NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2007; **14**:611–643
Published online 25 July 2007 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nla.541

Parallel coarse-grid selection

David M. Alber*, † and Luke N. Olson

Siebel Center for Computer Science, University of Illinois at Urbana-Champaign, 201 North Goodwin Avenue, Urbana, IL 61801, U.S.A.

SUMMARY

Algebraic multigrid (AMG) is a powerful linear solver with attractive parallel properties. A parallel AMG method depends on efficient, parallel implementations of the coarse-grid selection algorithms and the restriction and prolongation operator construction algorithms. In the effort to effectively and quickly select the coarse grid, a number of parallel coarsening algorithms have been developed. This paper examines the behaviour of these algorithms in depth by studying the results of several numerical experiments. In addition, new parallel coarse-grid selection algorithms are introduced and tested. Copyright © 2007 John Wiley & Sons, Ltd.

Received 12 September 2006; Revised 25 January 2007; Accepted 25 May 2007

KEY WORDS: algebraic multigrid; coarsening; parallel computing

1. INTRODUCTION

Engineers and scientists are faced with problems too large for computation on a single processor. For that reason, parallel linear solvers play a prominent role in solving large, sparse systems. Multigrid [1, 2] is effective for solving certain classes of problems, but is limited to problems discretized on logically rectangular grids. Algebraic multigrid (AMG) [1, 3, 4] is designed to solve problems on unstructured meshes, is effective for certain classes of problems, and various parallel packages that implement AMG are available [5, 6].

The effectiveness of the AMG solve phase relies on the setup phase algorithms, which are responsible for selecting coarse grids and building grid transfer operators. A number of parallel coarse-grid selection algorithms have been developed [7–10]. The focus of this paper is on the efficiency and scalability of the setup phase algorithms and the quality of their results.

*Correspondence to: David M. Alber, Siebel Center for Computer Science, University of Illinois at Urbana-Champaign, 201 North Goodwin Avenue, Urbana, IL 61801, U.S.A.
†E-mail: alber@uiuc.edu

The contributions in this paper are twofold. First, we introduce new colour-based coarse-grid selection algorithms. A parallel compatible relaxation algorithm is also introduced. The second purpose of this paper is to compare and analyse the behaviour of the growing number of parallel coarsening algorithms. In addition to reporting results from a number of experiments, we develop new tools to examine the behaviour of the coarsening algorithms beyond global measures like operator complexity alone. The detailed behaviour of the coarsening algorithms is enlightening and can be useful in identifying strengths and weaknesses of the methods.

The remainder of this paper is organized as follows. Section 2 contains a brief introduction of the general AMG algorithm. In Section 3, several coarse-grid selection algorithms are outlined. Numerical experiments and the results from those experiments are contained in Section 4. Finally, Section 5 contains conclusions.

2. ALGEBRAIC MULTIGRID

AMG [1, 3, 4] is an iterative method designed to numerically solve problems of the form

$$A_h u_h = f_h \tag{1}$$

where A_h is an $n \times n$ matrix. Unlike geometric multigrid, AMG does not operate on a physical grid, but instead relies only on the algebraic properties of the matrix problem. For conceptual convenience, the degrees of freedom, u_h^i, are called *grid points*, and the collection of nodes, $\Omega = \{u_h^1, u_h^2, \ldots, u_h^n\}$, are called a *grid*.

In a multigrid solver, the error, e_h, is composed of two components: high-frequency error and low-frequency (or smooth) error. In AMG, smooth error is any component of error not reduced by relaxation on the fine grid. In this case, smooth error is called *algebraically smooth error* (as opposed to geometrically smooth error). For convergence, a complementary process called *coarse-grid correction* is used to reduce smooth error. Coarse-grid correction is the process of approximately solving the defect equation $A_h e_h = r_h$, where $r_h = f_h - A_h u_h$ is the *residual*, on a coarse grid and interpolating a correction back to the fine grid.

An AMG user provides only the fine-grid matrix and right-hand side to the solver. However, AMG produces several additional components before attempting to solve the linear system. Below, superscripts are used to denote the grid level, where level 1 is the finest level. Therefore, $A^1 = A_h$ and $\Omega^1 = \Omega$. The components needed by AMG are all created in the *setup phase* and are defined as follows.

Grids: $\Omega^1 \supset \Omega^2 \supset \cdots \supset \Omega^M$

Grid operators: $A = \{A^1, A^2, \ldots, A^M\}$

Restriction operators: $R = \{R^1, R^2, \ldots, R^{M-1}\}$

Interpolation operators: $P = \{P^1, P^2, \ldots, P^{M-1}\}$

Smoothers: $S = \{S^1, S^2, \ldots, S^{M-1}\}$

Algorithm 1 shows the AMG setup phase and how each component is constructed. The 'stopping size' in Line 1 is defined by the designer of the AMG solver and is often as small as one. Line 2 encapsulates the coarse-grid selection step.

DOI: 10.1002/nla

Once the grids, grid operators, grid transfer operators, and smoothers have been constructed, the AMG algorithm is ready to begin solving the linear system. The actual solution is computed in the *solve phase* of AMG, which is given in Algorithm 2. The solve phase is similar to a geometric multigrid cycle, except the information generated in the setup phase (A, S, R, and P) is used instead of information that is explicitly known about the grids.

Algorithm 1. AMG SETUP(A_h).

Initialize:

$k \leftarrow 1$, $A^k \leftarrow A_h$

1. **while** $|\Omega^k| >$ stopping size **do**
2. Select C^k and F^k: $C^k \cup F^k = \Omega^k$ and $C^k \cap F^k = \emptyset$
3. $\Omega^{k+1} \leftarrow C^k$
4. Define P^k (interpolation operator from level $k+1$ to level k)
5. Define R^k (restriction operator from level k to level $k+1$, often $R^k = (P^k)^{\mathrm{T}}$)
6. $A^{k+1} \leftarrow R^k A^k P^k$ (Galerkin operator)
7. Construct smoother, S^k, if necessary
8. $k \leftarrow k+1$
9. **end while**
10. $M \leftarrow k$

Algorithm 2. AMG-V(A, S, R, P, u, f, k).

1. **if** $M = k$ **then** {k is the coarsest level}
2. Solve $A^k u = f$
3. **return** u
4. **else**
5. Apply smoother S^k to $A^k u = f$, μ_1 times
6. $r^k \leftarrow f - A^k u$
7. $r^{k+1} \leftarrow R^k r^k$
8. $e^{k+1} \leftarrow$ AMG-V(A, S, R, P, 0, r^{k+1}, $k+1$)
9. $u \leftarrow u + P^k e^{k+1}$
10. Apply smoother S^k to $A^k u = f$, μ_2 times
11. **return** u
12. **end if**

3. COARSE-GRID SELECTION

The setup phase of AMG accomplishes three goals:

1. Selection of a hierarchy of coarse grids. Recall a coarse grid is a collection of degrees of freedom, and the 'grid' is the graph of the coarse-level matrix.
2. Construction of transfer operators (restriction and prolongation operators, R and P, respectively) to transfer information from one level to another.
3. Construction of coarse-level operators (matrices). The coarse-level operator is typically the Galerkin operator: RAP.

DOI: 10.1002/nla

The most straightforward way to build a coarse grid is to select a subset of the degrees of freedom on the fine level to be the unknowns on the coarse level. The algorithms in this paper build the coarse grid in this way. However, other methods exist, such as *aggregation* [11, 12]. In aggregation, several unknowns from the fine level are combined into a single coarse level degree of freedom.

3.1. Classical coarsening

The first coarse-grid selection algorithm is called Ruge–Stüben (RS) coarsening [3]. A *strength of connection* measure is employed to determine if a given node has a strong influence on the solution at a neighbouring node. This information is used to decide whether a node should be marked a C-point or an F-point.

This strength of connection measure is based on the size of off-diagonal entries in the matrix. The set of nodes that node i strongly depends upon, denoted S_i, is defined as

$$S_i = \left\{ j : j \neq i, -a_{ij} \geqslant \theta \max_{k \neq i}(-a_{ik}) \right\} \tag{2}$$

where a_{ij} is the entry in row i, column j of matrix A. Often, θ is set to 0.25. The set of nodes that i strongly influences, denoted S_i^{T}, is defined as the set of nodes that strongly depend on i: $S_i^{\mathrm{T}} = \{j : i \in S_j\}$.

Two heuristics are used by RS to select a coarse grid:

H1: For each node j that strongly influences an F-point i, j is either a C-point or strongly depends on a C-point k that also strongly influences i.

H2: The set of C-points needs to form a maximal independent set in the reduced graph of the matrix such that no C-point strongly depends on another C-point.

Note that, in general, heuristics *H1* and *H2* cannot both be satisfied. *H1* is required by the RS interpolation scheme, so it must be satisfied. *H2*, on the other hand, is used only as a guideline. The purpose of *H2* is to encourage the selection of small, sparse coarse grids.

