Electronic Transactions on Numerical Analysis.
Volume 23, pp. 105-128, 2006.

ISSN 1068-9613.

ETNA
Kent State University
etna@mcs.kent.edu

SCALABLE ALGEBRAIC MULTIGRID ON 3500 PROCESSORS*

WAYNE JOUBERT† AND JANE CULLUM‡

Abstract. A parallel algebraic multigrid linear solver method is presented which is scalable to thousands of processors on significant classes of two- and three-dimensional problems. The algorithm is entirely algebraic and does not require prior information on the physical problem. Scalability is achieved through the use of an innovative parallel coarsening technique in addition to aggressive coarsening and multipass interpolation techniques. Details of this algorithm are presented together with numerical results on up to several thousand processors.

Key words. algebraic multigrid, AMG, parallel computing, simulations, scalable, linear solvers, parallel coarsening

AMS subject classifications. 65F10

1. Introduction.

The demand for larger physical simulations on large-scale parallel computers drives an increasing need for efficient and scalable linear solver methods. This need has sparked recent interest in algebraic multigrid (AMG) methods. On single processor computers these methods afford scalable solutions to a variety of problems of interest. Scalability in this context indicates that the time to solution to a specified tolerance for a class of linear systems requires bounded or slowly increasing time per unknown as the number of unknowns is increased.

On parallel computers, linear solver methods are said to be scalable if the time to solution is bounded or slowly increasing as the number of unknowns per processor is kept nearly constant and the number of processors is increased. Recent efforts, e.g., [2], [7], [12] and [16], have been made to develop scalable parallel AMG methods, in some cases generalizing the serial AMG algorithm of Ruge and Stüben [14]; for a survey of recent work see [1].

This paper focuses on parallel AMG methods implemented in the parallel algebraic multigrid solver library, LAMG [9], [10]. As will be shown, the LAMG solvers are scalable for important classes of problems derived from 2-D and 3-D physical simulations. The solvers are also fully algebraic, not requiring any physics or grid information from the user, and can solve problems defined on arbitrary unstructured grids.

The solver methods described in this paper are parallel extensions of the classical AMG method of Ruge and Stüben [14] and the aggressive coarsening AMG methods of Stüben [15]. The parallelization of these methods is targeted to scalability on up to thousands of processors.

Other recent work on parallel AMG has been based on similar but different approaches. Henson and Yang [7] describe a parallelization of Ruge/Stüben AMG which makes use of new parallel coarse grid selection algorithms, derived in part from parallel maximal independent set (MIS) methods, see e.g. [8], [13]. The reported performance of these methods on 3-D problems in parallel, however, indicates high computational expense for the AMG setup phase. In contrast, the method described in this paper uses a generalization of the Ruge/Stüben coarse grid selection algorithm which does not rely on the approach of [13]. Also, this method makes use of aggressive coarsening AMG and therefore does not have efficiency or scalability problems for the 3-D parallel case.

* Received August 31, 2003. Accepted for publication January 29, 2006. Recommended by T. Manteuffel. Los Alamos National Laboratory LA-UR 05-4129, unlimited release. This work was supported in part by the Department of Energy through grant W-7405-ENG-36, with Los Alamos National Laboratory.

† Xiylon Software, P.O. Box 74, Gable, SC 29051 (wayne@xiylon.com).

‡ Guest Scientist, Los Alamos National Laboratory, Group CCS-2, Mail Stop D413 Los Alamos, NM 87545 (cullumj@lanl.gov).

ETNA
Kent State University
etna@mcs.kent.edu

Krechel and Stüben [12] present a parallelization of AMG based primarily on modifying the coarsening and the interpolation operators at the processor subgrid boundaries to control interprocessor connectivity as well as to facilitate parallel coarsening. Their work presents parallel performance on up to 64 processors. The present work is different in that it uses the same interpolation scheme as the corresponding serial methods for all problem unknowns. This approach avoids any problems which may occur from using a different interpolation at the processor subgrid boundaries, which may be important for simulations with complex physics solved on very many processors.

Tuminaro and Tong [16] present a parallel method based on the smoothed aggregation method of Vanek et. al. [17]. The numerical results presented indicate effective parallelization. However, the performance of the method is sensitive to the size and shape of the aggregates formed by the algorithm, which may be a particular concern for more challenging problem geometries or parallel decompositions. The current work, on the other hand, does not require the formation of aggregates and thus does not have this concern.

The organization of the remainder of this paper is as follows. Section 2 briefly reviews the basic elements of AMG algorithms. Sections 3 and 4 describe the particular techniques used in the parallel LAMG algorithms. Section 5 discusses some performance considerations relevant to the parallel methods. Then, section 6 gives the results of parallel numerical experiments on large numbers of processors.

2. AMG algorithmic components. It is assumed that the reader is familiar with AMG methods as described for example in [14], [15]. Here we give a very brief overview of some of the fundamental issues involved.

The basic idea of AMG is to improve the convergence of an iterative method on a given problem by solving smaller versions of the same problem and using these solutions to improve the solution estimate on the original problem.

A sparse linear system can be represented as a graph, where the nodes of the graph indicate the system unknowns and the edges of the graph are the nonzero entries of the matrix.

An AMG algorithm is composed of two parts: a setup phase and a solve phase. Given the linear system and its associated graph, the AMG setup phase entails the following steps.

- *Smoothers* are built, which are iterative methods used to solve the linear system for the given matrix, accounting for short-range influences between nodes of the graph. Common examples of smoothers are the underrelaxed Jacobi method and the Gauss-Seidel method.
- Edges of the original matrix graph are identified as either *strong* or *weak connections*. This identification refers to the implied influence between graph nodes indicated by the given matrix. For a matrix $A = \{a_{ij}\}$, a point i is considered to be *strongly connected* to a point j if $-a_{ij} \geq \theta \max_{k \neq i} -a_{ik}$, where θ is a preset parameter, typically chosen to be $\theta = .25$. (Note: [15] presents an additional criterion for identifying strong positive connections; we do not make use of this criterion here.)
- A *coarsening process* is applied to the graph of strong connections in order to select a subset of the original nodes of the problem. These nodes are considered to be strategic for representing the smaller or coarse version of the problem. Two methods are available. First, standard coarsening as described in [14] attempts to account for influences via nearest-neighbor connections in the graph. The algorithm involves a first pass to make the initial selection of coarse points and a second pass to add additional coarse points which may be needed in some cases. Second, aggressive coarsening [15] attempts to account for more long-range influences. The algorithm which implements aggressive coarsening essentially applies the first pass of the standard coarsening algorithm several times in sequence, resulting in a smaller number

ETNA
Kent State University
etna@mcs.kent.edu

of coarse points.

- *Intergrid transfer operators* are formed to transfer vectors between the original problem nodes and the coarse problem nodes. Two methods are considered: Ruge/Stüben interpolation [14], used with standard coarsening, and multipass interpolation [15], used with aggressive coarsening. The resulting interpolation operator can then optionally be modified in several ways. First, Jacobi relaxation of interpolation can be used to improve the quality of the interpolation, though this may also make the operator less sparse. Second, thresholding can be applied to make the operator more sparse by discarding small entries. For details, see [15].
- These operators are used to *project* the original matrix to a representation of the original problem on the coarse set of nodes.
- The above procedure is repeated *recursively* on the resulting matrix, until a maximum number of levels is reached. The algorithm may coarsen down to a single point or alternatively stop coarsening when some specified criterion is attained. The problem on the coarsest grid can then be solved by a direct or an iterative method.

Given an AMG setup, the following steps are performed to execute a single AMG iteration or V-cycle:

- A residual $r^{(n)} = b - Au^{(n)}$ is formed based on the current approximate solution $u^{(n)}$ for the given linear system $Au = b$.
- Several smoothing steps are applied to this residual.
- The result is transferred to the coarse grid.
- An approximate solve is performed on this smaller system, using the AMG solve process applied recursively.
- The result is transferred back to the original problem nodes.
- Several additional smoothing steps are applied.

This V-cycle AMG iteration can be accelerated by a conjugate gradient-type method, for which the V-cycle is considered to be a preconditioning step.

3. AMG algorithms and their parallelization. The LAMG code implements parallel extensions of Ruge/Stüben AMG [14] and Stüben's aggressive coarsening AMG [15]. Unless otherwise noted, the algorithm definitions in LAMG follow the descriptions given in [14] and [15]. For additional details, see [9].

Many underlying components of these two serial algorithms are easily parallelized using standard sparse parallel linear algebra techniques. This includes such operations as sparse matrix-vector products, sparse matrix-matrix products and global inner products. The parallelization of certain other parts of AMG requires particular attention so that parallel scalability is attained. The parallelization of these parts is described below.

3.1. Smoother. The primary smoother used at each level is underrelaxed Jacobi smoothing. The relaxation factor is computed via Gershgorin's circle theorem applied to the given matrix.

