Coarse-grid Selection for Parallel Algebraic Multigrid

Andrew J. Cleary, Robert D. Falgout, Van Emden Henson, Jim E. Jones

Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA

Abstract. The need to solve linear systems arising from problems posed on extremely large, unstructured grids has sparked great interest in parallelizing algebraic multigrid (AMG). To date, however, no parallel AMG algorithms exist. We introduce a parallel algorithm for the selection of coarse-grid points, a crucial component of AMG, based on modifications of certain parallel independent set algorithms and the application of heuristics designed to insure the quality of the coarse grids. A prototype serial version of the algorithm is implemented, and tests are conducted to determine its effect on multigrid convergence, and AMG complexity.

1 Introduction

Since the introduction of algebraic multigrid (AMG) in the 1980's [4, 2, 3, 5, 19, 16, 18, 17] the method has attracted the attention of scientists needing to solve large problems posed on unstructured grids. Recently, there has been a major surge of interest in the field, due in large part to the need to solve increasingly larger systems, with hundreds of millions or billions of unknowns. Most of the current research, however, focuses either on improving the standard AMG algorithm [9, 7], or on dramatic new algebraic approaches [20, 6]. Little research has been done on parallelizing AMG. The sizes of the modern problems, however, dictate that large-scale parallel processing be employed.

Methods for parallelizing geometric multigrid methods have been known for some time [10], and most of the AMG algorithm can be parallelized using existing technology. Indeed, much of the parallelization can be accomplished using tools readily available in packages such as PETSc or ISIS++. But, the heart of the AMG setup phase includes the coarse-grid selection process, which is inherently sequential in nature.

In this note we introduce a parallel algorithm for selecting the coarse-grid points. The algorithm is based on modifications of parallel *independent set* algorithms. Also, we employ heuristics designed to insure the quality of the coarse grids. A prototype serial code is implemented, and we examine the effect the algorithm has on the multigrid convergence properties.

In Section 2 we outline the basic principles of AMG. Section 3 describes our parallelization model and the underlying philosophy, while the details of the parallel algorithm are given in Section 4. Results of numerical experiments with the serial prototype are presented and analyzed in Section 5. In Section 6 we make concluding remarks and indicate directions for future research.

2 Algebraic Multigrid

We begin by outlining the basic principles of AMG. Detailed explanations may be found in [17]. Consider a problem of the form $A\mathbf{u} = \mathbf{f}$, where A is an $n \times n$ matrix with entries a_{ij}. For AMG, a "grid" is simply a set of indices of the variables, so the original grid is denoted by $\Omega = \{1, 2, \ldots, n\}$. In any multigrid method, the central idea is that error $\mathbf{e}$ not eliminated by relaxation is eliminated by solving the residual equation $A\mathbf{e} = \mathbf{r}$ on a coarser grid, then interpolating $\mathbf{e}$ and using it to correct the fine-grid approximation. The coarse-grid problem itself is solved by a recursive application of this method. Proceeding through all levels, this is known as a multigrid cycle. One purpose of AMG is to free the solver from dependence on geometry (which may not be easily accessible, if it is known at all). Hence, AMG fixes a relaxation method, and its main task is to determine a coarsening process that approximates error the relaxation cannot reduce.

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: "grids" $\Omega^1 \supset \Omega^2 \supset \ldots \supset \Omega^M$; grid operators $A^1, A^2, \ldots, A^M$; interpolation operators $I_{k+1}^k, k = 1, 2, \ldots M-1$; restriction operators $I_k^{k+1}, k = 1, 2, \ldots M-1$; and a relaxation scheme for each level. Once these components are defined, the recursively defined multigrid cycle is as follows:

Algorithm: $MV^k(\mathbf{u}^k, \mathbf{f}^k)$. The (μ_1, μ_2) V-cycle.
If $k = M$, set $\mathbf{u}^M = (A^M)^{-1}\mathbf{f}^M$.
Otherwise:
Relax μ_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 ν_2 times on $A^k\mathbf{u}^k = \mathbf{f}^k$.

