. The most effective coarsenings, in terms of convergence factors, are the RS3 and the Falgout methods. Of the two, RS3 shows significantly better setup times. Both the RS3 and Falgout

Table 1
Asymptotic convergence factors for the 9pt 2d Laplacian operator on a Cartesian grid

p	N	CLJP	RS	RSgm	RS3	Falg.	BC-RS
1	122,500	0.31	0.12	0.12	0.12	0.12	0.12
16	1,960,000	0.39	0.92	0.61	0.15	0.17	0.28
64	7,840,000	0.42	0.95	0.62	0.15	0.17	0.30
144	17,640,000	0.43	0.94	0.63	0.15	0.18	0.29
256	31,360,000	0.41	0.93	0.63	0.15	0.18	0.30
400	49,000,000	0.43	0.94	0.63	0.15	0.18	0.31
576	70,560,000	0.45	0.94	0.63	0.17	0.19	0.32
784	96,040,000	0.42	0.94	0.63	0.15	0.18	0.31

Table 2
Setup times (in seconds) for the 9pt 2d Laplacian on a Cartesian grid

p	CLJP	RS	RSgm	RS3	Falg.	BC-RS
1	5.5	3.7	3.7	3.7	4.3	3.8
16	7.6	4.6	5.6	4.8	5.8	5.9
64	8.9	4.9	6.0	5.4	6.9	7.2
144	11.4	5.8	7.0	6.8	9.1	9.8
256	15.0	6.7	8.0	7.7	11.7	12.6
400	19.4	7.9	10.1	9.5	16.2	16.8
576	23.8	9.7	13.8	11.4	19.9	22.7
784	32.2	10.8	15.8	14.4	25.2	28.3

methods show excellent scalability in the solution phase (in fact, all methods except the RS and RSgm show good solution-time scalability).

Relative to the 2-dimensional problem, scalability degrades when solving the 3-dimensional Laplace equation (using a 7-point discretization on a Cartesian grid in the unit cube). This is not surprising, since the processor boundaries now involve planes of data, and significantly more information must be exchanged. The results are displayed in Tables 4 – 7. It can be determined empirically that for this problem, a strength threshold of $\alpha = 0.5$ leads to the best timings, while the choice of a smaller α leads to better convergence rates but significantly higher setup times. The problem size is determined using 64,000 ($40 \times 40 \times 40$) points per processor, and letting the number of processors grow.

Table 3
Solution times in seconds (number of iterations) for the 9pt 2-d Laplacian on a Cartesian grid

p	CLJP	RS	RSgm	RS3	Falg.	BC-RS
1	9.5(9)	5.0(6)	5.0(6)	5.0(6)	5.0(6)	5.0(6)
16	15.0(11)	74.3(69)	20.9(19)	6.5(6)	6.5(6)	9.3(8)
64	15.5(11)	136.2(119)	23.6(21)	7.1(6)	6.7(6)	10.0(8)
144	16.6(11)	110.7(93)	27.0(22)	7.7(6)	6.9(6)	10.6(8)
256	18.0(11)	113.5(80)	29.0(22)	7.5(6)	8.0(6)	11.4(8)
400	18.1(11)	122.2(95)	29.4(22)	8.2(6)	8.5(6)	11.6(8)
576	18.6(11)	133.6(102)	30.4(22)	8.2(6)	8.1(6)	12.0(8)
784	18.8(11)	141.8(95)	33.1(22)	9.2(6)	9.0(6)	13.7(8)

Table 4
Asymptotic convergence factors for the 7pt 3d Laplacian operator on a Cartesian grid

p	N	CLJP	RS	RSgm	RS3	Falg.
1	64,000	0.32	0.10	0.10	0.10	0.10
8	512,000	0.39	0.11	0.31	0.10	0.15
27	1,728,000	0.45	0.20	0.34	0.13	0.21
64	4,096,000	0.45	0.58	0.40	0.17	0.21
125	8,000,000	0.47	0.74	0.41	0.18	0.25
216	13,824,000	0.44	0.70	0.44	0.21	0.25
343	21,952,000	0.49	0.87	0.44	0.19	0.28
512	32,768,000	0.47	0.90	0.45	0.21	0.21
729	46,656,000	0.47	0.85	0.46	0.14	0.26
1000	64,000,000	0.45	0.90	0.48	0.27	0.28

As observed for the 2-dimensional experiments, the RS3 and Falgout coarsenings exhibit the best convergence factors. Interestingly, despite a somewhat larger operator complexity, the Falgout coarsening yields smaller setup times than RS3 coarsening. It can be seen that CLJP is not a suitable coarsening for this problem; it generates far too many coarse points, resulting in extremely high operator complexities.