Though underrelaxed Jacobi smoothing performs well for the cases studied, in some cases using the Jacobi-preconditioned conjugate gradient method (JCG) as a smoother can give improved performance. Conjugate gradient methods have been used as multigrid smoothers before; see e.g. [4], [6]. Our experience has been that for sufficiently well-behaved problems, e.g. discretizations of Poisson problems, JCG smoothing can improve both solution time and scalability relative to underrelaxed Jacobi smoothing. However, the use of a nonstationary method such as JCG as a smoother makes the AMG V-cycle preconditioning a nonlinear operator, potentially causing convergence problems. To increase robustness in the presence of these nonlinearities, the flexible GMRES acceleration method FGMRES is used as the acceleration for the multigrid V-cycle preconditioner when JCG smoothing is used.

ETNA
Kent State University
etna@mcs.kent.edu

3.2. Coarsening. The coarsening strategy used for parallel computation must not only be parallelizable but also yield coarse grid problems which are suitable for efficient parallel solution.

Ruge/Stüben [14] coarsening generates coarse grid problems which can be solved in parallel efficiently for matrices derived from 2-D physical problems. However, for 3-D problems the coarse grid matrices generated generally have increasing numbers of nonzeros per row as the number of multigrid levels is increased. This results in increased interprocessor communication which destroys scalability, especially for the AMG setup phase; see section 5 below.

To mitigate this effect, aggressive coarsening as defined in [15] is used. The rate of coarsening between levels is tunable; aggressive coarsening methods considered here are A2 coarsening (moderately aggressive), A1 coarsening (more aggressive), and (4,4) coarsening (even more aggressive); see [15].

Our experience has been that for a given aggressiveness setting, more difficult problems tend to generate more coarse grid points to represent the problem on coarse grids. For these cases, the aggressiveness of the coarsening may need to be increased to enforce adequate sparsity of the coarse grid matrices.

The algorithms of [15] utilize aggressive coarsening only for the coarsening step at the first level of the AMG hierarchy. However, our experiments with large 3-D problems indicate the need to apply aggressive coarsening at every level of the AMG hierarchy, to insure that the coarse grid matrices at each level are kept sparse (see Subsection 5.5). Therefore, in the present work aggressive coarsening is used at every level.

Each step of the aggressive coarsening process requires the use of a standard coarsening kernel. The original Ruge/Stüben algorithm [14] defined standard coarsening as a two-pass method for generating coarse grids. This entailed a first pass similar to a greedy maximal independent set (MIS) algorithm and a second pass which added additional coarse grid points. The second pass of this algorithm is sequential but can be efficiently parallelized via a simple reordering of the unknowns. In this case the processor subgrid interiors are visited first, then the processor subgrid boundaries are visited by first coloring the processors in such a way that processors of the same color can work independently and then visiting each color in sequence. However, the second pass, as pointed out by Stüben [15], is not essential when aggressive coarsening and multipass interpolation are used. Therefore we use only the first pass of the coarsening algorithm.

Parallelization of the first pass of standard coarsening presents significant challenges. The parallel coarsening strategy used is described in section 4.

3.3. Interpolation. As in [15], we use multipass interpolation in combination with aggressive coarsening. Multipass interpolation parallelizes in a straightforward way, and the parallelized method gives identical results (up to roundoff) to the serial method. Also following [15], we use (full) Jacobi relaxation of interpolation to refine the resulting operators, followed by thresholding to eliminate very small entries of these operators. These algorithm components also parallelize in a natural way.

3.4. Number of levels. Multiplicative multigrid methods of the type examined in this paper have an inherent $\log(n)$ complexity component pertaining to the communication requirements for each multigrid level as the number of levels is increased by $\log(n)$ for larger problems. In a parallel context, this means there is an order $\log(p)$ component in the execution time. Fortunately, this $\log(p)$ factor in the scaling is typically small and in fact can be made even smaller by limiting the number of multigrid levels used.

For parallel AMG solves, performing work on the coarsest grids can be expensive because of the communication requirements relative to the computations performed. For this

ETNA
Kent State University
etna@mcs.kent.edu

reason, when the linear system matrix is symmetric, the coarsening process is halted when a bound on the condition number of the problem at that level is sufficiently small. This bound is computed using the conservative Gershgorin eigenvalue bounds for the matrix at that level. If a halt is indicated, the corresponding coarse grid is treated as the coarsest grid in the AMG hierarchy and the problem at that level is solved without recourse to more multigrid levels.

3.5. Solves on the coarsest grid. Since the matrix on the coarsest grid necessarily has a bounded condition number, the number of iterations required for an iterative method to solve the corresponding linear system to a given tolerance is bounded. Thus we solve the coarsest system with an iterative method. The two methods used are underrelaxed Jacobi iteration and the Jacobi conjugate gradient method.

The iteration terminates when the ratio of the two-norm of the residual vector to the two-norm of the right-hand side vector is less than or equal to a given tolerance.

It should be noted that using a convergence tolerance rather than specifying a fixed number of iterations for the coarse grid solve introduces a mild nonlinearity into the multigrid V-cycle preconditioner. Likewise, use of the Jacobi conjugate gradient method rather than underrelaxed Jacobi also introduces a slight nonlinearity. Though problematic in theory, these factors did not cause a problem for any of the runs presented here. In any case, the nonlinearity can easily be eliminated by requiring a fixed number of iterations of standard or underrelaxed Jacobi for the coarse grid solve without use of a convergence tolerance.

3.6. Acceleration for the V-cycle preconditioner. To achieve speed and robustness, the AMG V-cycle is used as a preconditioner to a conjugate gradient-type iterative acceleration method. Two methods are considered: the conjugate gradient (CG) method and a restarted flexible GMRES (FGMRES) method.

The iteration terminates when the ratio of the two-norm of the residual vector to the two-norm of the right-hand side vector is less than or equal to a given tolerance.

3.7. Load rebalancing. For irregular problems it is possible for the number of coarse grid points generated on different processors to be nonuniform, even if the original problem is properly load balanced, resulting in uneven workloads across processors. To insure efficient performance on coarse grids, a simple load balancing scheme is applied to the coarse grid matrix at each level to reduce any significant load imbalance with respect to the original problem distribution introduced by the coarsening. In this scheme, a diffusive load rebalancing method is used to migrate subgrid boundary unknowns from overweighted processors to underweighted ones.

3.8. Reducing the number of processors used for coarse grids. For the coarse grid matrices, parallel efficiency is decreased because a small problem is being solved across a large number of processors. In particular, each processor may be connected to a very large number of other processors via the given matrix, which is problematic because of communication latencies. This is not an issue for large numbers of unknowns per processor but may be an issue when the number of unknowns per processor is small and thus communication costs are more dominant (for details regarding the impact of this issue on multigrid performance, see e.g. [5]).

To address this, an operation is performed which reduces the number of processors which store the coarse grid problems, via a redistribution of these problems. In this technique, at each multigrid level the number of processors used to hold the problem is optionally reduced by a power of two. A heuristic is used to determine how much reduction, if any, is to be made, based on the relevant computation and communication costs.

Experiments show that this technique is effective, reducing the total solve time by as

ETNA
Kent State University
etna@mcs.kent.edu

much as a factor of two or more for some cases of smaller numbers of unknowns per processor.

3.9. Processor graph coloring algorithms. Several parts of the parallel AMG algorithm require the formation of a graph incidence matrix of size p on a processor. This includes for example the processor reduction operation mentioned above, pass 2 of Ruge/Stüben coarsening, and the parallel coarsening strategy described in the next section.

These coloring algorithms have order p complexity and are potentially problematic for very large numbers of processors. However, for the experiments described in this paper on up to 3500 processors, these algorithm components did not pose a problem to scalability.

For larger numbers of processors, e.g. for machines such as Blue Gene/L, alternative graph algorithms can be used to circumvent the order p complexity problem. For example, parallel graph coloring algorithms [8] provide a significantly better probabilistic performance bound than the deterministic algorithms described here. Our experience with these algorithms on smaller numbers of processors thus far suggest that they give good results.

4. New parallel coarsening strategy. The coarse grid selection process is the most difficult component of an AMG algorithm to parallelize efficiently. As described in [14] and [15], the uniprocessor coarse grid selection algorithm resembles a greedy MIS algorithm which is inherently sequential.

This algorithm is roughly defined by Algorithm 4.1. The labels C and F refer respectively to the selected set of coarse grid points and the complement of this set. Also, A denotes the matrix at the given level, and A_{strong} denotes the matrix of strong connections in A obtained by dropping entries in A which are small as described in section 2. The value λ_i is initialized to the degree of node i in the adjacency graph of A_{strong}. The procedure cycles until all nodes in this graph are labeled C or F. Nodes $\{i, j\}$ are said to be connected via A_{strong} if and only if $A_{strong}(i, j) \neq 0$.