The RS algorithm proceeds in two sweeps: the first sweep establishes an independent set based on strength of connection, and the second sweep ensures that each pair of strongly connected F-points share a common C-point neighbour (*H1*). RS is outlined in Algorithm 3. In each iteration of the `while` loop in Line 6, a single C-point is selected. Therefore, without modification, this algorithm will not scale well in a parallel environment.

3.2. Parallel coarsening algorithms

The first parallel algorithms developed were variants of the RS algorithm. A straightforward modification is as follows. First, run RS on each processor's domain. This works well within each processor. Along the processor boundaries, however, there are regions where *H1* is not satisfied. The algorithm known as RS3 [7] addresses this problem by doing a third sweep along only the processor boundaries. Positions where *H1* is violated are corrected by assigning an extra node to the coarse grid.

Although RS3 produces a viable coarse grid (i.e. a coarse grid for which a RS prolongation operator can be constructed), this method typically has an abundance of C-points along the

DOI: 10.1002/nla

Algorithm 3. RUGE–STÜBEN.

Initialize:
$U = \Omega,\ C = \emptyset,\ F = \emptyset$
1. **for all** $i \in \Omega$ **do**
2. $w_i \leftarrow |S_i^{\mathrm{T}}|$
3. **end for**
4. **while** $|U| > 0$ **do** {First pass}
5. select i: $w_i \geqslant w_j,\ \forall j \in U$
6. $U \leftarrow U \backslash \{i\}$
7. $C \leftarrow C \cup \{i\}$
8. **for all** $j \in S_i^{\mathrm{T}} \cap U$ **do**
9. $U \leftarrow U \backslash \{j\}$
10. $F = F \cup \{j\}$
11. **for all** $k \in S_j \cap U$ **do**
12. $w_k \leftarrow w_k + 1$
13. **end for**
14. **end for**
15. **end while**
16. **for all** $i \in F$ **do** {Second pass}
17. **for all** $j \in S_i \cap S_i^{\mathrm{T}} \cap F$ **do**
18. **if** $S_i \cap S_j \cap C = = \emptyset$ **then**
19. make i or j into C-point
20. **end if**
21. **end for**
22. **end for**

processor boundaries. This is undesirable, however, because it demands more communication between processors.

This observation led to the development of additional parallel coarse-grid selection algorithms. A taxonomy of the coarse-grid selection algorithms discussed in this paper is shown in Figure 1. Other classes of coarse-grid selection algorithms, such as subdomain blocking [13] and compatible relaxation techniques [14–16], are not shown.

The remainder of this section is spent discussing and outlining the parallel coarse-grid selection algorithms used in the experiments later in the paper. Example output from these algorithms on a small problem is shown in Figure 2.

3.2.1. Cleary–Luby–Jones–Plassmann. The Cleary–Luby–Jones–Plassmann (CLJP) algorithm [8] utilizes a parallel independent set algorithm by Luby [17] to concurrently select several C-points in each iteration. This means unlike RS, CLJP can be run in parallel and will select C-points on each processor domain in each iteration. Algorithm 4 outlines CLJP coarsening.

Three lines of Algorithm 4 require further clarification: the initial value in Line 2, the selection of the independent set D in Line 5, and updating w_k in Line 9. CLJP implements these lines as follows.

DOI: 10.1002/nla

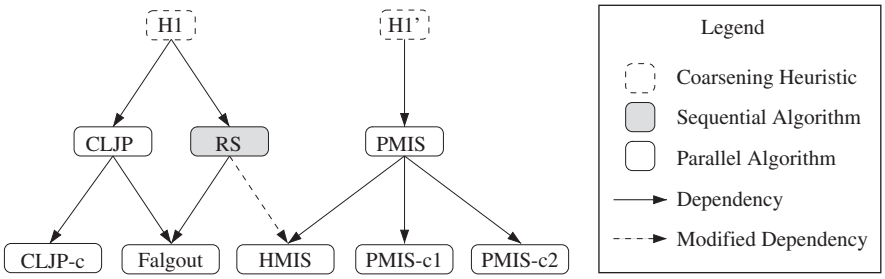

Figure 1. Taxonomy of coarsening algorithms. Algorithms with two incident lines are hybrid algorithms. For example, Falgout coarsening is a hybrid which uses RS and CLJP. The dashed line from RS to HMIS signifies that HMIS uses a modified version of RS, which is not subjected to *H1*.

Algorithm 4. CLJP.

Initialize:

$F = \emptyset$, $C = \emptyset$

1. **for all** $i \in \Omega$ **do**
2. $w_i \leftarrow$ initial value
3. **end for**
4. **while** $|C| + |F| \neq n$ **do**
5. select independent set D
6. **for all** $j \in D$ **do**
7. $C = C \cup j$
8. **for all** k in set local to j **do**
9. update w_k
10. **if** $w_k == 0$ **then**
11. $F = F \cup k$
12. **end if**
13. **end for**
14. **end for**
15. **end while**

The initial values of w_i are $|S_i^{\mathrm{T}}| + \mathrm{rand}(i)$, where $|S_i^{\mathrm{T}}|$ is the number of nodes strongly influenced by node i and $\mathrm{rand}(i)$ is a random number in $(0, 1)$. The purpose of the random number is to give each node a unique weight w. This random augmentation makes it possible to break ties between node weights, which allows C-points to be selected concurrently.

The independent set D is selected such that $D = \{i : w_i > w_j, \forall j \in S_i \cup S_i^{\mathrm{T}}\}$. This set will be independent, but is not necessarily maximally independent.

The values of w are updated using two heuristics:

1. Values at C-points are not interpolated; hence, neighbours that strongly influence a C-point are less valuable as potential C-points themselves.
2. If k and j both strongly depend on $i \in C$ and j strongly influences k, then j is less valuable as a potential C-point since k can be interpolated from i.

DOI: 10.1002/nla

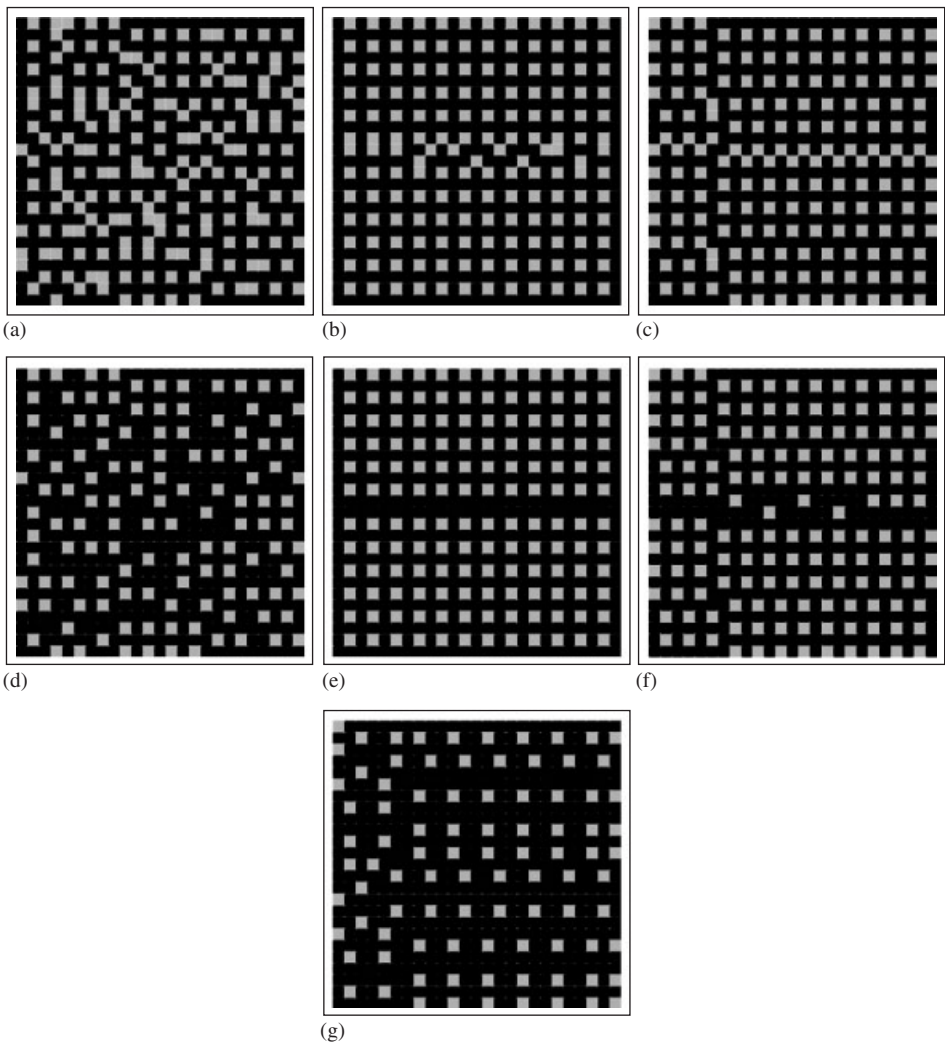

Figure 2. Coarse grids selected by various coarsening algorithms on a 25×25 9-point Laplacian problem. Two processors were used, and the processor boundary is in the vertical middle of the figures (the processor boundary is most obvious in the HMIS results). Filled black squares are F-points. Filled grey squares are C-points. The shifts seen in the results for the pre-coloured methods (CLJP-c, PMIS-c1, and PMIS-c2) are due to the initial selections made by the colouring algorithms. However, this shift becomes less apparent on larger problems. A slight change to the algorithm will lead to a more cosmetically appealing coarse grid: (a) CLJP; (b) Falgout; (c) CLJP-c; (d) PMIS; (e) HMIS; (f) PMIS-c1; and (g) PMIS-c2.