Table 5
Operator complexities for the 7pt 3d Laplacian operator on a Cartesian grid

p	CLJP	RS	RSgm	RS3	Falg.
1	14.35	3.62	3.62	3.62	3.62
8	16.05	3.73	5.41	3.97	4.45
27	16.70	3.73	5.62	4.33	4.84
64	16.95	3.72	5.54	4.53	5.07
125	17.14	3.71	5.95	4.63	5.20
216	17.27	3.71	5.70	4.73	5.32
343	17.36	3.70	5.52	4.80	5.38
512	17.43	3.70	5.37	4.87	5.43
729	17.48	3.70	5.25	4.92	5.48
1000	17.53	3.70	5.14	4.85	5.52

Table 6
Setup times in seconds for the 7pt 3d Laplacian operator on a Cartesian grid

p	CLJP	RS	RSgm	RS3	Falg.
1	9.9	4.5	4.5	4.5	4.9
8	22.8	8.3	8.7	9.7	11.7
27	35.7	11.9	16.6	21.5	21.6
64	39.1	12.7	24.1	28.2	26.9
125	47.4	13.5	36.9	37.3	31.0
216	61.8	15.8	44.7	45.0	37.4
343	64.8	18.0	49.1	56.9	44.7
512	76.3	20.5	58.9	72.9	53.2
729	90.8	23.2	68.4	86.9	62.4
1000	105.8	28.4	79.9	97.5	78.2

6.2 Anisotropic Problems

We consider next the anisotropic problem

$$-cu_{xx} - u_{yy} - u_{zz} = f \tag{10}$$

with $c = 0.001$. Numerical results are presented in Table 8.

Table 7
Solution times in seconds (number of iterations) for the 7pt 3d Laplacian operator on a Cartesian grid

p	CLJP	RS	RSgm	RS3	Falg.
1	15.6(8)	2.8(4)	2.8(4)	2.8(4)	2.8(4)
8	25.3(9)	4.9(5)	9.6(8)	4.1(4)	5.5(5)
27	29.9(9)	5.4(5)	12.4(8)	5.3(4)	6.8(5)
64	35.4(10)	11.2(10)	17.3(10)	7.5(5)	7.8(5)
125	35.7(10)	14.0(12)	18.3(9)	6.2(4)	8.0(5)
216	37.9(10)	14.7(12)	23.2(10)	9.2(5)	11.4(6)
343	36.8(9)	36.9(23)	22.6(9)	8.9(4)	9.2(5)
512	48.9(11)	37.0(26)	30.6(10)	12.2(5)	9.8(5)
729	38.0(9)	26.0(17)	26.3(9)	10.8(4)	11.6(5)
1000	49.8(11)	27.7(20)	31.3(11)	14.7(5)	15.1(6)

Table 8
Test results for the 3d anisotropic problem.

	Op. compl.		Conv. factors		Setup times		Solve times	
p	Falg.	RS3	Falg.	RS3	Falg.	RS3	Falg.	RS3
1	3.55	3.55	0.04	0.04	3.5	3.3	2.5(4)	2.5(4)
8	3.75	3.94	0.09	0.09	6.9	6.8	4.6(5)	4.8(5)
27	3.82	4.16	0.10	0.10	10.3	11.3	5.1(5)	5.8(5)
64	3.86	4.32	0.13	0.13	12.2	15.1	5.4(5)	6.5(5)
125	3.88	4.45	0.12	0.12	14.7	21.1	5.7(5)	7.0(5)
216	3.90	4.52	0.18	0.19	18.3	26.7	7.3(6)	10.2(6)
343	3.91	4.56	0.11	0.12	25.0	35.0	7.1(5)	9.1(5)
512	3.93	4.56	0.21	0.23	31.3	44.6	8.5(6)	14.5(7)

Only the results for the Falgout and the RS3 coarsenings are displayed; all other coarsenings exhibited poorer performance. The strength threshold $\alpha = 0.25$ is used, and the problem size is 64,000 ($40 \times 40 \times 40$ Cartesian grid) points per processor.