ALGORITHM 4.1. (*Uniprocessor Coarsening*)
At each level of the AMG hierarchy form A_{strong} *from* A.
Pass 1: Initialize λ_i *to the degree of node* i *in* A_{strong}.
  Until all nodes are labeled C or F points
    Select node i *with maximum* λ_i *and label* i *a C point.*
    Label all nodes connected to i *via* A_{strong} *as F points.*
    Set $\lambda_j = \lambda_j + 1$ *for each node* j *connected to these F points via* A_{strong}
  End
End
Pass 2:
  For each pair of F points $\{i, j\}$ *labeled in pass 1 connected via* A_{strong},
    ensure a common C point by adding new C points if necessary.
End

Pass 1 of Algorithm 4.1 is sequential in nature, tending to sweep through the grid in a wavefront fashion. This typically keeps the number of strong F-F connections small, which is good for convergence. However, sequential calculations are problematic for scalable parallel computations.

Parallelization is obtained by generalizing Algorithm 4.1. A naive approach to parallelization would apply Algorithm 4.1 independently and in parallel to the subgrid of the problem on each processor. However, this approach results in coarsenings which tend to have a large number of strong F-F connections across processor subgrid boundaries, due to labeling mismatches between neighbor processors. Some of these F points could be converted to C

ETNA
Kent State University
etna@mcs.kent.edu

points by applying pass 2 of the algorithm. However, particularly for 3-D problems of interest, the resulting coarse grids would not be coarse enough to control the overall operator complexity of the resulting parallel AMG algorithm.

The novel approach used here *introduces a limited amount of sequentiality* between neighbor processor computations to decrease the number of mismatched subgrid boundary cells. First a coloring of the processors relative to the matrix A_{strong} of strong connections is generated. Pass 1 of the uniprocessor Algorithm 4.1 is then applied to each color in sequence on the processors corresponding to that color. When pass 1 of Algorithm 4.1 is applied to the subgrid on a processor for a given color, this coarsening operation flags some of the unknowns on neighbor processors, to label these unknowns as F points or modify their λ_i values. At the end of processing each color, these modifications are communicated to neighbor processors as a prelude to commencing the next color.

This algorithm makes use of the processor connectivity graph relative to A_{strong}. This is the graph whose number of nodes equals the number of processors and for which two nodes are connected if and only if the matrix A_{strong} has at least one connection between the corresponding processors. Note that this graph is small, thus computations using this graph are very fast.

ALGORITHM 4.2. (*Parallel Coarsening*)
Generate a MIS with respect to the processor graph.
Assign a color value of 1 *to each processor in the resulting MIS.*
Set $i = 2$.
Until all processors are colored
  Generate a MIS with respect to the subgraph of the processor graph
    composed of all processors connected to processors previously colored.
  Assign a color value of i *to each processor in the resulting MIS set.*
  Set $i = i + 1$.
End
Artificially limit the number of colors n_c *generated.*
For each color in sequence
  Perform pass 1 of Algorithm 4.1 on every processor of the given color,
    taking into account possible F/C point selections and λ_i *modifications*
    from neighbor processors for previous colors.
  Communicate all F/C point selections and λ_i *modifications to*
  neighbor processors.
End

For the experiments presented in this paper, an order(p) greedy MIS algorithm is used for computations with the processor connectivity graph, though one may alternatively use a Luby-type parallel randomized MIS algorithm which has a better (probabilistic) computational complexity bound; cf. [13]. Also, if a smaller number of points in the resulting set for the first color is desired, then it is possible to apply alternatively a MIS algorithm to some power of the processor graph, e.g. the square of the processor graph, rather than the processor graph itself. These approaches are not used for the experiments presented here.

The number of colors n_c generated is artificially limited by recoloring all processors with color value higher than n_c to have color n_c. This limits the loss of parallelism of this part of the AMG algorithm to a small amount.

For the numerical experiments presented in this paper, the first AMG coarsening level uses $n_c = 2$. For coarser grids at lower levels, there are fewer unknowns, thus the time required for the calculation of the coarse points is less significant and therefore computa-

ETNA
Kent State University
etna@mcs.kent.edu

tions can be allowed to be more sequential without serious impact on parallel performance. The number of steps allowed at level ℓ is $(n_c)_\ell = 2(N_1/N_\ell)^f$, where N_ℓ is the number of unknowns of the problem at level ℓ and $f = .7$. Thus larger numbers of parallel steps are allowed based on the effective overall grid cell reduction at the given level.

The effect of Algorithm 4.2 is in effect to create small computational wavefronts in the processor grid. It is easy to see that for a connected domain, if the initial MIS were instead chosen to to be a single processor, and n_c were not limited, then the processor coloring would sweep across the processors as a wavefront. This coupled with the fact that the computation on each processor is itself a wavefront sweep shows that the resulting parallel algorithm is very similar to the uniprocessor algorithm, which itself performs a wavefront sweep over the entire global grid.

Introducing some sequentiality in the computation recovers most of the quality of the grids generated by the serial algorithm, Algorithm 4.1. At the same time, the fact that the coarse grid selection consumes a small fraction of the cost of the entire AMG algorithm (typically less than ten percent) makes it feasible to accept a slight sacrifice of parallelism for this part of the algorithm in order to recover scalability of the overall AMG algorithm. This will be demonstrated in the parallel performance data presented later in this paper.

5. Performance considerations.

5.1. AMG parameter settings and code versions. The experiments presented in this paper use several combinations of solver options. This variation of solver options combinations does not imply that problem-specific tuning is required to solve the targeted problems effectively, but rather that for some situations certain adjustments can give some degree of improved performance. The choices are described in Table 5.1.

The first combination of options, denoted "standard" or **S1**, assumes LAMG code version 1.4.5. This set of options is intended to give good performance on general problems without any problem-specific tuning.

The second combination of options, denoted "fast" or **F**, also assumes LAMG code version 1.4.5. This set of options is tuned to give better performance than the standard options for some very well-behaved problems, e.g. Poisson problems. In particular, the improved interpolation and the stronger smoother give better performance for these problems, though as discussed earlier the latter is problematic for more general linear systems.

The final combination of settings, denoted **S2**, assumes a later version of LAMG, code version 1.5.5. This more recent LAMG version includes all the functionality and performance of the earlier version but also includes the processor reduction algorithm described earlier as well as other small performance improvements. These options are identical to the **S1** options except for use of the processor reduction algorithm and a change in the number of smoother steps which was found to give comparable but slightly better performance in many cases.

5.2. Platforms used for experiments. Tests are run on the Compaq QA, QB and QSC platforms at Los Alamos National Laboratory (LANL). Each of these machines is a cluster of Compaq AlphaServer ES45 nodes, each of which is a 4-CPU SMP with EV68 1.25 GHz processors with 16 MB cache and 8, 16 or 32 GB of node main memory. The nodes are connected by a high-speed Quadrics Interconnect fat-tree network. The QA and QB platforms each have 4096 processors, 512 of which are file server nodes, while the QSC platform has 1024 processors, 256 of which are file server nodes. All calculations were performed using 64-bit floating point arithmetic. MPI was used for interprocessor communication. Jobs can be run using either 3 or 4 processors of each 4-processor node; using 3 out of 4 processors per node can result in better performance since one processor is left free to process system tasks for that node.

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 5.1
LAMG Options

Name	**S1**	**F**	**S2**
LAMG Code Version	1.4.5	1.4.5	1.5.5
Pre/Postsmoother Method	Jacobi	JCG	Jacobi
Pre/Postsmoother Iterations	2	4	4
V-cycle Acceleration	CG	FGMRES	CG
Convergence Tolerance	1e-12	1e-12	1e-12
Coarsening Method	A1	A1	A1
Jacobi Relaxation of Interp. Steps	1	2	1
Interpolation Thresholding Factor	.2	.1	.2
Coarsest Matrix Condition Bound	30.	30.	30.
Coarsest Grid Solver	Jacobi	JCG	Jacobi
Coarsest Grid Convergence Tolerance	.1	.1	.1
Load Rebalancing	yes	yes	yes
Processor Reduction	no	no	yes

For numerical experiments, unless otherwise noted wallclock times are reported in seconds.

5.3. Description of model problems. Several classes of model problems are used for numerical experiments.

- *3-D Poisson Problems.* This problem set consists of a sequence of model 3-D Poisson problems based on a 7-point discretization of the equation $-u_{xx}-u_{yy}-u_{zz} = f$ on a regular rectahedral domain with Dirichlet boundary conditions. Each processor has a cube-shaped subgrid of dimensions $n \times n \times n$. These subgrids are combined as tiles to form global grids of the respective sizes. For example, for p processors a problem decomposition of $p_x \times p_y \times p_z$ is used, where $p = p_x \cdot p_y \cdot p_z$. In each test the true solution to the problem is a vector of all 1s; experiments with other right-hand-side vectors gave qualitatively similar results. In each case the initial guess is the zero vector.
- *3-D Discontinuous Problems.* This problem set consists of the following modifications of the Poisson problems defined above. Material discontinuities are introduced. In particular, the diffusion coefficient in the lower half of the domain with respect to the x-coordinate differs from that of the upper half of the domain by a factor of 10^6. Otherwise, the problem specifications are the same as those for the 3-D Poisson problems.
- *3-D Poisson Problems, Higher-Order Discretizations.* This problem set consists of Poisson problems constructed as in the 3-D Poisson problems described above but which utilize higher-order discretizations of a 3-D Poisson problem. Problems using a 19-point stencil and problems using a 27-point stencil are constructed. Otherwise, the problem specifications are the same as those for the 3-D 7-point Poisson problems.
- *2-D Poisson Problems.* This problem set consists of a sequence of model 2-D Poisson problems discretized with 5-point finite differences for the equation $-u_{xx} - u_{yy} = f$ on a regular rectangular domain with Dirichlet boundary conditions. Each processor has a square-shaped subgrid of dimensions $n \times n$. These subgrids are combined as tiles to form global grids of the respective sizes, e.g. for p processors a problem decomposition of $p_x \times p_y$ is used, where $p = p_x \cdot p_y$. Otherwise, the problem specifications are the same as those for the 3-D Poisson problems.