The weight update method depends on using the *auxiliary influence matrix*, which is defined as

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

DOI: 10.1002/nla

This means that $S_{ij} = 1$ if i strongly depends on j. Row i of S gives the strong dependencies of node i, and column i gives the strong influences of node i. These heuristics are implemented as shown in Algorithm 5.

Algorithm 5. CLJP-UPDATE-WEIGHTS.

```
1.  for all i ∈ D do
2.    for all j : S_ij ≠ 0 do {Heuristic 1}
3.      w_j ← w_j − 1
4.      S_ij ← 0
5.    end for
6.    for all j : S_ji ≠ 0 do {Heuristic 2}
7.      S_ji ← 0
8.      for all k : S_kj ≠ 0 do
9.        if S_ki ≠ 0 then
10.         w_j ← w_j − 1
11.         S_kj ← 0
12.       end if
13.     end for
14.   end for
15. end for
```

For some problems, such as problems on structured grids and problems in three dimensions, CLJP selects coarse-grid hierarchies with many more C-points than alternative algorithms, such as Falgout coarsening. Analysis of this behaviour is found in [10].

3.2.2. CLJP in colour. A modified form of CLJP, called CLJP in colour (CLJP-c), was first presented in [10]. CLJP-c modifies the initialization phase of CLJP to encourage the selection of a more structured coarse grid. Therefore, all changes for CLJP-c occur in Line 2 of Algorithm 4. In CLJP-c, a graph colouring algorithm is used to assign a colour such that each node i is painted a colour that is different than all nodes $j \in S_i \cup S_i^{\mathrm{T}}$. Then, these colours (along with other factors) are used in a function that assigns each node's initial weight. Algorithm 6 gives the details of how the initial weights are assigned in CLJP-c.

Algorithm 6. CLJP-C-INITIALIZATION.

```
1. Colour graph of S : σ_i ≠ σ_j ∀j ∈ S_i ∪ S_i^T {σ_i is colour of node i}
2. c_ℓ ← set of colours {colours are sequential integers beginning at 1}
3. for all c ∈ c_ℓ do
4.   c_w(c) ← (c − 1)/|c_ℓ|
5. end for
6. for all i ∈ V do
7.   w_i ← |S_i^T| + rand(0, 1/|c_ℓ|) + c_w(σ_i)
8. end for
```

The magnitude of a node's initial weight depends on three factors. The number of strong influences is the greatest factor in a node's initial weight. If two adjacent nodes have the same

DOI: 10.1002/nla

number of strong influences, the node colour will break the tie. Between two adjacent nodes, one is guaranteed to have the larger weight because two adjacent nodes will not be the same colour.

3.2.3. Parallel Modified Independent Set. The operator complexities for 3D problems generated by RS-like coarsening algorithms (CLJP, Falgout, and CLJP-c) do not scale well (see Section 4). To address this, the Parallel Modified Independent Set (PMIS) algorithm was developed [9].

PMIS produces coarse-grid hierarchies with lower operator complexities by using a relaxed form of *H1*. This new heuristic, called $H1'$, only requires each F-point to strongly depend on one C-point. Recall *H1* placed restrictions on strong F–F connections. By no longer requiring any shared C-points for that case, $H1'$ enables the selection of sparser coarse grids.

PMIS is similar to CLJP, with the exception that it implements $H1'$ (see Figure 1). Since RS prolongation operators cannot be constructed for coarse grids that do not satisfy *H1*, a modified form of the RS prolongator is also introduced in [9]. This modifies the operator so that strong F–F connections are handled when there is no shared strongly connected C-point. The PMIS algorithm is shown in Algorithm 7.

Algorithm 7. PMIS.

Initialize:

$F = \emptyset$, $C = \emptyset$

1. **for all** $i \in \Omega$ **do**
2. $w_i \leftarrow S_i^{\mathrm{T}} + \mathrm{rand}(0, 1)$
3. **end for**
4. **while** $|C| + |F| \neq n$ **do**
5. $D = \{i : w_i > w_j,\ \forall j \in S_i \cup S_i^{\mathrm{T}}\}$ {select independent set}
6. **for all** $j \in D$ **do**
7. $C = C \cup j$
8. **for all** $k \in S_j^{\mathrm{T}}$ **do**
9. $F = F \cup k$
10. **end for**
11. **end for**
12. **end while**

3.2.4. Hybrid Modified Independent Set. Also introduced in [9] is the Hybrid Modified Independent Set (HMIS) algorithm. If PMIS is the logical equivalent to CLJP with $H1'$, then HMIS is most similar to Falgout coarsening. Besides using $H1'$ instead of *H1*, HMIS employs only the first pass of RS on the interiors of processor domains. As with Falgout coarsening, the purpose of HMIS is to combine the best aspects of two algorithms, and on regular meshes this pays off in the form of better convergence factors in the solve phase (see Section 4).

3.2.5. PMIS-c1 and PMIS-c2. In [10], it was suggested that applying the graph colouring idea from CLJP-c to PMIS might yield positive results. Two new algorithms have come from this idea: PMIS-c1 and PMIS-c2.

PMIS-c1 is a direct application of the idea from [10]. That is, in PMIS-c1, the reduced graph of strong connections is coloured such that no two adjacent nodes are assigned the same colour. A colour priority is formed, and the weights in the initialization phase of PMIS are modified so

DOI: 10.1002/nla

that the colour priority is enforced. This method produces coarse grids similar to those produced by HMIS. The algorithm is generated by replacing Line 2 of Algorithm 7 with Algorithm 6.

PMIS-c2 developed from the observation that, when taken to the limit, PMIS can (and will, given appropriate weights) select a coarse grid such that 'neighbouring' C-points are three hops from one another in the graph. To produce a coarse grid with such sparsity using the graph colouring framework, a distance two colouring algorithm is used. Now each node is assigned a colour that is unique from the colours of all nodes within two hops in the graph. After substituting the graph colouring algorithm, PMIS-c2 proceeds in the same way as PMIS-c1. This algorithm is written the same as PMIS-c1, with the exception that Line 1 of Algorithm 6 should now read as follows:

1. Colour graph of $S : \sigma_i \neq \sigma_j$ and $\sigma_i \neq \sigma_k \ \forall j \in S_i \cup S_i^{\mathrm{T}}$ and $\forall k \in S_j \cup S_j^{\mathrm{T}}$ {σ_i is the colour of node i}.

3.3. Parallel compatible relaxation

In contrast to the coarsening algorithms discussed thus far, compatible relaxation (CR) [14–16] does not utilize a strength of connection measure. CR methods instead use the relaxation method to identify smooth error. This idea is intuitive since algebraically smooth error is defined as the error remaining when relaxation stalls. Algorithm 8 shows the CR algorithm from [16]. For our implementation $\nu = 5$.

Algorithm 8. COMPATIBLE RELAXATION.

Initialize:

$F = \Omega$, $C = \emptyset$,
$e^{(0)} = 1 + \mathrm{rand}(0,\ 0.25)$, $\alpha = 0.7$

1. **repeat**
2. Perform ν CR iterations on F, where $e_f^{(0)} = \mathbf{0} + (e^{(0)})_f$
3. $\rho_{\mathrm{cr}} = \frac{\|e_f^{(\nu)}\|_{A_{ff}}}{\|e_f^{(\nu-1)}\|_{A_{ff}}}$
4. **if** $\rho_{\mathrm{cr}} \geqslant \alpha$ **then**
5. Form candidate set

$$U = \left\{ i : \frac{|(e_f^{(\nu)})_i|}{\|e_f^{(\nu)}\|_\infty} \geqslant 1 - \rho_{\mathrm{cr}} \right\}$$

6. $D =$ Independent set of U
7. $C = C \cup D$
8. $F = F \backslash D$
9. **end if**
10. **until** $\rho_{\mathrm{cr}} < \alpha$

The term computed in each iteration (ρ_{cr}) is the CR rate, which is used to measure the quality of the tentative coarse grid. A small CR rate signifies the coarse grid is effectively representing the smooth error in the problem and that the coarsening process on that level can, therefore, be terminated.