As the number of processors increases (and hence, overall problem size) we observe increasing convergence factors and slightly increasing operator complexities. Consequently, we cannot expect scalable timings. The best overall results are obtained with the Falgout coarsening.

Table 9
Total times in seconds (no. of iterations) for anisotropic problem, with $\gamma = 45$.

p	N	Falgout	RS3	Falg.-CG	RS3-CG
1	65,536	5.9 (7)	5.7 (7)	6.1 (5)	6.0 (5)
4	262,144	10.0 (9)	10.6 (10)	9.8 (6)	10.6 (7)
16	1,048,576	13.2 (11)	16.6 (15)	13.2 (7)	13.1 (8)
64	4,194,304	17.6 (13)	39.8 (36)	15.8 (8)	19.9 (12)
144	9,437,184	22.8 (15)	44.0 (35)	20.5 (9)	22.7 (11)
256	16,265,216	27.0 (16)	45.1 (34)	23.1 (9)	26.9 (12)
400	26,214,400	33.4 (17)	51.7 (33)	28.2 (9)	29.8 (12)

We also consider the specialized 2-dimensional anisotropic problem

$$-(c^2 + \epsilon s^2)u_{xx} + 2(1-\epsilon)scu_{xy} - (s^2 + \epsilon c^2)u_{yy} = 1 \tag{11}$$

with $s = \sin\gamma$, $c = \cos\gamma$, and $\epsilon = 0.001$. This problem is strongly anisotropic, with the direction of dependence given by the angle γ. Hence, in contrast to the previous example, the anisotropy is not grid aligned. Sequential AMG produces poor convergence rates for $\gamma = 30$ or $\gamma = 60$, while rather good convergence is obtained for $\gamma = 45$ [15]. Geometric multigrid fails for this problem [17]. We use a problem size of 65,536 (256×256) points per processor.

For $\gamma = 45$ we obtain a slight increase in operator complexities (from 2.2 to 2.4) for Falgout coarsening, and a complexity of 2.5 for the RS3 coarsening, as the number of processors grows. Falgout coarsening produces convergence factors that increase gradually from 0.1 to 0.37 as the number of processors grows to 512. For RS3 coarsening, we observe a fast increase from 0.1 to 0.65 as the number of processors grows to 64, while further increases in the processor count up to 512 result in a convergence factor that is almost constant at 0.65. Using *BoomerAMG* as a preconditioner for a conjugate gradient solver lead to substantially improved convergence factors: 0.03 for one processor, increasing steadily to 0.1 for 400 processors (with Falgout coarsening giving somewhat better factors than RS3).

For the case $\gamma = 60$, we obtain constant (as processor count grows) values of 3.3 (operator complexities) and 0.7 (convergence factors) for both coarsenings.

The total times (including both the setup and solution phases) are given in Tables 9 and 10.

Table 10
Total times in seconds (no. of iterations) for anisotropic problem with $\gamma = 60$.

p	Falgout	RS3	Falg.-CG	RS3-CG
1	18.4 (24)	18.2 (24)	14.2 (13)	14.0 (13)
4	27.9 (29)	31.2 (29)	19.3 (14)	19.0 (14)
16	32.7 (32)	32.9 (33)	21.7 (15)	21.3 (15)
64	38.5 (36)	38.0 (36)	25.1 (16)	24.6 (16)
144	44.8 (38)	44.4 (37)	30.0 (17)	29.8 (17)
256	48.4 (38)	51.8 (39)	34.9 (17)	31.7 (17)
400	57.5 (39)	60.3 (40)	38.6 (17)	36.1 (17)

6.3 Nonsymmetric Problems

Having previously considered only symmetric problems, we turn now to nonsymmetric problems, and demonstrate the use of the *BoomerAMG* algorithm on these problems. We examine the following nonsymmetric problem

$$-\Delta u + c(u_x + u_y + u_z) = f, \tag{12}$$

posed on a regular Cartesian grid in three dimensions.

An immediate question arises: how can we appropriately scale up the problem for large numbers of processors? If we solve the equation with a constant c, increasing the problem size (by growing the number of processors) leads to very different matrices for different size problems. Therefore, we choose $c = 10q$ where q^3 is the number of processors. This leads to comparable coefficients in the matrices generated by increasing the number of processors (hence, the problem size). The basic problem size is 64,000 ($40 \times 40 \times 40$) grid points per processor, and the strength threshold $\alpha = 0.75$ is used. We also include comparisons to two of the most popular iterative solvers, GMRES and BiCGSTAB.