For this to work efficiently, two principles must be followed:

P1: *Errors not efficiently reduced by relaxation must be well-approximated by the range of interpolation.*

P2: *The coarse-grid problem must provide a good approximation to fine-grid error in the range of interpolation.*

AMG satisfies **P1** by automatically selecting the coarse grid and defining interpolation, based solely on the algebraic equations of the system. **P2** is satisfied by defining restriction and the coarse-grid operator by the *Galerkin* formulation [14]:

$$I_k^{k+1} = \left(I_{k+1}^k\right)^T \qquad \text{and} \qquad A^{k+1} = I_k^{k+1} A^k I_{k+1}^k. \tag{1}$$

Selecting the AMG components is done in a separate preprocessing step:

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.

2.1 Selecting Coarse Grids and Defining Interpolation

Step 2 is the core of the AMG setup process. 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. Interpolation is then of the form:

$$\left(I_{k+1}^k \mathbf{u}^{k+1}\right)_i = \begin{cases} \mathbf{u}_i^{k+1} & \text{if } i \in C, \\ \sum_{j \in C_i} \omega_{ij} \mathbf{u}_j^{k+1} & \text{if } i \in F. \end{cases} \tag{2}$$

We do not detail the construction of the interpolation weights ω_{ij}, instead referring the reader to [17] for details.

An underlying assumption in AMG is that smooth error is characterized by small residuals, that is, $A\mathbf{e} \approx \mathbf{0}$, which is the basis for choosing coarse grids and defining interpolation weights. For simplicity of discussion here, assume that A is a symmetric positive-definite M-matrix, with $a_{ii} > 0, a_{ij} \leq 0$ for $j \neq i$, and $\sum a_{ij} \geq 0$.

We say that point i *depends* on point j if a_{ij} is "large" in some sense, and hence, to satisfy the ith equation, the value of u_i is affected more by the value of u_j than by other variables. Specifically, the *set of dependencies* of i is defined by

$$S_i \equiv \left\{ j \neq i : -a_{ij} \geq \alpha \max_{k \neq i}(-a_{ik}) \right\}, \tag{3}$$

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 depend on point i, and we say that S_i^T is the set of *influences* of point i.

A basic premise of AMG is that relaxation smoothes the error in the direction of influence. Hence, we may select $C_i = S_i \cap C$ as the set of interpolation points for i, and adhere to the following criterion while choosing C and F:

P3: *For each $i \in F$, each $j \in S_i$ is either in C or $S_j \cap C_i \neq \emptyset$.*

That is, if i is a fine point, then the points influencing i must either be coarse points or must themselves depend on the coarse points used to interpolate u_i.

The coarse grid is chosen to satisfy two criteria. We enforce **P3** in order to insure good interpolation. However, we wish to keep the size of the coarse-grid as small as possible, so we desire that

P4: *C is a maximal set with the property that no C-point influences another C-point.*

It is not always possible to enforce both criteria. Hence, we enforce **P3** while using **P4** as a guide in coarse-point selection.

AMG employs a two-pass process, in which the grid is first "colored", providing a tentative C/F choice. Essentially, a point with the largest number of influences ("influence count") is colored as a C point. The points depending on this C point are colored as F points. Other points influencing these F points are more likely to be useful as C points, so their influence count is increased. The process is repeated until all points are either C or F points. Next, a second pass is made, in which some F points may be recolored as C points to ensure that **P3** is satisfied. Details of the coarse-grid selection algorithm may be found in [17], while a recent study of the efficiency and robustness of the algorithm is detailed in [7].

Like many linear solvers, AMG is divided into two main phases, the *setup* phase and the *solve* phase. Within each of these phases are certain tasks that must be parallelized to create a parallel AMG algorithm. They are:

- **Setup phase:**
  - Selecting the coarse grid points, Ω^{k+1}.
  - Construction of interpolation and restriction operators, I_{k+1}^k, I_k^{k+1}.
  - Constructing the coarse-grid operator $A^{k+1} = I_k^{k+1} A^k I_k^{k+1}$.
- **Solve phase:**
  - Relaxation on $A^k \mathbf{u}^k = \mathbf{f}^k$.
  - Calculating the residual $\mathbf{r}^k \leftarrow \mathbf{f}^k - A^k \mathbf{u}^k$.
  - Computing the restriction $\mathbf{f}^{k+1} = I_k^{k+1} \mathbf{r}^k$.
  - Interpolating and correcting $u^k \leftarrow \mathbf{u}^k + I_{k+1}^k \mathbf{u}^{k+1}$.