DOI: 10.1002/nla

Theoretical results show that a coarse grid with a fast CR rate has an 'ideal' prolongation operator which when used with that coarse grid leads to a fast multigrid linear solver [18]. This valuable result motivates the use of the CR coarsening method.

Our contribution to CR is a parallel implementation of the CR algorithm. Most components of the algorithm already have parallel implementations. For instance, parallel relaxation methods are already used in parallel AMG codes. Our primary interest was in the independent set algorithms used to select C-points from the candidate set. Any of the algorithms discussed in Section 3.2 may be used as the independent set algorithm. For this paper, we used two parallel CR implementations: one with CLJP as the independent set algorithm, and another with PMIS as the independent set algorithm.

4. EXPERIMENTS

4.1. Methods

Each problem was tested using nine coarsening algorithms: Falgout, CLJP, CLJP-c, PMIS, HMIS, PMIS-c1, PMIS-c2, CR (CLJP independent set), and CR (PMIS independent set). Runs were made with a power of two number of processors ranging from 1 to 512. The strength threshold θ (see Equation (2)) is 0.25 for all coarsening algorithms in these experiments. Note changing θ may affect the complexities, but we expect the overall trends to remain unchanged.

Thunder, a large parallel machine at Lawrence Livermore National Laboratory, was used for all experiments. Thunder has 1002 quad-processor Itanium2 computing nodes, each with 8-GB RAM.

4.1.1. Problem generation and partitioning. Problems on regular grids were generated using routines in *hypre* [5]. Load balancing and problem partitioning on these types of problems is not problematic because they are handily partitioned into portions of equal size.

The generation of problems on unstructured grids was done using the aFEM package [19], which is a scalable, unstructured finite element problem generator. It uses ParMETIS [20] to partition the problem domain prior to discretization. To test the behaviour of AMG as problem size is scaled, the amount of work given to each processor should be comparable. Equal-sized partitions are not guaranteed for unstructured problems. For this reason, the amount of work given to each processor was monitored in each test. Where the partitioning significantly departs from what is desired, plots containing information on the partitioning of the problem are provided. Figure 12 contains a plot which reports the partitioning for one of the experiments. The plot contains several pieces of information. First, the solid black line shows the average nodes per processor. In an ideal situation, this line would be constant. Surrounding the line is two shaded fields. The light grey field shows the range of nodes per processor in the middle 90% of the distribution. The dark grey field shows the range of nodes per processor for all of the processors. Finally, the dashed line is drawn horizontally from the average nodes per processor on the single processor trial.

4.1.2. Grid and operator complexity. In AMG, complexities are used to measure the size of the coarse-grid hierarchy. Grid complexity is the number of unknowns (or nodes, in terms of the graph) on all levels relative to the fine level:

$$C_{\text{grid}} = \frac{\sum_{\ell=0}^{M} n_\ell}{n_0} \tag{3}$$

DOI: 10.1002/nla

where M is the number of levels in the grid hierarchy and n_ℓ is the number of rows in the matrix on level ℓ.

Operator complexity is the number of nonzeros in the matrices on all levels relative to the nonzeros in the fine level matrix:

$$C_{\text{op}} = \frac{\sum_{\ell=0}^{M} \text{nnz}_\ell}{\text{nnz}_0} \tag{4}$$

where nnz_ℓ is the number of nonzeros in the matrix on level ℓ. The operator complexity is a measure on the amount of memory needed, relative to the fine level, to store all of the matrices. It is also a lower bound on the computation needed since the work done for smoothing on all levels depends on the number of nonzeros in the matrices.

4.1.3. Convergence factors. Convergence factor results provide information about the overall quality of the solve phase. For the results in this paper, convergence factors are computed by averaging the convergence factors from all iterations until the norm of the relative residual is smaller than 10^{-8}. If a relative residual norm of 10^{-8} is not attained within 100 iterations, the convergence factors from the first 100 iterations are averaged.

4.1.4. Work per digit of accuracy. Neither the convergence factor nor the operator complexity results are, by themselves, a measure of the work a solve phase must do for convergence. By combining the convergence factor and cycle complexity, we arrive at a measure for the work needed per digit of accuracy. Work per digit of accuracy is defined as

$$W_{\text{digit}} = -\frac{C_{\text{cycle}}}{\log \rho} \tag{5}$$

where ρ is the convergence factor and cycle complexity is a measure of work in each multigrid cycle. The cycle complexity is related to the operator complexity and is defined as

$$C_{\text{cycle}} = \frac{\sum_{\ell=0}^{M} \text{nnz}_\ell \cdot v_\ell \cdot \gamma^\ell}{\text{nnz}_0} \tag{6}$$

where nnz_ℓ is the number of nonzeros in the matrix on level ℓ, v_ℓ is the sum of the pre- and post-smoothing steps on level ℓ, and γ is the cycle index. The cycle index is used in the definition of the multigrid cycle. For example, $\gamma = 1$ denotes the V-cycle and $\gamma = 2$ denotes the W-cycle. All experiments in this paper use a V(1, 1) cycle, which means the cycle complexity is usually close to double the operator complexity.

4.1.5. Tower plots. To visualize and examine the properties of the grid hierarchy in more detail, we introduce tower plots. The tower plot is used to visualize the entire coarse-grid hierarchy, separated by level. Figure 3 shows a typical tower plot.

Each tower plot contains four pieces of information. First, the height of the rectangle is that level's contribution to the operator complexity. For example, the height of level ℓ is $\text{nnz}_\ell/\text{nnz}_0$. The total height of the tower is the total operator complexity of the grid level hierarchy. Second, the width of each level corresponds to that level's contribution to the grid complexity (i.e. the number of degrees of freedom on that level relative to the fine level). This can be read in the plot by determining where the right edge of a level falls on the scale at the bottom. For example,

DOI: 10.1002/nla

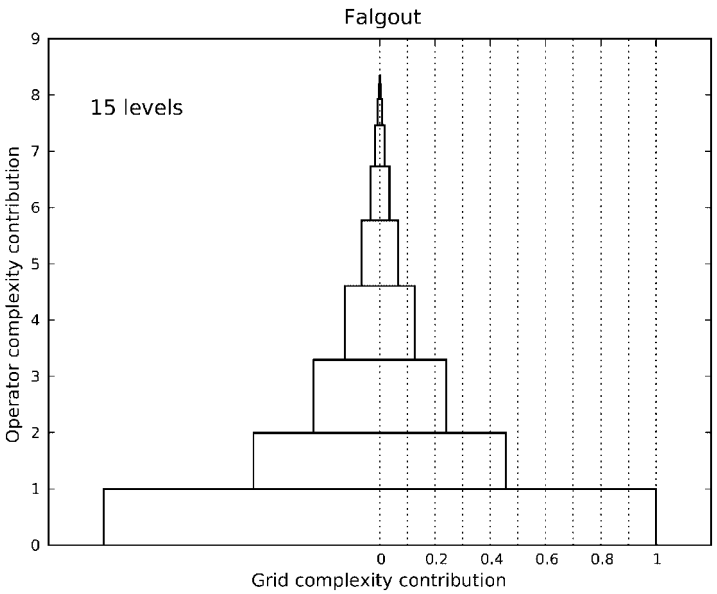

Figure 3. An example tower plot. The tower plot represents information about the coarse-grid hierarchy. Each level in the tower represents a level of the grid hierarchy, with the bottom level being the finest level. Four pieces of information are represented in each level of the tower: operator complexity, grid complexity, operator density, and the number of levels in the grid hierarchy.

the third level in the grid level hierarchy in Figure 3 contains approximately 24% of the number of degrees of freedom of the fine level. Third, the darkness of each block is determined by the sparsity of the matrix on that level. In most cases, the levels remain white until the very coarsest levels. The plot indicates the coarsening algorithm used to produce the plot in the title, and the total number of levels in the grid level hierarchy is shown.

4.2. Fixed problem sizes

The experiments are divided into two broad types of tests. The total size of the problem is fixed in the first set of tests, regardless of the number of processors used, to demonstrate the behaviour of the coarsening algorithms on the processor boundaries. By keeping the size of the problem fixed, the 'surface area' of each processor domain increases as the number of processors is increased. This is not a natural test for performance scalability—in a real-world simulation, the experimenter would give each processor as much work as could be processed in a reasonable amount of time. However, this type of test is useful for investigating the behaviour of the coarsening algorithms in parallel.

4.2.1. 3D 7-point Laplacian. The first problem is a 3D Laplacian:

$$\begin{aligned} -\Delta u &= 0 \quad \text{on } \Omega \ (\Omega = (0,1)^3) \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned} \tag{7}$$

DOI: 10.1002/nla

The problem is discretized using finite differences to yield the common 7-point stencil. The domain in all tests is a $128 \times 128 \times 128$ grid, which gives approximately two million unknowns.