The results, shown in Table 11, reveal that *BoomerAMG* with Falgout coarsening leads to the best timings with a fairly scalable iteration count. GMRES and BiCGSTAB entail increasing numbers of iterations to achieve convergence as the global problem size increases. Although these results are not strictly scalable, *BoomerAMG*'s timings increase much more slowly than do those of the Krylov solvers. For example, the Falgout method is about twice as fast as BiCGSTAB when using 1 processor, but five times as fast while using 512 processors.

Table 11
Total times in seconds (no. of iterations) for nonsymmetric problem

p	Falgout	RS3	GMRES(10)	BiCGSTAB
1	8.6 (5)	8.2 (5)	24.8 (143)	17.1 (77)
8	18.8 (7)	20.6 (7)	60.2 (273)	40.9 (144)
27	23.1 (7)	28.4 (8)	92.6 (398)	66.7 (226)
64	28.4 (9)	35.2 (10)	135.0 (534)	100.4 (323)
125	34.1 (8)	38.8 (10)	168.6 (642)	117.7 (367)
216	37.9 (9)	47.0 (12)	210.5 (793)	188.3 (493)
343	46.1 (8)	51.4 (11)	263.2 (842)	187.0 (500)
512	56.7 (9)	98.7 (14)	358.7 (1046)	317.6 (644)

6.4 Unstructured Problems

Finally, we include several results on 3-dimensional unstructured finite-element grids on a cube.

We examine two problems: 1) the 3-dimensional Laplace equation, and 2) equation (12) employing the constant convection factor $c = 10$. The problems are generated so that there are roughly 20,000 grid points per processor.

The results for the Laplace equation are given in Tables 12, 13, and 14, and the results for the convection problem are contained in Tables 15 and 16. For most of the experiments, we choose the strength threshold $\alpha = 0.5$, since this gives good convergence and produces fairly scalable iteration counts when *BoomerAMG* is used as preconditioner. However, the complexities are fairly high. If memory usage is an issue, it is possible to significantly decrease operator complexities and consequently memory usage through increasing α. Table 14 shows the times and iteration counts for the Laplace equation when $\alpha = 0.8$. The resulting operator complexities are between 2.3 and 2.4 for the CLJP and the RS3 coarsening, and between 2.4 and 2.5 for the Falgout coarsening. Clearly, both convergence and scalability degrade. This effect is extreme when *BoomerAMG* is used as a standalone solver or as a preconditioner with the RS3 coarsening, however when used as a preconditioner with the CLJP coarsening, times are comparable to those obtained with $\alpha = 0.5$.

Examining all the results, it may be seen that the most effective algorithm uses CLJP coarsening, followed by the Falgout and then the RS3 coarsenings. It is interesting to observe that while CLJP coarsening does not yield an effective solver for structured grids, it outperforms methods employing the other coarsening schemes on unstructured grids.

Table 12
Convergence factors and operator complexities for the Laplace problem on an unstructured grid, $\alpha = 0.5$

		Convergence factors			Operator complexities		
p	N	CLJP	Falgout	RS3	CLJP	Falgout	RS3
1	25,044	0.31	0.29	0.30	3.53	4.06	4.02
4	76,723	0.39	0.38	0.40	3.80	4.42	4.61
12	234,336	0.45	0.45	0.45	4.05	4.75	5.10
36	706,542	0.52	0.50	0.50	4.20	4.90	5.45
96	1,827,749	0.58	0.57	0.56	4.63	5.47	6.07
288	5,527,417	0.62	0.61	0.63	4.69	5.56	6.25
768	14,487,576	0.64	0.64	0.66	4.84	5.84	6.61