3 Parallelization Model

In this work we target massively parallel distributed memory architectures, though it is expected that the method will prove useful in other settings, as well. Currently, most of the target platforms support shared memory within clusters of processors (typically of size 4 or 8), although for portability we do not utilize this feature. We assume explicit message passing is used among the processors, and implement this with MPI [15]. The equations and data are distributed to the processors using a *domain-partitioning* model. This is natural for many problems of physics and engineering, where the physical domain is partitioned by subdomains. The actual assignment to the processors may be done by the application code calling the solver, by the gridding program, or by a subsequent call to a graph partitioning package such as Metis [12]. The domain-partitioning strategy should not be confused with *domain decomposition*, which refers to a family of solution methods.

We use object-oriented software design for parallel AMG. One benefit of this design is that we can effectively employ kernels from other packages, such as

PETSc [1] in several places throughout our code. Internally, we focus on a *matrix object* that generalizes the features of "matrices" in widely-used packages. We can write AMG-specific routines once, for a variety of matrix data structures, while avoiding the necessity of reinventing widely available routines, such as matrix-vector multiplication.

Most of the required operations in the solve phase of AMG are standard, as are several of the core operations in the setup phase. We list below the standard operations needed by AMG:

- *Matrix-vector multiplication*: used for residual calculation, for interpolation, and restriction (both use rectangular matrices; restriction multiplies by the transpose). Some packages provide all of the above, while others may have to be augmented, although the coding is straightforward in these cases.
- *Basic iterative methods*: used for the smoothing step. Jacobi or scaled Jacobi are most common for parallel applications, but any iterative method provided in the parallel package could be applied.
- *Gathering/scattering processor boundary equations*: used in the construction of the interpolation operators and in the construction of coarse-grid operators via the Galerkin method. Each processor must access "processor-boundary equations" stored on neighboring processors. Because similar functionality is required to implement additive Schwarz methods, parallel packages implementing such methods already provide tools that can be modified to fulfill this requirement.

4 Parallel Selection of Coarse Grids

Designing a parallel algorithm for the selection of the coarse-grid points is the most difficult task in parallelizing AMG. Classical AMG uses a two-pass algorithm to implement the heuristics, **P3** and **P4**, that assure grid quality and control grid size. In both passes, the algorithm is inherently sequential. The first pass can be described as:

1) Find a point j with maximal measure $w(j)$. Select j as a C point.
2) Designate neighbors of j as F points, and update the measures of other nearby points, using heuristics to insure grid quality.
Repeat steps **1)** and **2)** until all points are either C or F points.

This algorithm is clearly unsuitable for parallelization, as updating of measures occurs after each C point is selected. The second pass of the classical AMG algorithm is designed to enforce *P3*, although we omit the details and refer the reader to [17]. We can satisfy **P3** and eliminate the second pass through a simple modification of step **2)**.

Further, we may allow for parallelism by applying the following one-pass algorithm. Begin by performing step **1)** globally, selecting a *set* of C points, D, and then perform step **2)** locally, with each processor working on some portion of the set D. With different criteria for selecting the set D, and armed with various heuristics for updating the neighbors in **2)**, a family of algorithms may be developed. The overall framework is:

Input: the $n \times n$ matrix A^k (level k).
Initialize:
  $F = \emptyset, \; C = \emptyset$.
  $\forall \, i \, \in \, \{1...n\}$,
    $w(i) \leftarrow$initial value.
Loop until $|C| + |F| = n$:
  Select an independent set of points D.
  $\forall \, j \, \in \, D$:
    $C \; = \; C \, \cup \, j$.
      $\forall \, k$ in set local to j, update $w(k)$.
      if $w(k) = 0, \quad F = F \cup k$.
End loop

4.1 Selection of the set D

For the measure $w(i)$, we use $|S_i^T| + \sigma(i)$, 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. The set D is then selected using a modification of a parallel maximal independent set algorithm developed in [13, 11, 8].