Some coarsening algorithms are more affected by processor boundaries than others, so the degradation in the performance in some algorithms as the number of processors is increased is expected. The algorithms which are most sensitive to this are the hybrids (Falgout, HMIS) and the graph-colouring-based algorithms (CLJP-c, PMIS-c1, PMIS-c2).

Figure 4 plots setup times, convergence factors, operator complexities, and work per digit of accuracy for each of the trials in this experiment. The overall trend is for the setup time of each coarsening algorithm to decrease as the number of processors is increased. CLJP and CR (CLJP) both experience the greatest performance gains as the number of processors grows. On this problem, these algorithms both require large amounts of work to build the coarse-level hierarchy. The amount of work saved by splitting work across processors is much larger than the cost in communication. On the other hand, several coarsening algorithms initially experience an increase in setup time because on a single processor they are doing little work for this problem. The communication time spent on two processors is not offset by savings in computing time, so the total time increases. In the limit, however, all of the algorithms experience decreases in setup time.

There is a practical limit to gains made through parallelism due to communication across the processor boundaries. Figure 5 shows the normalized setup times for these trials. The normalized setup time is the setup time in a trial divided by the setup time in the single processor trial. At the right side of the plot, some lines are increasing, meaning the savings in computation are no longer larger than the extra cost in communication. Additionally, the setup phase requires the least amount of time on 128 processors, where the times are between 2% and 10% of the times on a single processor.

The preferred outcome for the convergence factors is invariance with the number of processors because parallelism does not improve the rate of convergence, but rather targets the computational cost in each iteration. In most cases, the convergence factor is constant across all trials; the largest exception to this is HMIS.

The increase in the convergence factors for HMIS is due to the large difference in the 'quality' of the coarse grid in the interior and the coarse grid on the processor boundary for HMIS. The interior part of each processor's piece of the mesh in HMIS is coarsened similarly as in RS, which performs at least as well as Falgout, in terms of convergence factor. However, the processor boundary coarse grid is selected using PMIS, and as the plot shows, the PMIS convergence factors are very large. As the number of processors is increased, the HMIS coarse grids have more nodes which are coarsened by PMIS, so the performance degrades.

Similar to convergence factor, it is desirable for the coarsening algorithms to have little impact on the operator complexity as the problem is divided among multiple processors. Both CLJP and PMIS can be made immune to this test since they have the ability to produce the same coarse-grid hierarchy independent of the number or processors on which the algorithm is run [8, 9]. The operator complexity analysed in Figure 4 shows each algorithm produces coarse-grid hierarchies of similar operator complexities on one processor versus hundreds of processors. The largest increase occurs with Falgout, which grows from approximately 5 to 6.5. In some cases, the operator complexity decreased by a small amount as the number of processors increased. Finally, as expected from previous observations [7, 9, 10], CLJP produces grid hierarchies with operator complexities so large the algorithm is not viable for this problem. CLJP produces unusually large operator complexities for problems on structured meshes, but this issue is not apparent with unstructured meshes.

DOI: 10.1002/nla

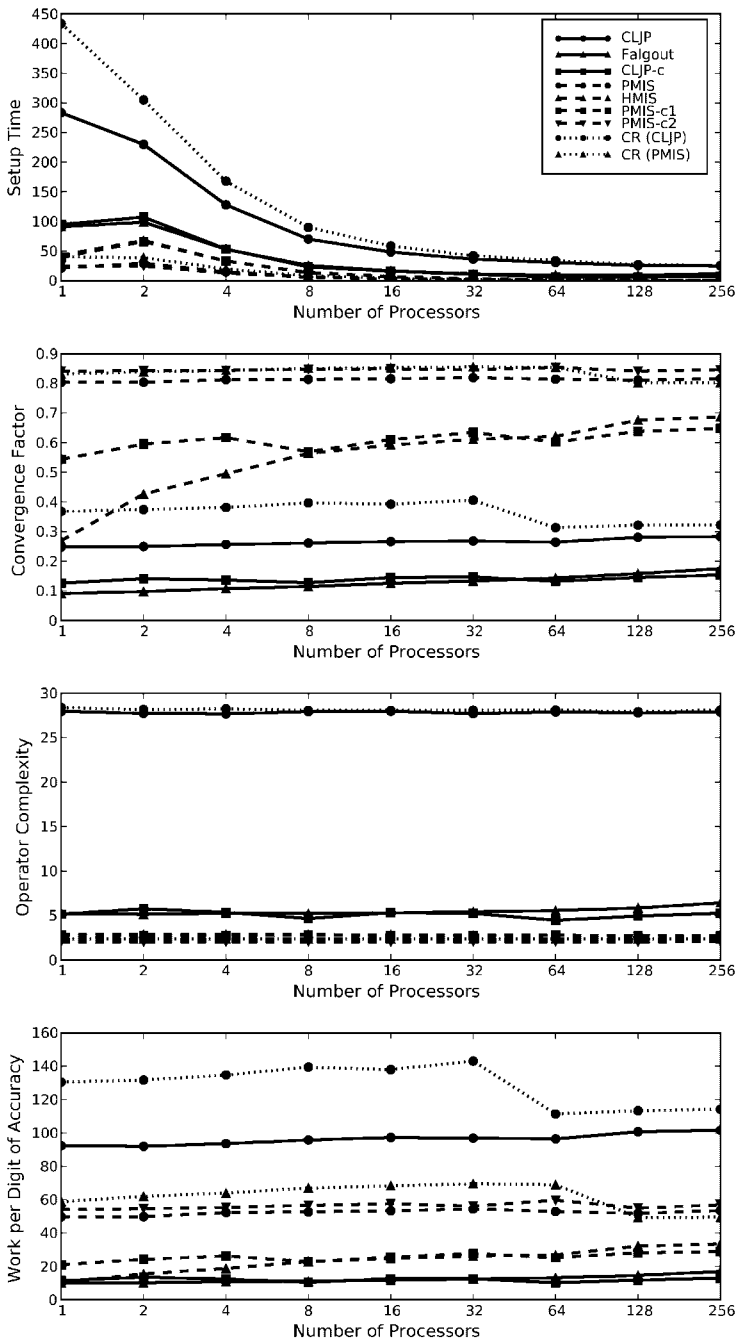

Figure 4. Results for the fixed problem size 3D 7-point Laplacian problem (Section 4.2.1). The total degrees of freedom in the problem is fixed while the number of processors increases. The legend from the first plot applies to all four plots.

DOI: 10.1002/nla

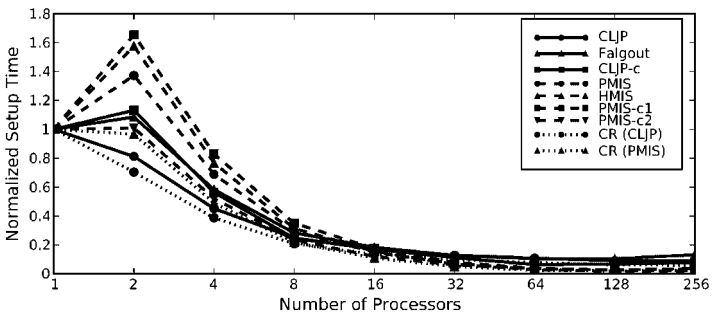

Figure 5. Normalized setup times for the fixed problem size 3D 7-point Laplacian problem (Section 4.2.1).

Recall work per digit of accuracy is a quantity depending on the cycle complexity and the convergence factor. As shown in Figure 4, Falgout and CLJP-c are the most cost-effective methods for this problem because they have the lowest convergence factors and also have operator complexities that are not particularly large, compared to the lowest operator complexities observed. Despite having a reasonable convergence factor, AMG with CLJP is more expensive than all other methods, except CR (CLJP), due to its extremely large operator complexity.

The operator complexities for CLJP and CR (CLJP) are very large. The other algorithms have much lower operator complexities. The tower plots for each coarsening algorithm run on 256 processors are shown in Figure 6. On 256 processors, CLJP selects coarse grids that produce matrices with a greater number of nonzeros on levels 1–9 than on level 0. Level 9 has more nonzero entries than level 0, despite having less than 5% the number of unknowns as level 0. The tower plots illustrate the similarity in complexities of the grid hierarchies selected by HMIS and PMIS-c1 and reveal that they appear nearly identical in terms of operator and grid complexity. This is further emphasized by the plots in Figure 4, which show the two methods are producing AMG solve phases with similar performance. On the other hand, there is little difference between the tower plots from HMIS and PMIS-c1 on the two processor trial (not shown), despite significant differences in performance.

4.2.2. 3D unstructured Laplacian. As before, this problem uses a fixed problem size and varies the number of processors used to solve the problem. The problem continues to be a 3D Laplacian on the unit cube (7), but is now discretized using finite elements on an unstructured mesh. The problem was run on up to 512 processors and has approximately 940 000 degrees of freedom. Figure 7 presents the setup time, convergence factor, operator complexity, and work per digit of accuracy results.