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 5.2
LAMG Performance, 3-D 7-pt Poisson, 1 Processor, **S2** *Settings*

n	Number of Unknowns	AMG Iter	AMG Setup Time	AMG Solve Time	AMG Total Time	JCG Iter	JCG Time
6	216	11	0.01	0.00	0.02	10	0.00
12	1728	12	0.03	0.02	0.06	37	0.01
25	15625	13	0.21	0.20	0.41	82	0.13
50	125000	15	2.37	3.69	6.06	158	3.93
100	1000000	20	26.78	53.60	80.39	312	91.13
200	8000000	19	230.44	444.72	675.15	611	1512.65

5.4. Single processor performance. Table 5.2 summarizes single processor results for 3-D 7-point Poisson problems with increasing numbers of unknowns. LAMG options **S2** as described in Table 5.1 are used. For comparison, results from applying Jacobi CG to the same problems with the same convergence tolerance are included.

For LAMG one can observe a slow increase in iteration count as the problem size is increased. This compares very favorably with the corresponding scaling of JCG, whose iteration count for this problem set roughly *doubles* whenever n is doubled. Also, as is typical for AMG solvers of this type, the AMG setup time consumes a significant fraction of the total time, in this case 35-50 percent.

For comparison purposes, Table 5.3 presents results obtained using the uniprocessor AMG1R6 code authored by Ruge, Stüben and Hempel. AMG1R6 implements Ruge/Stüben AMG as defined in [14]. The problems being solved, the platform used and stopping test used for this case are the same as those used to generate Table 5.2. A comparison of the iteration counts for LAMG, JCG and AMG1R6 is given in Figure 5.1. The corresponding solve times per unknown are compared in Figure 5.2.

It should be noted that it is difficult to compare results from two different codes in this way. Not only the algorithms but also the implementations are different, which can cause significant differences in runtime performance, e.g. due to cache effects for different problem sizes on the given platform. Nonetheless, some conclusions can be drawn from the comparisons.

Significantly, for LAMG and for AMG1R6, the growth rate of the iteration counts as the problem size is increased is roughly the same, indicating the same basic algorithmic scaling behavior of the two codes. The absolute cost in numbers of iterations is higher for LAMG than for AMG1R6. This is to be expected on account of the aggressive coarsening used in LAMG. Furthermore, the total timings of the two codes are comparable, with LAMG actually faster than AMG1R6 for the largest problem solved. We conclude that on a single processor LAMG performance is as scalable as the performance of the classical Ruge/Stüben AMG method, which is widely considered to be an effective scalable algorithm for serial computers. Finally, it should be noted that this iteration count scalability behavior is not specific to the Poisson problem or to the serial case; the 3-D discontinuous parallel results shown in the next section manifest similar iteration count scalability to the 3-D Poisson problems considered here.

Tables 5.4 and 5.5 give additional data for the same runs. Included are the number of multigrid levels; the number of unknowns associated with the coarsest grid; the maximum value over all multigrid levels of the average number of nonzeros per row of the matrix at the given level; the grid complexity, which is the sum of the matrix number of unknowns for all levels divided by the number of unknowns of the original matrix; and the operator complexity,

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 5.3
AMG1R6 Performance, 3-D 7-pt Poisson, 1 Processor

n	Unknowns	Iterations	Setup Time	Solve Time	Total Time
6	216	7	0.00	0.00	0.00
12	1728	8	0.03	0.00	0.03
25	15625	9	0.40	0.09	0.49
50	125000	10	4.53	1.50	6.03
100	1000000	11	54.70	20.02	74.72
200	8000000	13	726.61	200.46	927.07

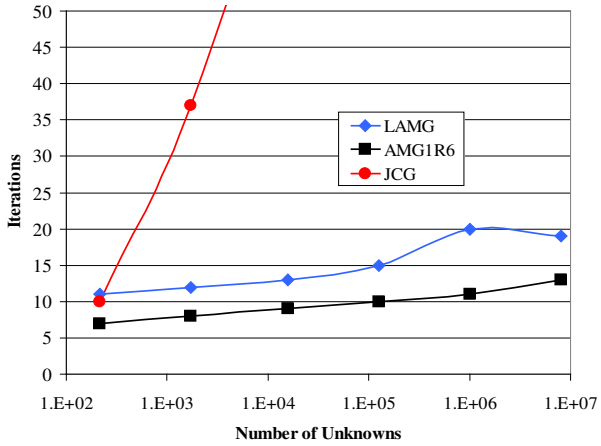

FIG. 5.1. *LAMG, JCG and AMG1R6 Iteration Counts*

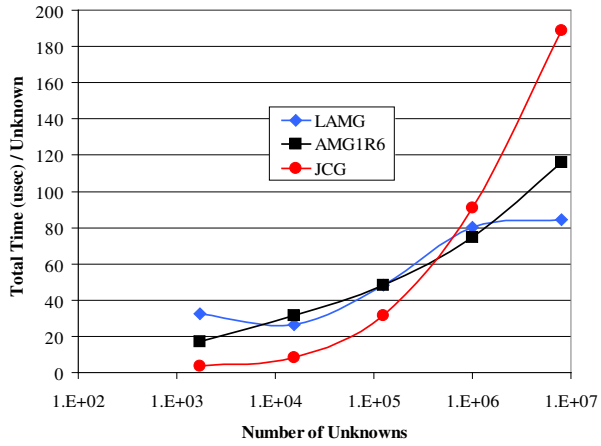

FIG. 5.2. *LAMG, JCG and AMG1R6 Solve Time per Unknown*

which is the sum of the matrix nonzero counts for all levels divided by the number of nonzeros of the original matrix.

Significantly, the maximum average nonzeros per row, which is a measure of matrix sparsity and its impact on communication for the parallel case, is strongly bounded for LAMG, showing very small growth with problem size, while for AMG1R6 this quantity grows very rapidly. The stencil growth in the latter case makes the algorithm impractical for parallel

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 5.4
LAMG Run Data, 3-D 7-pt Poisson, 1 Processor, **S2** *Settings*

n	Multigrid Levels	Size of Coarsest Grid	Maximum Average Nonzeros/Row	Grid Complexity	Operator Complexity
6	2	14	9.14	1.099	1.065
12	3	4	13.98	1.136	1.065
25	3	28	16.08	1.136	1.057
50	4	6	22.61	1.167	1.064
100	4	39	26.23	1.172	1.064
200	5	7	31.06	1.176	1.064

TABLE 5.5
AMG1R6 Run Data, 3-D 7-pt Poisson, 1 Processor

n	Multigrid Levels	Maximum Average Nonzeros/Row
6	6	17.48
12	8	44.92
25	11	105.75
50	13	255.94
100	15	545.51
200	18	1049.70

implementation for problems such as this, due to the excessive communication requirements.

It should also be noted that the number of levels for LAMG is much less than for AMG1R6. Aggressive coarsening as well as the condition number criterion for terminating coarsening causes this effect. This improves parallel performance also.

5.5. Aggressive coarsening effects. The algorithm described in this paper differs from those of [15] in that the former uses aggressive coarsening on all multigrid levels while the latter do so only for the coarsening of the finest grid level.

To compare these two approaches, performance data is presented in Table 5.6 using the same problems and solver settings as in Table 5.2, except that aggressive coarsening is used only for the first level and standard Ruge/Stüben (pass 1) coarsening is used for all other levels. Multipass interpolation is used for all levels.

It should be noted that, though the results of Table 5.6 for the maximum average nonzeros per row are considerably better than those of AMG1R6 (Table 5.5), the behavior of this statistic is still not scalable. This growth in the average stencil size is not favorable for parallel communication, thus motivating the need to coarsen aggressively on all multigrid levels.

5.6. Performance of parallel coarsening. The results presented thus far are for a single processor. Now we present parallel performance data to examine the impact of the parallel coarsening strategy on AMG performance.

For these experiments we use LAMG with settings **S2** applied to 3-D Poisson problems on 1, 8, 27 and 64 processors.