A point j will be placed in the set D if $w(j) > w(k)$ for all k that either influence or depend on j. By construction, this set will be independent. While our implementation selects a maximal set of points possessing the requisite property, this is not necessary, and may not be optimal. An important observation is that this step can be done entirely in parallel, provided each processor has access to the w values for points with influences that cross its processor boundary.

4.2 Updating $w(k)$ of neighbors

Describing the heuristics for updating $w(k)$ is best done in terms of graph theory. We begin by defining S, the auxiliary *influence matrix*:

$$S_{ij} = \begin{cases} 1 & \text{if } \; j \in S_i, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

That is, $S_{ij} = 1$ only if i depends on j. The ith row of S gives the dependencies of i while the ith column of S gives 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 $S_{ij} \neq 0$. Notice that the directed edges point in the direction of dependence. To update the $w(k)$ of neighbors, we apply the following pair of heuristics.

P5: Values at C points are not interpolated; hence, neighbors that influence a C point are less valuable as potential C points themselves.

P6: If k and j both depend on c, a given C point, and j influences k, then j is less valuable as a potential C point, since k can be interpolated from c.

The details of how these heuristics are implemented are:

$\forall\, c \,\in\, D,$	
	P5:
$\quad \forall\, j \mid S_{cj} \neq 0,$	(each j that influences c)
$\qquad w(j) \leftarrow w(j) - 1$	(decrement the measure)
$\qquad S_{cj} \leftarrow 0$	(remove edge cj from the graph)
	P6:
$\quad \forall\, j \mid S_{jc} \neq 0$	(each j that depends on c),
$\qquad S_{jc} \leftarrow 0$	(remove edge jc from the graph)
$\qquad \forall\, k \mid S_{kj} \neq 0,$	(each k that j influences),
$\qquad\quad$ if $S_{kc} \neq 0$	(if k depends on c),
$\qquad\qquad w(j) \leftarrow w(j) - 1$	(decrement the measure)
$\qquad\qquad S_{kj} \leftarrow 0$	(remove edge kj from the graph)

The heuristics have the effect of lowering the measure $w(k)$ 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. Frequently the step $w(j) \rightarrow w(j) - 1$ causes $\lfloor w(j) \rfloor = 0$. When this occurs j is flagged as an F point.

Once the heuristics have been applied for all the points in D, a global communication step is required, so that each processor has updated w values for all neighbors of all their points. The entire process is then repeated. C points are added by selecting a new set, D, from the vertices that still have edges attached in the modified graph of S. This process continues until all n points have either been selected as C points or F points.

5 Numerical Experiments

To test its effect on convergence and algorithmic scalability, we include a serial implementation of the parallel coarsening algorithm in a standard sequential AMG solver. Obviously, this does not test parallel efficiency, which must wait for a full parallel implementation of the entire AMG algorithm.

Figure 1 shows the coarse grid selected by the parallel algorithm on a standard test problem, the 9-point Laplacian operator on a regular grid. This test is important because the grid selected by the standard sequential AMG algorithm

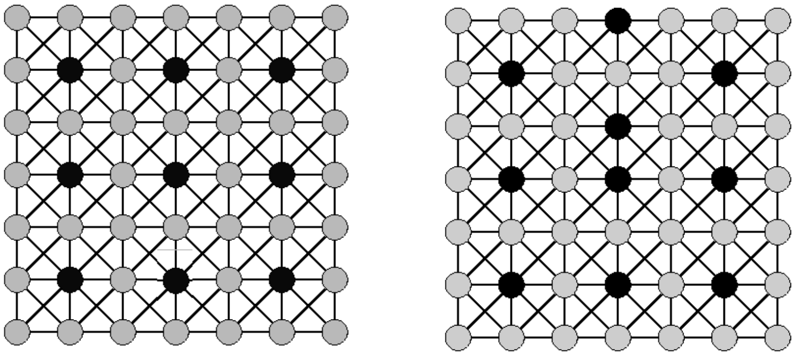

Fig. 1. *Coarse grids for the structured-grid 9-point Laplacian operator. The dark circles are the C points.* Left: *Grid selected by the standard algorithm.* Right: *Grid selected by the parallel algorithm.*