The setup times are relatively low compared to the previous problem. The CR (CLJP) algorithm yields the most time-consuming setup phase. The order of the algorithms by setup time is similar to the previous test, yet some differences are notable—e.g. CLJP is cheaper than both Falgout and CLJP-c in the problem, whereas in the last problem it was much more expensive. As before, the setup time as parallelism is increased reaches a minimum before beginning to increase after 128 processors. As Figure 8 shows, the algorithms reach 10% of their single processor cost when run on 128 processors, in the worst case. Finally, notice that the order of the algorithms by normalized setup time in Figure 8 is much different from the order in the structured case presented in Figure 5.

DOI: 10.1002/nla

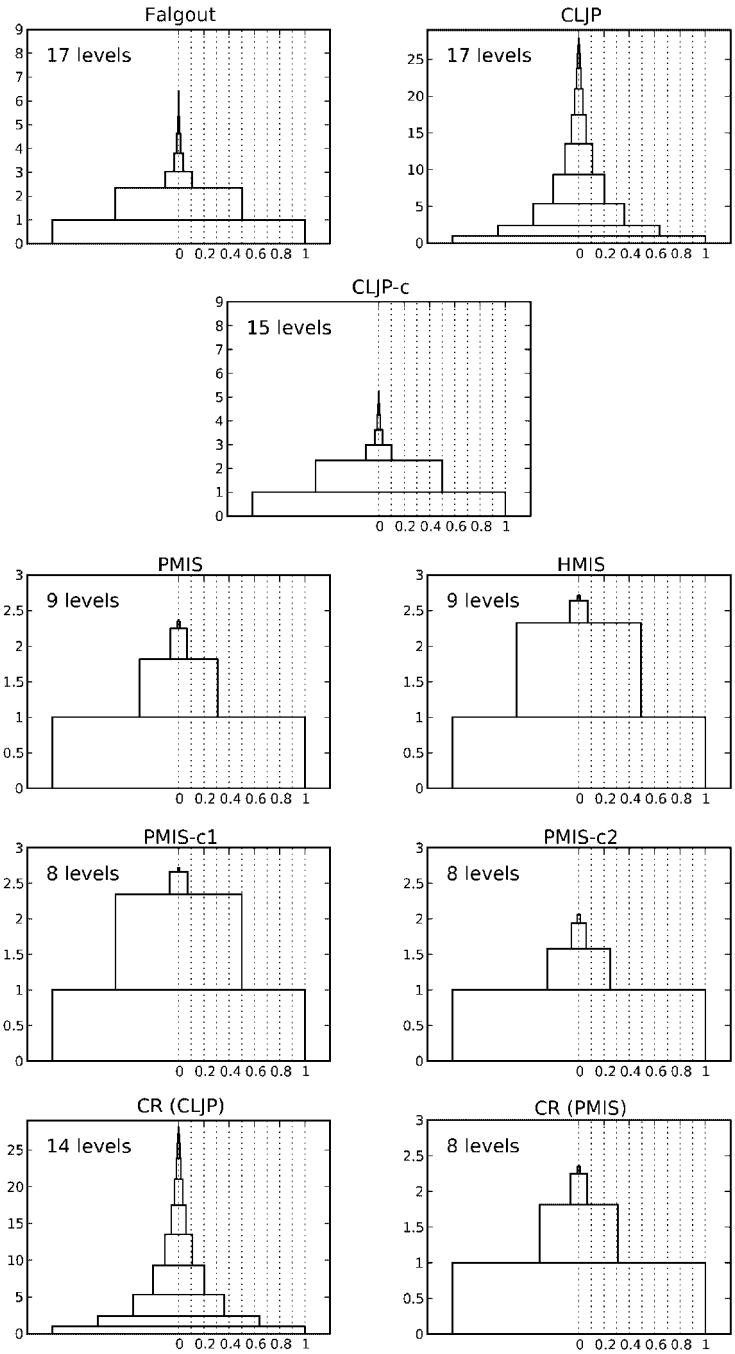

Figure 6. Tower plots for the fixed problem size 3D 7-point Laplacian problem (Section 4.2.1). The towers shown are for the 256 processor trials. Notice the scale is not the same in each plot.

DOI: 10.1002/nla

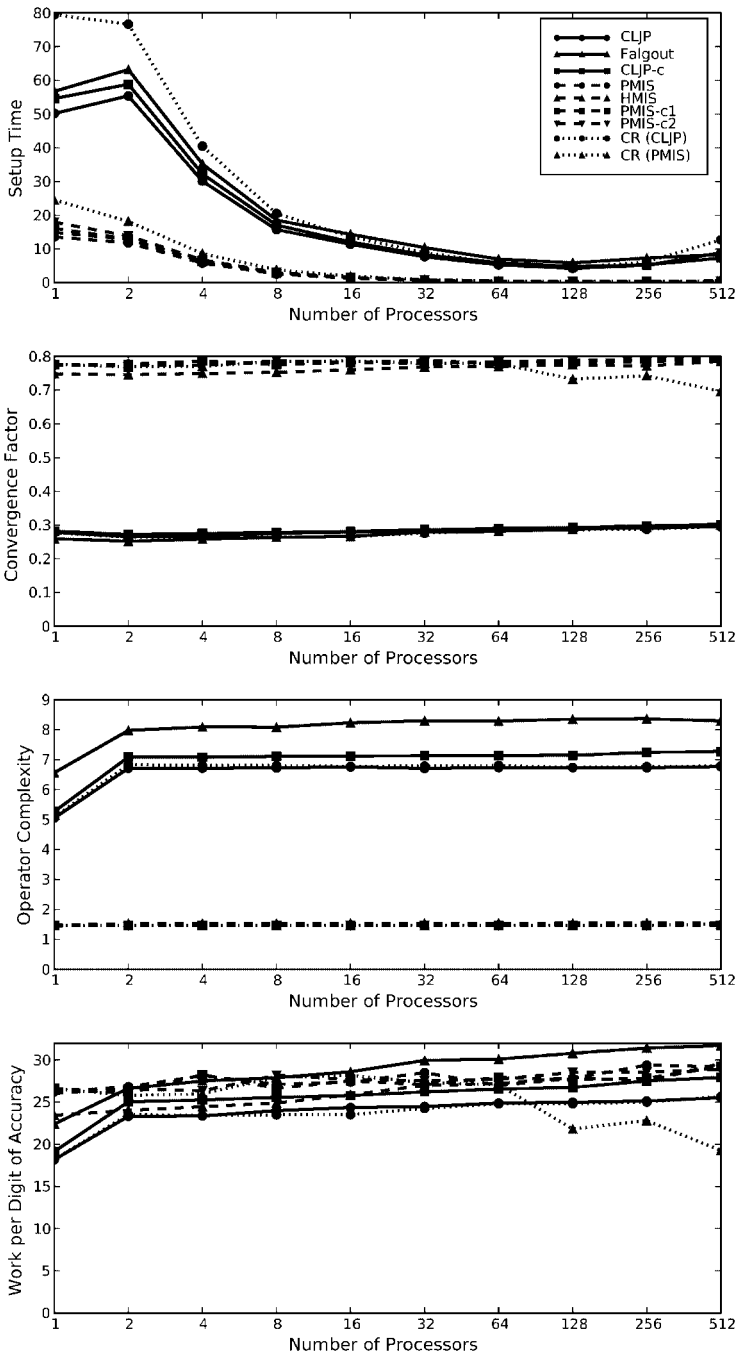

Figure 7. Results for the fixed problem size 3D unstructured Laplacian problem discretized on the unit cube (Section 4.2.2). The total degrees of freedom in the problem is fixed while the number of processors increases. The legend from the first plot applies to all four plots.

DOI: 10.1002/nla

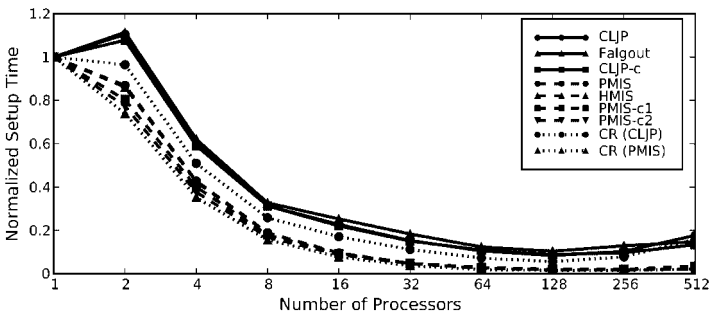

Figure 8. Normalized setup times for the fixed problem size 3D unstructured Laplacian problem discretized on the unit cube (Section 4.2.2).

The convergence factors and operator complexities exhibit little variance as the problem is partitioned into more pieces. The convergence factors exhibit little growth, with the exception of CR (PMIS), which decreases slightly on the largest number of processors. Initially, there is growth in operator complexity when moving from one processor to two. Subsequently, the operator complexities remain nearly constant. The operator complexity is much lower for CLJP in this experiment. Figure 9 contains the tower plots for this problem.