Table 13
Total times for the Laplace problem on an unstructured grid, $\alpha = 0.5$

p	CLJP	Falgout	RS3	CLJP-G	Falg.-G	RS3-G
1	9.0(13)	9.4(12)	9.7(13)	8.4(8)	8.4(7)	9.0(8)
4	13.3(16)	14.5(15)	15.2(16)	10.9(8)	12.1(8)	12.8(9)
12	20.2(19)	23.1(19)	25.0(19)	15.8(9)	18.0(9)	18.9(9)
36	28.7(23)	31.7(22)	34.9(22)	20.3(10)	23.1(10)	25.2(10)
96	40.1(28)	45.7(27)	53.0(27)	30.6(11)	34.4(11)	35.4(11)
288	59.8(32)	66.2(31)	84.7(33)	46.5(12)	48.7(12)	51.3(12)
768	96.2(35)	104.4(35)	129.0(36)	65.8(13)	75.6(13)	86.6(13)

Convergence factors and operator complexities are increasing for increasing problem sizes (numbers of processors). Convergence factors of *BoomerAMG* for unstructured problems are poorer than for structured problems (a phenomenon previously noted for sequential AMG [8]); however, the convergence factors can be improved significantly by using *BoomerAMG* as a preconditioner for GMRES.

7 Conclusions

We have presented *BoomerAMG*, an AMG solver for massively parallel computers. The most challenging aspect of parallelizing AMG is the selection of coarse grids, largely because all known methods are essentially multipass methods requiring updates around individual points at the time C-points

Table 14
Total times in seconds for the Laplace problem on an unstructured grid, $\alpha = 0.8$

p	CLJP	Falgout	RS3	CLJP-G	Falg.-G	RS3-G
4	18.2(38)	18.3(37)	22.4(48)	10.1(13)	10.4(13)	10.4(14)
12	30.0(51)	30.6(51)	42.3(74)	13.9(15)	14.8(15)	16.3(18)
36	50.0(70)	46.8(67)	86.7(133)	19.5(19)	21.8(18)	25.1(26)
96	77.1(92)	75.3(89)	162.9(197)	28.0(22)	28.3(22)	39.2(31)
288	158.8(120)	197.6(158)	($>$ 200)	50.8(26)	56.7(29)	76.7(43)

Table 15
Convergence factors and operator complexities for the convection problem on an unstructured grid, $\alpha = 0.5$

		Convergence factors			Operator complexities		
p	N	CLJP	Falgout	RS3	CLJP	Falgout	RS3
1	25,044	0.17	0.18	0.18	3.59	4.08	4.08
4	76,723	0.22	0.22	0.22	3.90	4.47	4.71
12	234,336	0.27	0.27	0.28	4.10	4.77	5.12
36	706,542	0.33	0.34	0.34	4.24	4.92	5.52
96	1,827,749	0.42	0.43	0.43	4.64	5.45	6.03
288	5,527,417	0.48	0.49	0.48	4.70	5.54	6.22
768	14,487,576	0.51	0.54	0.55	4.82	5.74	6.50

Table 16
Total times for the convection problem, $\alpha = 0.5$

p	CLJP	Falgout	RS3	CLJP-G	Falg.-G	RS3-G
1	7.5(10)	8.3(10)	8.3(10)	8.4(8)	9.1(8)	9.0(8)
4	11.4(12)	12.8(12)	12.9(12)	11.9(9)	13.1(9)	13.2(9)
12	16.9(14)	19.8(14)	20.1(14)	16.6(10)	18.8(10)	19.3(10)
36	24.2(17)	27.2(17)	29.7(17)	21.8(11)	24.6(11)	26.8(11)
96	33.2(20)	38.5(20)	44.3(20)	30.1(13)	34.8(13)	39.4(13)
288	49.7(24)	61.0(24)	65.6(24)	46.9(14)	51.1(15)	60.7(15)
768	83.1(26)	94.6(27)	110.4(28)	70.3(15)	86.3(16)	91.2(16)

are selected. The updates necessitate communication, which slows the algorithm considerably. Attempts to eliminate the communication tend to produce poorly selected grids, especially near processor boundaries. We have introduced several parallel coarsening approaches: the RS, RSgm, and RS3 coarsenings, all based on modifications to the classical Ruge-Stüben method; the CLJP method, based on parallel independent-set algorithms; and the Falgout and BC-RS methods, based on hybrid combinations of the RS and CLJP approaches. The algorithms are tested on a variety of test problems. The tests indicate that on structured grids the best coarsenings are the Falgout and the RS3 coarsening. On those problems, the Falgout coarsening often results in better complexities. Both the parallel RS and RSgm coarsening show poor convergence when they violate **C1**. The CLJP algorithm often produces too many C-points on structured grids. The BC-RS coarsenings is affected by the behavior of CLJP and thus produces higher complexities than other RS-based coarsenings. On unstructured grids, the CLJP coarsening appears to give slightly better results than the Falgout and RS3 coarsenings.