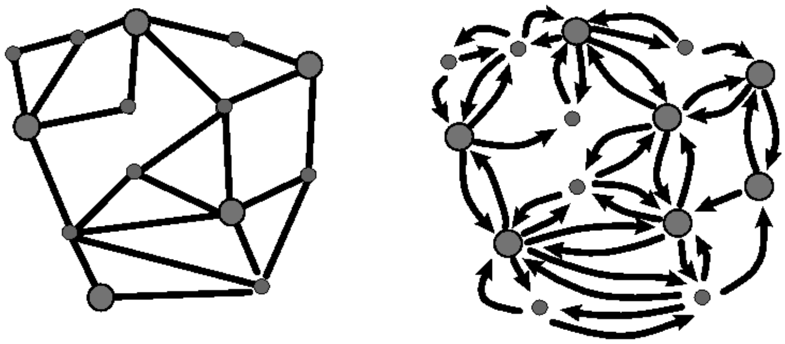

Fig. 2. *Coarse grids for an unstructured grid. The large circles are the C points.* Left: *Grid selected by the standard algorithm.* Right: *Grid selected by the parallel algorithm. Graph connectivity is shown on the left, while the full digraph is shown on the right.*

is also the optimal grid used in geometric multigrid for this problem. Examining many such test problems on regular grids, we find that the parallel coarsening algorithm typically produces coarse grids with 10–20% more C points than the sequential algorithm. On unstructured grids or complicated domains, this increase tends to be 40–50%, as may be seen in the simple example displayed in Figure 2.

The impact of the parallel coarsening algorithm on convergence and scalability is shown in two figures. Figure 3 shows the convergence factor for the 9-point Laplacian operator on regular grids ranging in size from a few hundred to nearly a half million points. Several different choices for the smoother and the parameter α are shown. In Figure 4 the same tests are applied to the 9-point Laplacian operator for anisotropic grids, where the aspect ratios of the under-

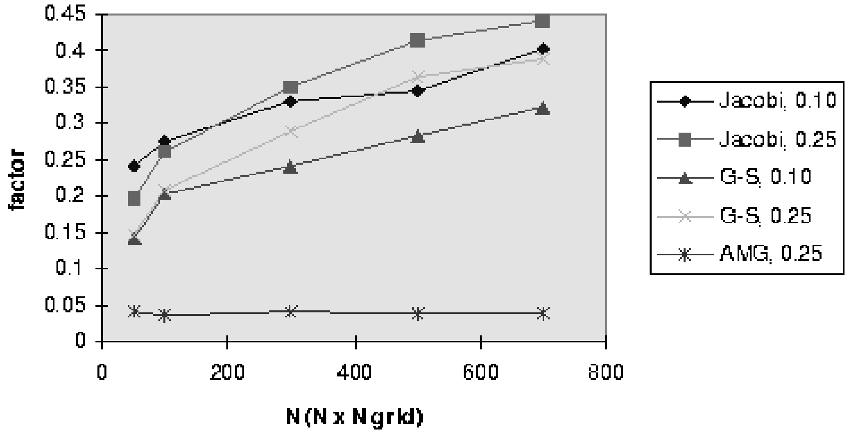

Fig. 3. *Convergence factors for parallel AMG for the 9-point Laplacian*

lying quadrilateral finite elements are extremely high. In both figures, we see that convergence factors for the grids chosen by the parallel algorithm are significantly larger than standard AMG (shown as "AMG" in Figure 3; not shown in Figure 4), although the parallel algorithm still produce solutions in a reasonable number of iterations. Of more concern is that the convergence factors do not scale well with increasing problem size. We believe that this may be caused by choosing too many coarse grid points at once, and that simple algorithmic modifications mentioned below may improve our results.

Figure 5 shows the grid and operator complexities for the parallel algorithm applied to the 9-point Laplacian operator. Grid complexity is the total number of grid points, on all grids, divided by the number of points on the original grid. Operator complexity is the total number of non-zeros in all operators $A^1, A^2, \ldots$ divided by the number of non-zeros in the original matrix. Both the grid and operator complexities generated using by the parallel algorithm are essentially constant with increasing problem size. While slightly larger than the complexities of the sequential grids, they nevertheless appear to be scalable.