Much less work is needed per digit of accuracy in the unstructured test. The work per digit of accuracy is growing slightly, in most cases, but the growth is small considering the number of processors used in the largest test.

The two fixed problem size tests are designed to explore the parallel behaviour of the setup phase while using a variety of coarse-grid selection algorithms. The performance of AMG is fairly insensitive to the number of processors used. The operator complexities in AMG show little change regardless of the number of processors, even for the structured problem, which is highly impacted by coarse grids that do not maintain the structure of the fine grid. The operator complexities should begin to degrade if a sufficiently large number of processors are used, but at 512 processors, this test is already a departure from practical conditions.

4.3. Scaled problem sizes

The fixed problem size tests are designed to give insight into how the coarse-grid selection algorithms work as the parallelism is increased. However, tests of that design do not model the conditions under which AMG will typically be used. In particular, it is expected that any AMG user will aim to utilize as few processors as necessary to solve their problem.

The remainder of the experiments in this paper are on problems where the size of the problem is scaled to match the number of processors used. That is, the number of unknowns per processor is kept as close as possible to the number of unknowns on a single processor. We believe this allows the setup phase algorithms to be observed under more natural conditions.

4.3.1. 3D 7-point Laplacian. The structured problem (7) is now readdressed, except the problem size is scaled as the number of processors increases. On one processor, the problem is on a $50\times50\times50$ grid, for a total of 125 000 unknowns. On 256 processors, the problem is on a

DOI: 10.1002/nla

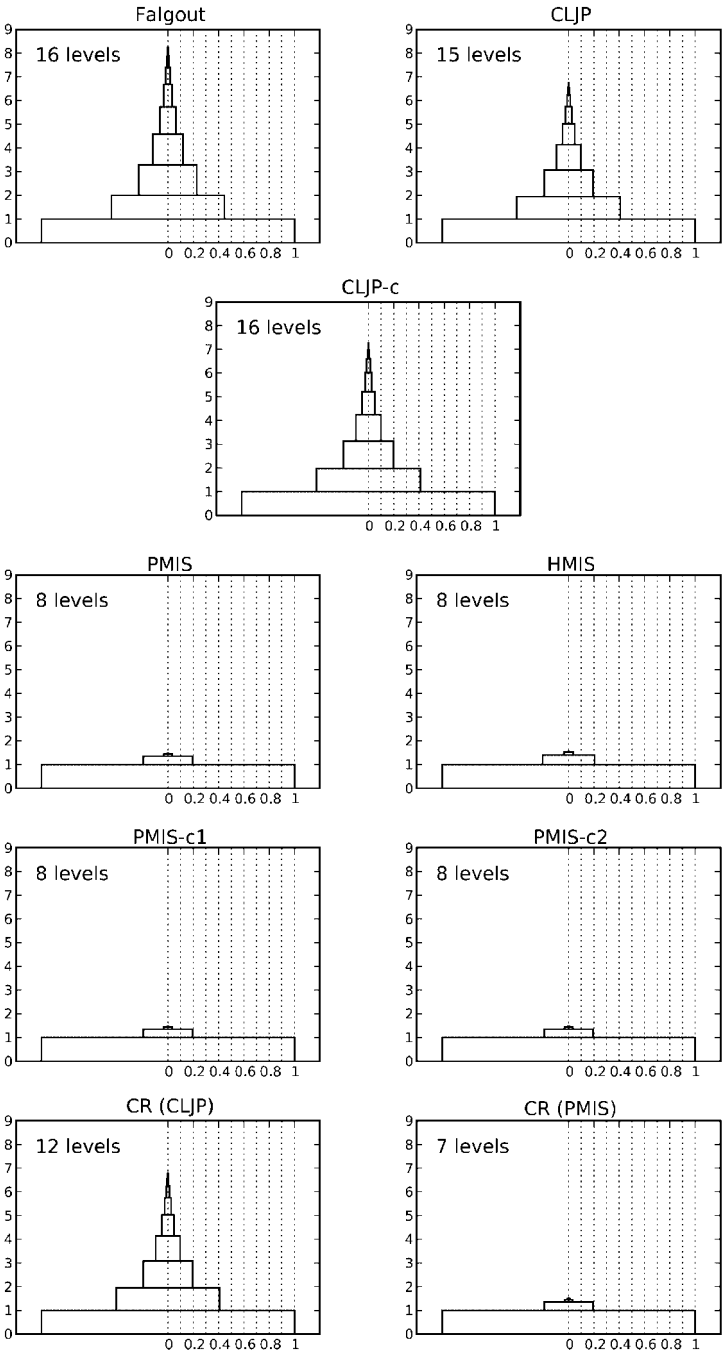

Figure 9. Tower plots for the fixed problem size 3D unstructured Laplacian problem on the unit cube (Section 4.2.2). The towers shown are for the 512 processor trials.

DOI: 10.1002/nla

$400 \times 400 \times 200$ grid, which makes 32 million unknowns. Such small problem sizes were necessary in order to have sufficient memory to run the algorithms which produce high operator complexities. The results for normalized setup time, convergence factor, operator complexity, and work per digit of accuracy are given in Figure 10. The plots reveal very different results than the plots of Section 4.2.1.

The figure illustrates that some algorithms are not performing near optimal in terms of setup time. In particular, CLJP, CLJP-c, and Falgout are each exhibiting large growths in their setup times. CLJP is growing much less than the other two, but its growth is still significant. In terms of actual time (not shown), CLJP is more expensive than CLJP-c or Falgout at the largest problem size, but if the trend continues, CLJP will require less time than both methods for a problem run on 1024 processors. This is interesting because the operator complexities of the grid hierarchies generated by CLJP are extremely large. This creates large numbers of edges in the coarse-level graphs, which requires large amounts of time for CLJP to update node weights. Neither CLJP-c nor Falgout are producing such large operator complexities, so the extra cost for these algorithms is not for the same reason. We believe some of this setup time cost is due to implementation and data structure issues, and this is something that will be investigated in the future.

The convergence factor results grow for all problem sizes and coarsening algorithms. At 512 processors, the convergence factors are growing at approximately the same rate as at two processors. Notice the PMIS-like algorithms (PMIS, HMIS, PMIS-c1, and PMIS-c2) are the slowest to converge. In the case of PMIS and PMIS-c2, the sparsity of the coarse grids selected and also the lack of preservation of the structure of the grid by PMIS lead to the slow convergence factors. Both methods produce coarse grids with good CR rates, which implies there is an interpolation operator that gives a fast multigrid method. Since slow results are observed, the prolongation operator currently used for these methods is inadequate to compensate for the sparse coarse grids. Our planned future work includes research on improved parallel prolongation methods. CLJP-c and Falgout yield the fastest convergence factors because these methods produce coarse grids that work well for structured problems such as this one.

The PMIS-like algorithms all produce grid hierarchies with much lower operator complexities than other methods, and the operator complexities display little or no growth as the problem size is increased. The performance of CLJP and CR (CLJP) is degraded for this problem because it is discretized on a logically rectangular grid. The growth of the operator complexities produced by these two methods is much larger than that of the other methods. The tower plots in Figure 11 illustrate the grid hierarchies for this problem.

The work per digit of accuracy will grow since all tests resulted in growing convergence factors. Despite producing relatively large operator complexities, CLJP-c and Falgout both create much cheaper AMG methods for this problem than any of the other methods. This is a result of their convergence factors being much lower than the PMIS-like methods and their operator complexities are much lower than CLJP.