One aspect of performance potentially affected by the parallel coarsening is the convergence rate of the method. Figure 5.3 shows the number of iterations required to converge for the test cases. It can be observed that the impact of parallelization on convergence is minimal, in the worst case increasing the iteration count by a factor of about 20 percent. This suggests

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 5.6
LAMG Run Data, 3-D 7-pt Poisson, 1 Processor, **S2** *Settings, Aggressive Coarsening on One Level Only*

n	Multigrid Levels	Maximum Average Nonzeros/Row
6	2	9.14
12	3	18.33
25	4	46.86
50	5	72.23
100	5	126.20
200	7	227.41

TABLE 5.7
Impact of LAMG Parallel Coarsening on Maximum Average Nonzeros per Row

n	Unknowns / Proc	1 Proc	8 Procs	27 Procs	64 Procs
6	216	9.14	15.87	21.64	23.03
12	1728	13.98	17.68	23.21	25.99
25	15625	16.08	22.81	26.38	28.69
50	125000	22.61	28.02	30.49	31.79
100	1000000	26.23	30.83	31.89	32.72

that the parallelization strategy for the coarsening has only a small effect on the convergence of the method.

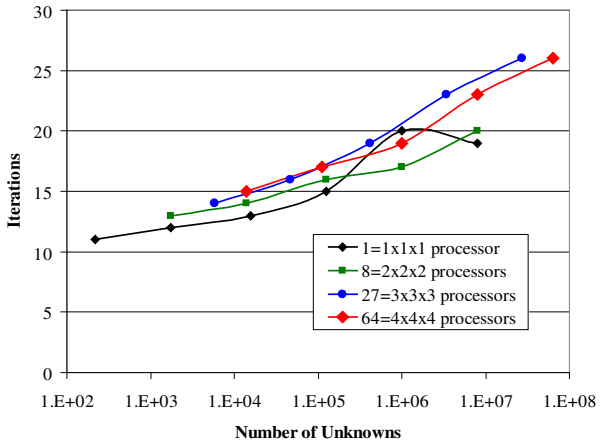

FIG. 5.3. *Impact of LAMG Parallel Coarsening on Iteration Count*

One can also consider the impact of the parallel coarsening on the maximum average nonzeros per row statistic discussed earlier, which indicates the communication bandwidth required to manipulate the sparse matrices generated by the algorithm. Table 5.7 shows for each of the test cases the maximum average nonzeros per row for the coarse grid matrices generated for the multigrid levels. The values are only slightly greater for the parallel cases compared to the serial cases and indicate little dependence on the number of processors, implying that the communication burden should be acceptable.

Finally we consider the cost of actually performing the parallel coarsening. For this we analyze a representative case from the above tests, namely the case of 64 processors and one

ETNA
Kent State University
etna@mcs.kent.edu

million unknowns per processor.

Two cost issues need be examined: the cost of the parallel gathers required for the parallel steps of the coarsening, and the impact of sacrificing some parallelism by introducing sequentiality to the coarsening process.

Table 5.8 presents performance data for this case. It should be noted that an aggressive coarsening step makes use of several passes of standard coarsening—in this case, three for each multigrid level. For every pass of standard coarsening on each level, the table shows the number of colors or parallel steps employed to perform the coarse grid selection; the number of unknowns per processor taken as input to the coarse grid selection; this quantity normalized to the problem size per processor of the original problem; and a cost estimate, which is the number of parallel steps multiplied by the normalized unknowns per processor, the latter being a measure of the relative computation cost per parallel step.

The number of parallel steps per level is no more than about 35. This is the number of parallel gathers required to do the parallel coarsening. This quantity is reasonable, since many more gathers than this are required to perform other AMG operations, e.g. the smoother steps for each level. Furthermore, other experiments indicate that this factor weakly depends on the number of processors, so scalability should not be adversely affected.

The sum of the estimated costs for all passes is about 5. This is an estimate of the factor of increase in wallclock time to perform the computations for the parallel coarsening, relative to the time to compute the coarse grid if all processors could coarsen independently. In other words, this is the performance penalty factor from introducing sequentiality into the coarsening process.

Since on a single processor the cost of the coarsening is small relative to the entire solve, multiplying the coarsening time by a factor of 5 has a noticeable but relatively minor effect on the solution time. For example, if the cost for coarsening were 5 percent of the total solve time for a single processor, in parallel the added expense would be limited to 20 percent of the entire solve time. Furthermore, the way the number of colors is calculated as described in the previous section imposes a limit on how much this loss can be, independent of the number of processors. Thus scalability for large numbers of processors is not impacted.

Thus we expect the expense of the parallel coarsening to be minimal. This evaluation is borne out by the parallel performance results presented in the next section.

6. Numerical results. In this section scalability results are presented for large numbers of processors.

In these experiments the total solve time over a set of test problems is measured. To test scalability, the number of unknowns per processor is kept nearly fixed, the number of processors is increased, and the resulting solution wallclock times are reported.

It would be expected for any solver method, including AMG, that the wallclock solve times on more than one processor would increase modestly as the number of processors is increased slightly above one processor, due to the introduction of interprocessor communication. For the structured grid cases considered in this section it is expected that the solution time would increase significantly until there exists a processor subgrid for which all four edges of the subgrid square (2-D) or all six faces of the subgrid cube (3-D) require communication. For 2-D structured problems this occurs at 9 processors; for 3-D structured problems this occurs at 27 processors.

6.1. 3-D Poisson problems. Tests are presented here for 3-D 7-point Poisson problems described in Subsection 5.3 using from 1 to 3584 processors of the LANL QB platform. As the number of processors is increased, the number of unknowns is increased from 1 million to 3.584 billion unknowns. For these experiments, each processor has a subgrid of size $100 \times$

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 5.8
Estimated Cost of Parallel Coarsening

Multigrid Level	Pass	Colors	Unknowns/ Processor	Unc/Proc Normalized	Estimated Cost
1	1	2	1000000.00	1.000	2.000
1	2	4	497420.83	0.497	1.990
1	3	4	123270.22	0.123	0.493
2	1	8	61207.19	0.061	0.490
2	2	8	12278.22	0.012	0.098
2	3	11	1777.61	0.002	0.020
3	1	11	959.44	0.001	0.011
3	2	11	146.70	0.000	0.002
3	3	10	36.78	0.000	0.000
4	1	11	21.30	0.000	0.000
4	2	10	4.63	0.000	0.000
4	3	11	1.66	0.000	0.000
5	1	10	0.81	0.000	0.000
5	2	14	0.47	0.000	0.000
5	3	9	0.22	0.000	0.000
Sums:		134			5.103

100×100, corresponding to one million unknowns per processor. The fast LAMG settings **F** described in Table 5.1 are used.

Table 6.1 summarizes the results of these runs using all 4 processors on each compute node. For selected cases, iteration counts and timings for a Jacobi conjugate gradient (JCG) solver applied to the same problems with the same convergence tolerance are given for comparison purposes. The wallclock times are also shown in Figure 6.1.

Observe that the iteration counts and the wallclock timings for the LAMG solver are very flat as the number of processors is increased, indicating good scalability. Some increase in solve time is evident, particularly as the number of processors is increased from 1 to 27. This increase is in part due to communication costs and in part due to the number of processors being used per compute node increasing from 1 to 4. This is slightly problematic for this computer architecture due to the need for one of the processors of each compute node to interrupt to service system requests, as well as the fact that more processors per node are required to share the same memory, causing memory bandwidth speed constraints. These performance penalties are not specific to LAMG but adversely impact any iterative method, including the Jacobi CG method used here.

Since a node of QB requires some processor cycles to handle system operations for that node, the tests depicted in Table 6.1 were rerun using only 3 out of the 4 CPUs per compute node. The results of those tests are summarized in Table 6.2 and Figure 6.2 which clearly illustrate the effects of the system demands upon that fourth processor on the wallclock timings. It is evident from Table 6.2 and Table 6.1 that the scalability performance of LAMG is significantly improved using this improved hardware configuration. In fact, as the number of processors is increased from 27 to 1000 and thus the problem size is increased by over two orders of magnitude, the total LAMG wallclock solution time increases by only 22 percent, an indication of excellent scalability.