The framework described in Section 4 permits easy modification of the algorithm. For example, one may alter the choice of the set D of C points. We believe that the convergence factor degradation shown in our results may be due to selecting too many coarse grid points. One possibility is to choose the minimal number of points in D, that is, one point per processor. This amounts to running the sequential algorithm on each processor, and there a number of different ways to handle the interprocessor boundaries. One possibility is to coarsen the processor boundary equations first, using a parallel MIS algorithm, and then treat each domain independently. Another option is to run the sequential algorithm on each processor ignoring the nodes on the boundary, and then patch up the

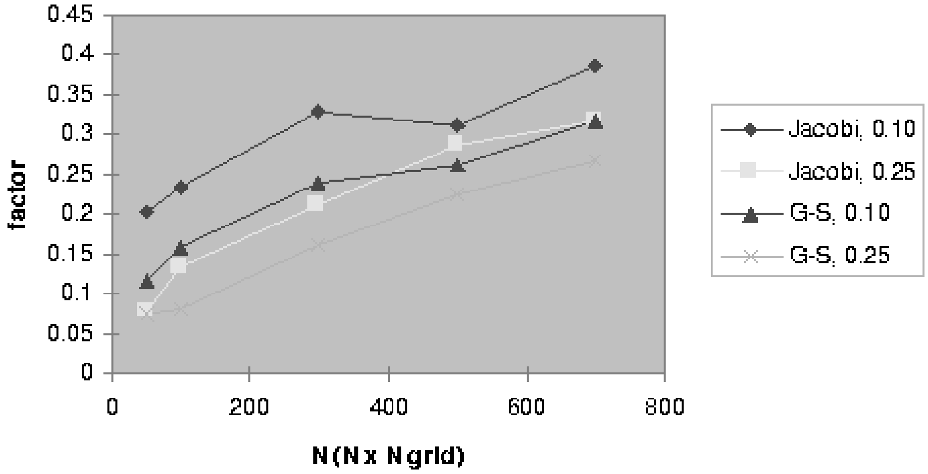

Fig. 4. *Convergence rates for parallel AMG for the anisotropic grid problem.*

grids on the processor boundaries.

6 Conclusions

Modern massively parallel computing requires the use of scalable linear solvers such as multigrid. For unstructured-grid problems, however, scalable solvers have not been developed. Parallel AMG, when developed, promises to be such a solver. AMG is divided into two main phases, the setup phase and the solve phase. The solve phase can be parallelized using standard techniques common to most parallel multigrid codes. However, the setup phase coarsening algorithm is inherently sequential in nature.

We develop a family of algorithms for selecting coarse grids, and prototype one member of that family using a sequential code. Tests with the prototype indicate that the quality of the selected coarse grids are sufficient to maintain constant complexity and to provide convergence even for difficult anisotropic problems. However, convergence rates are higher than for standard AMG, and do not scale well with problem size. We believe that this degradation may be caused by choosing too many coarse grid points at once, and that simple algorithmic modifications may improve our results. Exploration of these algorithm variants is the subject of our current research.

References

1. S. Balay, W. Gropp, L. C. McInnes, and B. Smith, *Petsc 2.0 user's manual*, Tech. Rep. ANL-95/11, Argonne National Laboratory, Nov. 1995.