Our numerical experiments demonstrated very good scalability for the solution phase of the algorithm for 2-dimensional problems. Scalability is poorer, but still fairly good for 3-dimensional problems, in some cases it could be improved by using *BoomerAMG* as a preconditioner for GMRES. The setup phase does not scale as well due to an increase in operations and necessary communication when dealing with the boundaries, nevertheless overall times achieved were very good compared to Krylov solvers.

Future research will focus on improving the complexities of the coarsenings as well as finding new ways of coarsening.

References

[1] A. Brandt, Algebraic multigrid theory: The symmetric case, in *Preliminary Proceedings for the International Multigrid Conference*, Copper Mountain, Colorado, April 1983.

[2] A. Brandt, Algebraic multigrid theory: The symmetric case, *Appl. Math. Comput.* 19 (1986), 23–56.

[3] A. Brandt, S. F. McCormick, and J. W. Ruge, Algebraic multigrid (AMG) for automatic multigrid solutions with application to geodetic computations. Report, Inst. for Computational Studies, Fort Collins, Colo., October 1982.

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

[5] M. Brezina, A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge, Algebraic multigrid based on element interpolation (AMGe), *SIAM Journal on Scientific Computing* 22 (2000), 1570–1592.

[6] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial* (SIAM, Philadelphia, PA, second ed., 2000).

[7] A. J. Cleary, R. D. Falgout, V. E. Henson, and J. E. Jones, Coarse grid selection for parallel algebraic multigrid, in *Proceedings of the fifth international symposium on solving irregularly structured problems in parallel* (Springer-Verlag, New York, 1998).

[8] A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, G. N. Miranda, and J. W. Ruge, Robustness and scalability of algebraic multigrid, *SIAM Journal on Scientific Computing*, 21 (2000), 1886–1908.

[9] G. Golubovici and C. Popa, Interpolation and related coarsening techniques for the algebraic multigrid method, in *Multigrid Methods IV, Proceedings of the Fourth European Multigrid Conference vol. 116 of ISNM* (Birkhäuser, Basel, 1994) 201–213.

[10] J. E. Jones and S. F. McCormick, Parallel multigrid methods, in: D. E. Keyes, A. H. Sameh, and V. Venkatakrishnan, eds., *Parallel Numerical Algorithms*, (Kluwer Academic Publications, Dordrecht, 1997).

[11] M. T. Jones and P. E. Plassman, A parallel graph coloring heuristic, *SIAM Journal on Scientific Computing* 14 (1993) 654–669.

[12] A. Krechel and K. Stüben, Parallel algebraic multigrid based on subdomain blocking, *GMD Report 71*, GMD, Sankt Augustin, Germany, submitted to Parallel Computing.

[13] M. Luby, A simple parallel algorithm for the maximal independent set problem, *SIAM Journal on Computing* 15 (1986) 1036–1053.

[14] J. W. Ruge and K. Stüben, Efficient solution of finite difference and finite element equations by algebraic multigrid (AMG), in: D. J. Paddon and H. Holstein, eds., *Multigrid Methods for Integral and Differential Equations, The Institute of Mathematics and its Applications Conference Series* (Clarendon Press, Oxford, 1985) 169–212.

[15] J. W. Ruge and K. Stüben, Algebraic multigrid (AMG), in : S. F. McCormick, ed., *Multigrid Methods, vol. 3 of Frontiers in Applied Mathematics* (SIAM, Philadelphia, 1987) 73–130.

[16] K. Stüben, Algebraic multigrid (AMG): experiences and comparisons, *Appl. Math. Comput.* 13 (1983) 419–452.

[17] K. Stüben, Algebraic multigrid (AMG): an introduction with applications, in : U. Trottenberg, C. Osterlee and A. Schüller, eds., *Multigrid* (Academic Press, 2000).

[18] K. Stüben, U. Trottenberg, and K. Witsch, Software development based on multigrid techniques, in: B. Enquist and T. Smedsaas, eds., *Proc. IFIP–Conference on PDE Software, Modules, Interfaces and Systems* (Söderköping, 1983).

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