6.2. 3-D discontinuous problems. Practical physical problems may involve more than one material and correspond to materials with very different properties. The test problems used for this set of experiments are 3-D 7-point discontinuous problems involving two ma-

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 6.1
LAMG and JCG Performance, 3-D 7-Point Poisson, 100^3 *Subgrid per Processor, 4 CPUs per Node, QB*

Number of Processors	Processor Tiling Grid	AMG Time	JCG Time	AMG Iter	JCG Iter
1	$1 \times 1 \times 1$	79	95	10	312
8	$2 \times 2 \times 2$	127	249	12	611
27	$3 \times 3 \times 3$	151	373	12	891
64	$4 \times 4 \times 4$	158	503	12	1181
125	$5 \times 5 \times 5$	178	—	12	—
216	$6 \times 6 \times 6$	180	—	12	—
343	$7 \times 7 \times 7$	210	—	12	—
512	$8 \times 8 \times 8$	202	—	12	—
729	$9 \times 9 \times 9$	233	—	12	—
1000	$10 \times 10 \times 10$	218	—	12	—
1331	$11 \times 11 \times 11$	258	—	12	—
1728	$12 \times 12 \times 12$	251	2041	12	3318
2197	$13 \times 13 \times 13$	306	—	12	—
2744	$14 \times 14 \times 14$	290	—	12	—
3375	$15 \times 15 \times 15$	342	—	12	—
3584	$16 \times 16 \times 14$	312	—	12	—

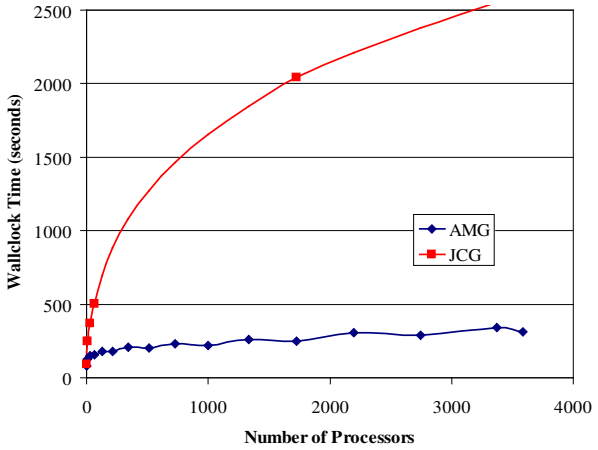

FIG. 6.1. *LAMG and JCG Performance, 3-D 7-Point Poisson,* 100^3 *Subgrid per Processor, 4 CPUs per Node, QB*

terials with diffusion coefficient ratio of one million (see Subsection 5.3). For these experiments, each processor has a subgrid of size $100 \times 100 \times 100$, corresponding to one million unknowns per processor. The standard LAMG settings **S1** described in Table 5.1 are used.

The results of the tests are shown in Table 6.3 and Figure 6.3. We observe that the scalability of the LAMG wallclock times is good, though there is some performance degradation for larger numbers of processors. However, as observed in Subsection 5.4, a slight increase in iteration count with problem size is typical of AMG methods applied to problems of this type and is considered to be good scalability. Also, these tests use all 4 processors per compute node on QB, which accounts for some of the performance loss. The solve time and scalability are vastly improved compared to JCG, which is used in many parallel applications because of its simplicity and minimal communication requirements.

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 6.2
LAMG and JCG Performance, 3-D 7-Point Poisson, 100^3 *Subgrid per Processor, 3 CPUs per Node, QB*

Number of Processors	Processor Tiling Grid	AMG Time	JCG Time
1	$1 \times 1 \times 1$	80	94
8	$2 \times 2 \times 2$	120	238
27	$3 \times 3 \times 3$	139	359
64	$4 \times 4 \times 4$	145	487
125	$5 \times 5 \times 5$	153	—
216	$6 \times 6 \times 6$	155	—
343	$7 \times 7 \times 7$	165	—
512	$8 \times 8 \times 8$	165	—
729	$9 \times 9 \times 9$	174	—
1000	$10 \times 10 \times 10$	170	—

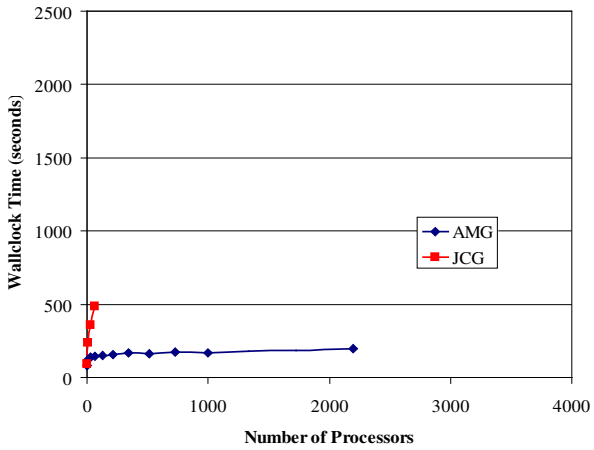

FIG. 6.2. *LAMG and JCG Performance, 3-D 7-Point Poisson,* 100^3 *Subgrid per Processor, 3 CPUs per Node, QB*

6.3. 3-D Poisson problems, higher-order discretizations. It is of interest to consider the effects of larger stencils on performance. Results of experiments using the test problems described in Subsection 5.3 generated using 3-D 19-point and 27-point stencils are shown in Table 6.4 and Figure 6.4. For these cases, each per-processor subgrid has dimensions $80 \times 80 \times 80$, resulting in 512,000 unknowns per processor. The fast LAMG settings **F** described in Table 5.1 are used.

The observed LAMG wallclock time performance is highly scalable. The smaller per-processor subgrid size $80 \times 80 \times 80$ results in a larger communication to computation ratio, leading to slightly less scalable performance than the 3-D 7-point stencil case. Each test was run on QB using all 4 processors on each compute node.

6.4. 2-D Poisson problems. Parallelization efforts have focused on the operator complexity issues which arise with 3-D problems. However, 2-D problems are also of interest in many applications. Table 6.5 summarizes the scalability performance of LAMG across the solution of a sequence of 2-D Poisson problems as described in Subsection 5.3. The corresponding LAMG wallclock times are shown in Figure 6.5.

For this case, each processor has a subgrid of size 1800×1800, resulting in roughly 3.2

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 6.3
LAMG and JCG Performance, 3-D 7-Point Discontinuous, 100^3 Subgrid per Processor, 4 CPUs per Node, QB

Number of Processors	Processor Tiling Grid	AMG Time	JCG Time	AMG Iter	JCG Iter
1	$1 \times 1 \times 1$	89	115	23	384
8	$2 \times 2 \times 2$	147	299	28	733
27	$3 \times 3 \times 3$	189	440	36	1051
64	$4 \times 4 \times 4$	211	583	39	1369
125	$5 \times 5 \times 5$	217	—	38	—
216	$6 \times 6 \times 6$	234	—	41	—
343	$7 \times 7 \times 7$	241	—	39	—
512	$8 \times 8 \times 8$	265	—	44	—
729	$9 \times 9 \times 9$	287	—	43	—
1000	$10 \times 10 \times 10$	295	—	45	—
1331	$11 \times 11 \times 11$	315	—	43	—
1728	$12 \times 12 \times 12$	322	2170	45	3562
2197	$13 \times 13 \times 13$	350	—	44	—
2744	$14 \times 14 \times 14$	382	—	49	—
3375	$15 \times 15 \times 15$	408	—	45	—
3584	$16 \times 16 \times 14$	429	—	51	—

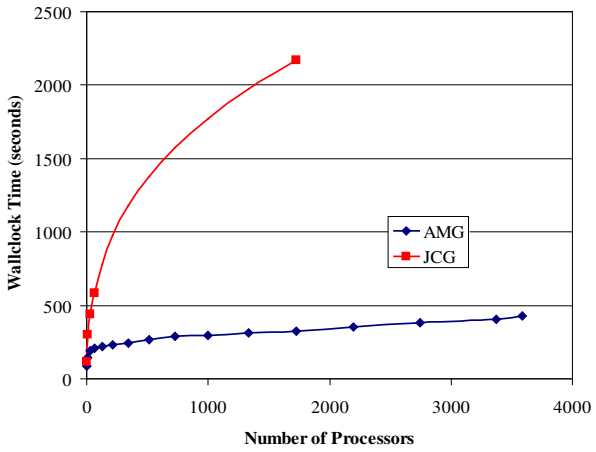

FIG. 6.3. *LAMG and JCG Performance, 3-D 7-Point Discontinuous, 100^3 Subgrid per Processor, 4 CPUs per Node, QB*

million unknowns per processor. The LAMG solution settings **F** in Table 5.1 are used, and only 3 out of the 4 processors per compute node are used.

For these 2-D test problems, on a single processor the LAMG solver is already more than an order of magnitude faster than the Jacobi CG solver. Also, based on known scaling behavior of Jacobi CG, one would expect that AMG would be over *two* orders of magnitude faster than Jacobi CG for the largest case run, and nearly *three* orders of magnitude faster at 3500 processors. Furthermore, LAMG exhibits nearly perfect scalability: as the number of processors is increased from 9 to 289, the performance loss is only 6 percent.