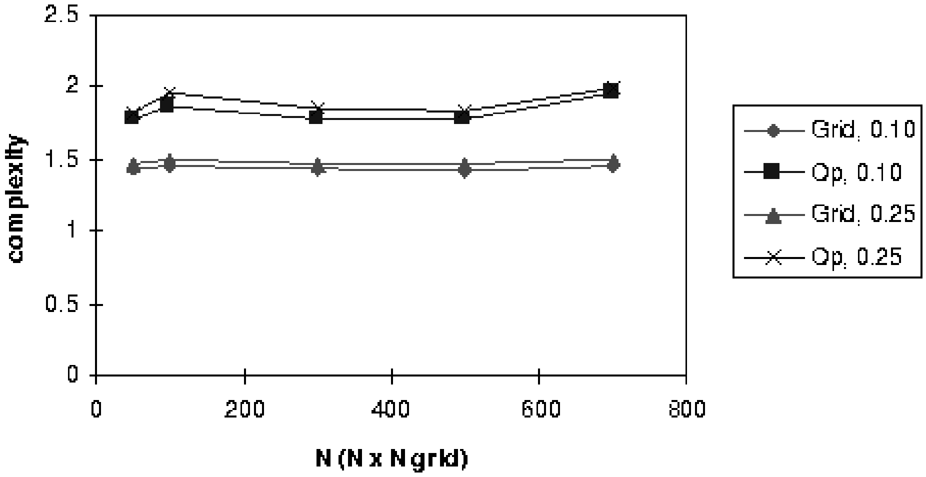

Fig. 5. *Operator complexity for parallel AMG on example problem*

2. A. Brandt, *Algebraic multigrid theory: The symmetric case*, in Preliminary Proceedings for the International Multigrid Conference, Copper Mountain, Colorado, April 1983.
3. A. Brandt, *Algebraic multigrid theory: The symmetric case*, Appl. Math. Comput., 19 (1986), pp. 23–56.
4. 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.
5. ———, *Algebraic multigrid (AMG) for sparse matrix equations*, in Sparsity and Its Applications, D. J. Evans, ed., Cambridge University Press, Cambridge, 1984.
6. 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).* Submitted to the SIAM Journal on Scientific Computing special issue on the Fifth Copper Mountain Conference on Iterative Methods, 1998.
7. 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.* Submitted to the SIAM Journal on Scientific Computing special issue on the Fifth Copper Mountain Conference on Iterative Methods, 1998.
8. R. K. Gjertsen, Jr., M. T. Jones, and P. E. Plassman, *Parallel heuristics for improved, balanced graph colorings*, Journal of Parallel and Distributed Computing, 37 (1996), pp. 171–186.
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, Amsterdam, July 6-9, 1993, vol. 116 of ISNM, Basel, 1994, Birkhäuser, pp. 201–213.

10. J. E. Jones and S. F. McCormick, *Parallel multigrid methods*, in Parallel Numerical Algorithms, D. E. Keys, A. H. Sameh, and V. Venkatakrishnan, eds., Dordrecht, Netherlands, 1997, Kluwer Academic Publications.
11. M. T. Jones and P. E. Plassman, *A parallel graph coloring heuristic*, SIAM Journal on Scientific Computing, 14 (1993), pp. 654–669.
12. G. Karypis and V. Kumar, *A coarse-grain parallel multilevel k-way partitioning algorithm*, in Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing, 1997.
13. M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM Journal on Computing, 15 (1986), pp. 1036–1053.
14. S. F. McCormick, *Multigrid methods for variational problems: general theory for the V–cycle*, SIAM J. Numer. Anal., 22 (1985), pp. 634–643.
15. MPI Forum, *MPI: A message-passing interface standard*, International J. Supercomputing Applications, 8(3/4) (1994), pp. 654–669.
16. J. W. Ruge and K. Stüben, *Efficient solution of finite difference and finite element equations by algebraic multigrid (AMG)*, in Multigrid Methods for Integral and Differential Equations, D. J. Paddon and H. Holstein, eds., The Institute of Mathematics and its Applications Conference Series, Clarendon Press, Oxford, 1985, pp. 169–212.
17. ———, *Algebraic multigrid (AMG)*, in Multigrid Methods, S. F. McCormick, ed., vol. 3 of Frontiers in Applied Mathematics, SIAM, Philadelphia, PA, 1987, pp. 73–130.
18. K. Stüben, *Algebraic multigrid (AMG): experiences and comparisons*, Appl. Math. Comput., 13 (1983), pp. 419–452.
19. K. Stüben, U. Trottenberg, and K. Witsch, *Software development based on multigrid techniques*, in Proc. IFIP–Conference on PDE Software, Modules, Interfaces and Systems, B. Enquist and T. Smedsaas, eds., Sweden, 1983, Söderköping.
20. P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid based on smoothed aggregation for second and fourth order problems*, Computing, 56 (1996), pp. 179–196.