6.5. 3-D SAGE problems. The following experiments make use of test problems generated using a modified version of the Los Alamos National Laboratory / Science Applications International Corporation (SAIC) SAGE code. SAGE is an Adaptive Grid Eulerian code, a

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 6.4
LAMG Performance, 3-D 19- and 27-Point Poisson, 100^3 *Subgrid per Processor, 4 CPUs per Node, QB*

Number of Processors	Processor Tiling Grid	AMG Time	AMG Iter	AMG Time	AMG Iter
		19-Point	19-Point	27-Point	27-Point
1	$1 \times 1 \times 1$	59	10	73	8
8	$2 \times 2 \times 2$	87	11	112	10
27	$3 \times 3 \times 3$	101	12	141	10
64	$4 \times 4 \times 4$	110	13	148	10
125	$5 \times 5 \times 5$	112	12	165	10
216	$6 \times 6 \times 6$	115	12	166	10
343	$7 \times 7 \times 7$	136	13	190	10
512	$8 \times 8 \times 8$	138	13	195	11
729	$9 \times 9 \times 9$	152	13	220	11
1000	$10 \times 10 \times 10$	157	13	218	11
1331	$11 \times 11 \times 11$	193	13	252	11

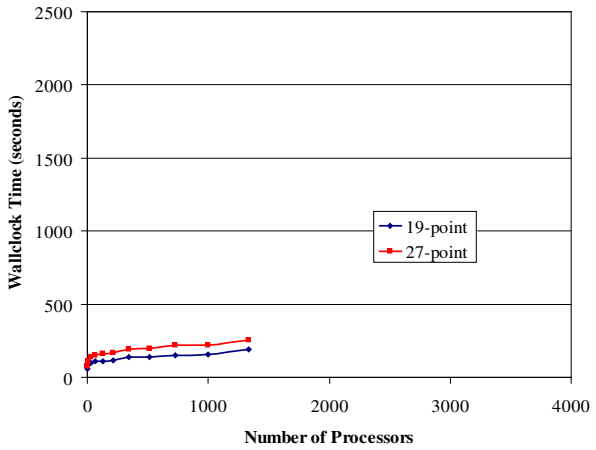

FIG. 6.4. *LAMG Performance, 3-D 19- and 27-Point Poisson,* 100^3 *Subgrid per Processor, 4 CPUs per Node, QB*

multidimensional multimaterial hydrodynamics code with adaptive mesh refinement (see e.g. [11]) which was modified by Tom Betlach of SAIC to generate test problems which simulate radiation diffusion problems with material discontinuities and complex geometry.

The problem set is composed of 3-D heat conduction problems involving three materials and three levels of mesh refinement. The grid is generated using the SAGE adaptive mesh generator which allows for the successive generation of complex grids by using overlays to replace the existing grid. The geometry of the object is a cube composed of Material 1. Within this cube is a sphere composed of Material 2. Within this sphere is a spherical shell composed of Material 3. The shell 's interior is composed of Material 2, as are two connecting pipes which connect the interior of the shell to the exterior.

The regions of the domain have diffusion coefficients with ratios equal to one million. These problems are designed to represent difficulties which arise in typical SAGE workflows which may pose a challenge to linear solvers, including large diffusion coefficients with strong discontinuities, complex gridding and multiple levels of adaptive mesh refinement. The simulation involves one timestep designed to be representative of a single timestep

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 6.5
LAMG and JCG Performance, 2-D 5-Point Poisson, 1800^2 *Subgrid per Processor, 3 CPUs per Node, QSC*

Number of Processors	Processor Tiling Grid	AMG Time	JCG Time	AMG Iter	JCG Iter
1	1×1	249	3533	12	3759
4	2×2	317	8429	12	7398
9	3×3	365	—	13	—
16	4×4	373	—	13	—
25	5×5	374	—	13	—
36	6×6	381	—	13	—
49	7×7	379	—	13	—
64	8×8	381	—	13	—
81	9×9	389	—	13	—
100	10×10	385	—	13	—
121	11×11	386	—	13	—
144	12×12	385	—	13	—
169	13×13	387	—	13	—
196	14×14	386	—	13	—
225	15×15	386	—	13	—
256	16×16	391	—	13	—
289	17×17	388	—	13	—

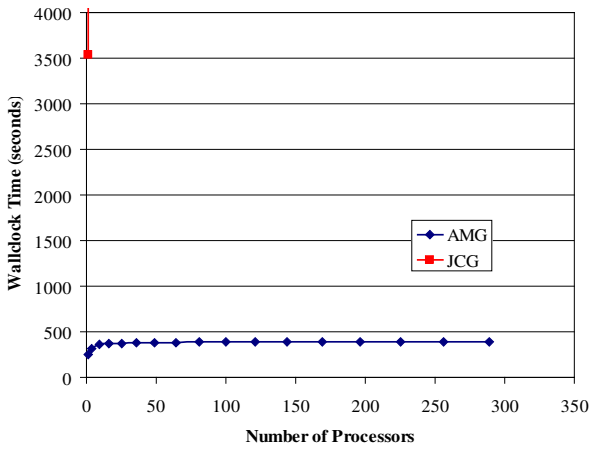

FIG. 6.5. *LAMG and JCG Performance, 2-D 5-Point Poisson,* 1800^2 *Subgrid per Processor, 3 CPUs per Node, QSC*

of a longer simulation.

Problems are generated such that the number of unknowns per processor is approximately constant. Tests with these problems cannot be viewed as scaling tests in the same sense as the model problem tests presented above, since the problem difficulty depends not only on the number of unknowns but also on the specific adaptive grid generated.

For all cases, the relative residual convergence tolerance for both LAMG and JCG is 10^{-12}. In each test all 4 processors per compute node are used.

For the first set of tests, the number of unknowns per processor is kept between 710,000 and 750,000 unknowns. The largest problem considered has 181.4 million variables on 256 processors. These tests were run on the QSC computer at LANL in a shared user environment

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 6.6
LAMG and JCG Performance, 3-D SAGE Problems, $710,000$ *to* $750,000$ *Unknowns per Processor, 4 CPUs per Node, QSC*

Number of Processors	AMG Time	JCG Time	Number of Unknowns
1	96	118	735,288
2	116	178	1,507,458
4	146	263	3,065,728
8	152	307	5,925,400
16	175	528	11,958,784
32	199	651	24,977,000
64	226	805	46,065,280
128	247	1005	90,239,544
256	326	1338	181,444,416

using the **S1** LAMG settings listed in Table 5.1.

The results are shown in Table 6.6 and Figure 6.6. These results indicate that LAMG scales very well and also vastly outperforms Jacobi CG in both runtime and scalability with respect to problem size and number of processors.

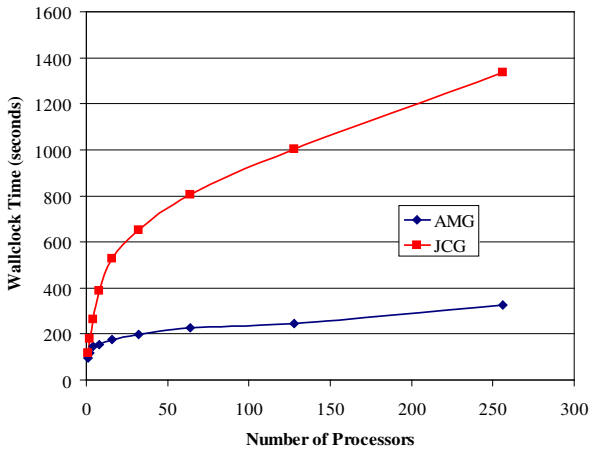

FIG. 6.6. *LAMG and JCG Performance, 3-D SAGE Problems,* $710,000$ *to* $750,000$ *Unknowns per Processor, 4 CPUs per Node, QSC*

In the next set of tests, the number of unknowns per processor is approximately $250,000$. The largest problem considered has about 350 million variables on 1408 processors. These tests were run on the QB computer at LANL in a shared user environment using the **S2** LAMG settings listed in Table 5.1, with the exception that the more aggressive (4,4) coarsening was used rather than A1 coarsening. The (4,4) coarsening was required to cope with the communication demands required by this 3-D problem with complex physics and smaller numbers of unknowns per processor.

The results are shown in Figure 6.7. Again, both solve time and scalability are much improved for LAMG compared to the JCG solver. These results confirm the robustness and scalability of the algorithm on difficult real-world problems.

6.6. 2-D RAGE problems. The following experiments use the LANL/SAIC RAGE code. The RAGE code is a version of the SAGE code described above which also includes

ETNA
Kent State University
etna@mcs.kent.edu

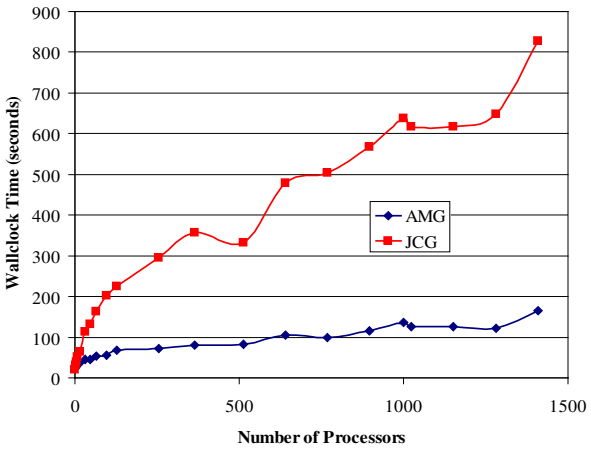

FIG. 6.7. *LAMG and JCG Performance, 3-D SAGE Problems,* $250,000$ *Unknowns per Processor, 4 CPUs per Node, QB*

TABLE 6.7
LAMG and JCG Performance, 2-D RAGE Problems, 250,000 Unknowns, 4 Processors, QB

	AMG	JCG
Total simulation time (hours)	25.5	37.1
Time in solver (hours)	1.76	13.39
Total solver iterations	3.4e4	4.05e6

radiation diffusion and radiation-material energy coupling.

The problem solved is a simulation of a laser-driven hohlraum high density energy physics experiment. The problem geometry is a 3-D cylindrically symmetric domain, reduced to a 2-D domain for the simulation. Radiation transport is modeled with a gray diffusion model. The problem has material discontinuities which are captured using an unstructured AMR grid with up to 8 levels of mesh refinement.

The simulation is run over 1300 time steps on 4 processors of the LANL QB platform. The linear systems each have approximately 250,000 unknowns. One linear solve is performed for each time step. Tests on these problems use the **S2** LAMG settings listed in Table 5.1.

The simulation timings are shown in Table 6.7. The LAMG solver reduces the solve times by a factor of 7.6 compared to the Jacobi CG solver. Notably, this is a huge cost savings for a relatively small problem; larger problems should expect even more savings. Also, the cost of the linear solver has been reduced to a nearly inconsequential 7 percent of the entire simulation time, which attains a difficult objective for simulations requiring a linear solver, namely, making the linear solve time negligible.

6.7. 2-D Zathras problems. The following experiments make use of the Zathras code developed at Los Alamos National Laboratory [3]. Zathras is a non-equilibrium radiative transport code that solves the gray radiation transport equation using the diffusion or the P1 first order spherical harmonic approximation on 2-D and 3-D unstructured grids with adaptive mesh refinement.

The problem being solved is radiative transfer in a 2-D domain with material discontinuities of two orders of magnitude or more that vary over time. This is a structured problem with 22,400 unknowns per processor run on 1 to 512 processors of the LANL QB platform. An example solution profile is shown in Figure 6.8. For the simulation, the number of time

ETNA
Kent State University
etna@mcs.kent.edu

steps increases from 400 to 900 as the problem size is increased. For more details, see [3]. Nearly all of the simulation time is spent in the solver.

Tests on these problems use the **S2** LAMG settings listed in Table 5.1. Two other solvers are compared: SSOR CG with manually set relaxation parameter, and an overlapping approximate block Jacobi or additive Schwarz CG with 1 step of SSOR used for the subgrid solver and one layer of cells overlap. The convergence tolerance for the solves is 10^{-8}.

Two observations should be made regarding these experiments. First, because the number of time steps is increased as the problem size increases, one would not expect the total simulation time to be independent of the number of processors, but rather one would expect a slow increase in simulation time, even if the solver is perfectly scalable. Second, this number of unknowns per processor is comparatively small for LAMG, so one would expect significant communication requirements for this case.

In spite of this, the results shown in Figure 6.9 show LAMG is not only faster than the one-level solvers for large numbers of processors but also the scalability of LAMG compared to the other solvers is better as more processors are added.

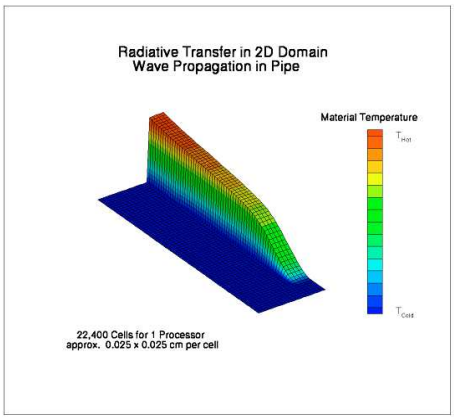

FIG. 6.8. *Solution Profile, 2-D Zathras Problem,* 22, 400 *Unknowns per Processor*

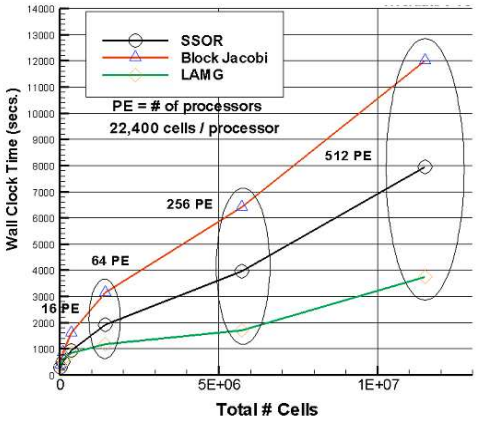

FIG. 6.9. *Solver Performance, 2-D Zathras Problem,* 22, 400 *Unknowns per Processor, QSC*

7. Conclusions. The new AMG algorithm described here and implemented in the LAMG package is an effective, scalable parallel algebraic multigrid method. Performance data for

ETNA
Kent State University
etna@mcs.kent.edu

2-D and 3-D problems on up to 3584 processors indicates highly scalable behavior and one to two orders of magnitude speed improvement or more for large problems compared to the competing Jacobi-preconditioned conjugate gradient method. This new AMG method is also shown to be robust for difficult problems involving complex physics and gridding.

Acknowledgments. The authors would like to thank John Ruge, Klaus Stüben and Michael DeLong for helpful discussions regarding AMG. The authors would also like to thank Tom Betlach of SAIC for his assistance in the use of the SAGE code and in the generation of test problems in his modified version of SAGE. In addition, the authors would like to thank Dave Carrington of LANL for providing Zathras performance data and Don Haynes for providing RAGE performance data. Finally, the authors would like to thank the reviewers for their helpful comments.

REFERENCES

[1] E. CHOW, R. FALGOUT, J. HU, R. TUMINARO AND U. YANG, *A survey of parallelization techniques for multigrid solvers*, in M. Heroux, P. Raghavan, and H. Simon, eds., *Frontiers of Parallel Processing for Scientific Computing*, SIAM, Philadelphia, 2006, to appear.

[2] A. CLEARY, R. FALGOUT, V. HENSON AND J. JONES, *Coarse-grid selection for parallel algebraic multigrid*, in A. Ferreira, J. Rolim, H. Simon and S-H. Teng, eds., *Proceedings of the Fifth International Symposium on Solving Irregularly Structured Problems in Parallel*, Lecture Notes in Comput. Sci., Vol. 1457, Springer, New York, 1998, pp. 104–115.

[3] D. B. CARRINGTON AND V. A. MOUSSEAU, *Preconditioning and solver optimization ideas for radiative transfer*, in Proceedings of the ASME Summer Heat Transfer Conference 2005, (San Francisco, CA, July 17–22, 2005), ASME, New York, 2005.

[4] H. C. ELMAN, O. G. ERNST AND D. P. O'LEARY, *A multigrid method enhanced by Krylov subspace iteration for discrete Helmholtz equations*, SIAM J. Sci. Comput., 23 (2001), pp. 1291–1315.

[5] S. GOEDECKER AND A. HOISIE, *Performance Optimization of Numerically Intensive Codes*, SIAM, Philadelphia, 2001.

[6] M. J. HAGGER, A. SPENCE AND K. A. CLIFFE, *Two grid iteration with a conjugate gradient smoother applied to a groundwater flow model*, University of Bath, 1994.

[7] V. HENSON AND U. YANG, *BoomerAMG: a parallel algebraic multigrid solver and preconditioner*, Appl. Numer. Math., 41 (2002), pp. 155–177.

[8] M. T. JONES AND P. E. PLASSMAN, *A parallel graph coloring heuristic*, SIAM J. Sci. Comput., 14 (1993), pp. 654–669.

[9] W. JOUBERT, *LAMG: Los Alamos Algebraic Multigrid Code Reference Manual*, Release 1.7.2, Los Alamos National Laboratory, Report LA-UR 05-5000, May 2005.

[10] W. JOUBERT, *LAMG: Los Alamos Algebraic Multigrid Code User's Manual*, Release 1.7.2, Los Alamos National Laboratory, Report LA-UR 05-4041, May 2005.

[11] D. J. KERBYSON, H. J. ALME, A. HOISIE, F. PETRINI, H. J. WASSERMAN AND M. GITTINGS, *Predictive performance and scalability modeling of a large-scale application*, in Proceedings of the 2002 ACM/IEEE Conference on Supercomputing (CDRom), (Denver, CO, November 10–16, 2001), ACM Press, New York, 2001.

[12] A. KRECHEL AND K. STÜBEN, *Parallel algebraic multigrid based on subdomain blocking*, Parallel Comput., 27 (2001), pp. 1009–1031.

[13] M. LUBY, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1053.

[14] J. RUGE AND K. STÜBEN, *Algebraic Multigrid (AMG)*, in S. McCormick, ed., *Multigrid Methods*, Frontiers in Applied Mathematics, Vol. 5, SIAM, Philadelphia, 1986.

[15] K. STÜBEN, *Algebraic multigrid (AMG): an introduction with applications*, in U. Trottenberg, A. Schuller and C. Oosterlee, eds., *Multigrid*, Academic Press, 2000, pp. 413–532.

[16] R. TUMINARO AND C. TONG, *Parallel smoothed aggregation multigrid: aggregation strategies on massively parallel machines*, in J. Donnelley, ed., Proceedings of the 2000 ACM/IEEE Conference on Supercomputing (CDRom), (Dallas, TX, November 4–10, 2000), ACM Press, New York, 2000.

[17] P. VANEK, J. MANDEL AND M. BREZINA, *Algebraic multigrid based on smoothed aggregation for Second and Fourth Order Problems*, Computing, 56 (1996), pp. 179–196.