4.3.2. 3D unstructured Laplacian. In this section, results are reported for the 3D unstructured Laplacian problem (7). The problem size on a single processor is approximately 211 000 unknowns. The largest problem is on 512 processors and has about 100 million unknowns, which gives an average of 198 000 unknowns per processor. The partition size data for this problem are shown in Figure 12. The partition sizes fluctuate, and these fluctuations are reflected in the results,

DOI: 10.1002/nla

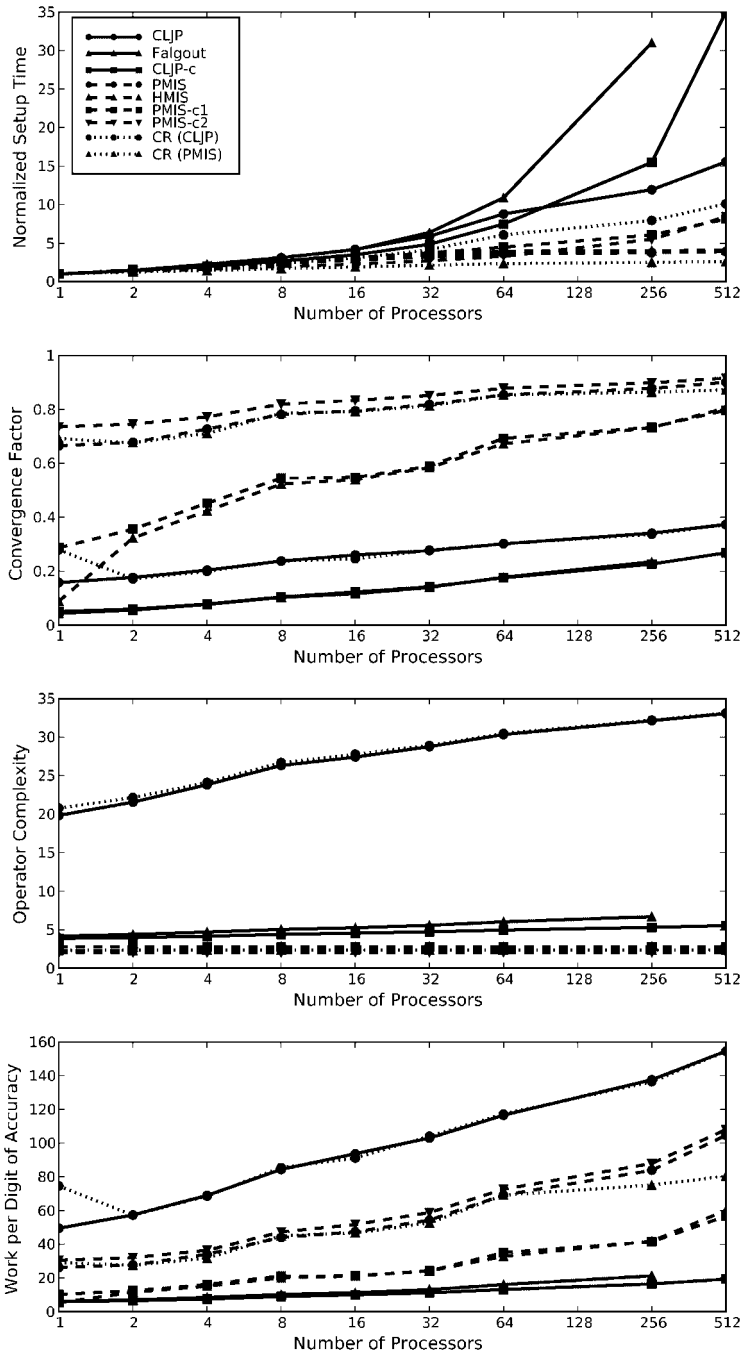

Figure 10. Results for the scaled 7-point Laplacian problem (Section 4.3.1). The legend from the first plot applies to all four plots.

DOI: 10.1002/nla

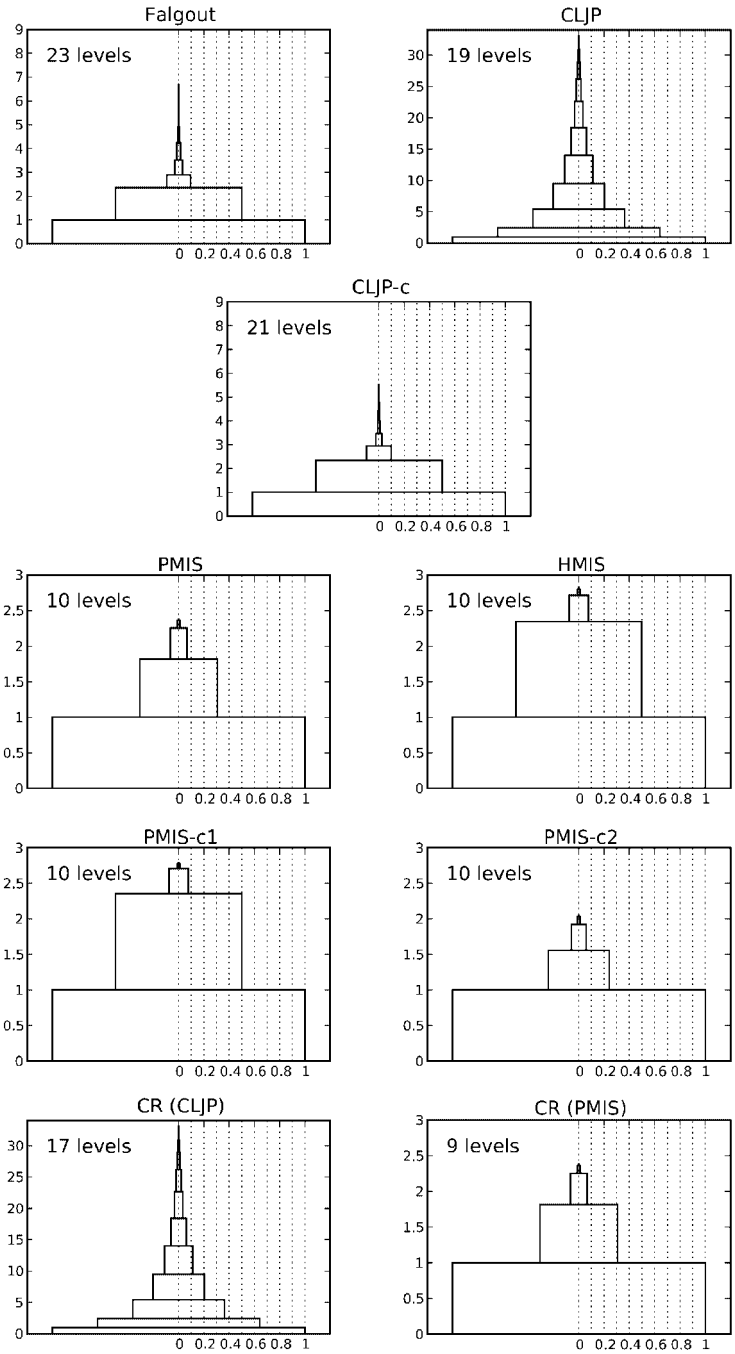

Figure 11. Tower plots for the scaled 7-point Laplacian problem (Section 4.3.1). The towers shown are for the 512 processor trials. Notice the scale is not the same in each plot.

DOI: 10.1002/nla

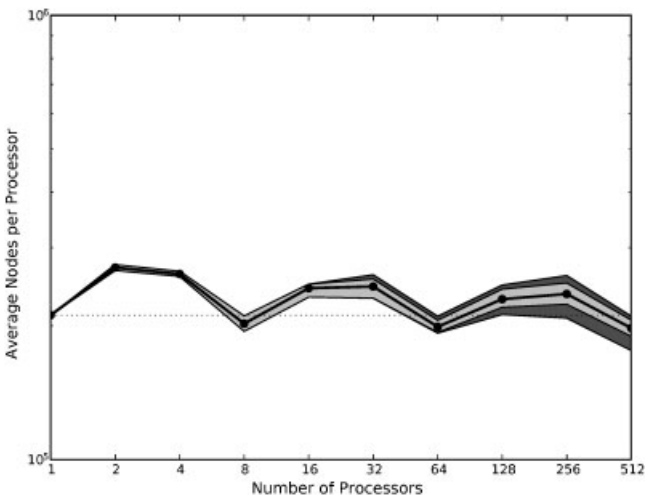

Figure 12. Partition size data for the scaled 3D unstructured Laplacian problem (Section 4.3.2) and the scaled 3D anisotropic problem (Section 4.3.3).

especially in the operator complexity plot. Normalized setup times, convergence factors, operator complexities, and work per digit of accuracy are reported in Figure 13.

As in the structured problem, the setup times for the RS-like algorithms are observed to be growing as the problem size grows. A 20 time increase in setup time from one processor to 512 processors is observed, and as before, the PMIS-like algorithm setup times are growing, but at a much slower rate than the RS-like algorithms.

The convergence factor plot looks similar to the convergence factors from the previous test with one major difference. Both CLJP and PMIS perform better on unstructured meshes than on structured meshes. In the previous problem, several groups of lines were present in the plot. Now two groups appear in the plot: one for the RS-like algorithms and one for the PMIS-like algorithms, meaning CLJP and PMIS both perform as well as algorithms related to them. Finally, notice the RS-like algorithms converged more slowly for this unstructured problem than the structured problem from before. In all cases, the convergence factors increased as the problem size increased.

The operator complexity results demonstrate the PMIS-like methods produce grid hierarchies with extremely low operator complexities which do not grow as the problem size grows. There is little variation in the operator complexities produced by each of those algorithms, and little variation is apparent in the tower plots in Figure 14. The RS-like algorithms, on the other hand, produce operator complexities that are both much larger and increase as the problem size grows. For this problem, Falgout coarsening produces operator complexities consistently and significantly larger than those produced by CLJP and CLJP-c.

The work per digit of accuracy results show CLJP to be the cheapest method available for this problem, once again highlighting the large difference in CLJP performance on structured versus unstructured grids. PMIS-c2 is the most expensive method, but is comparable to the other PMIS-like algorithms.

DOI: 10.1002/nla

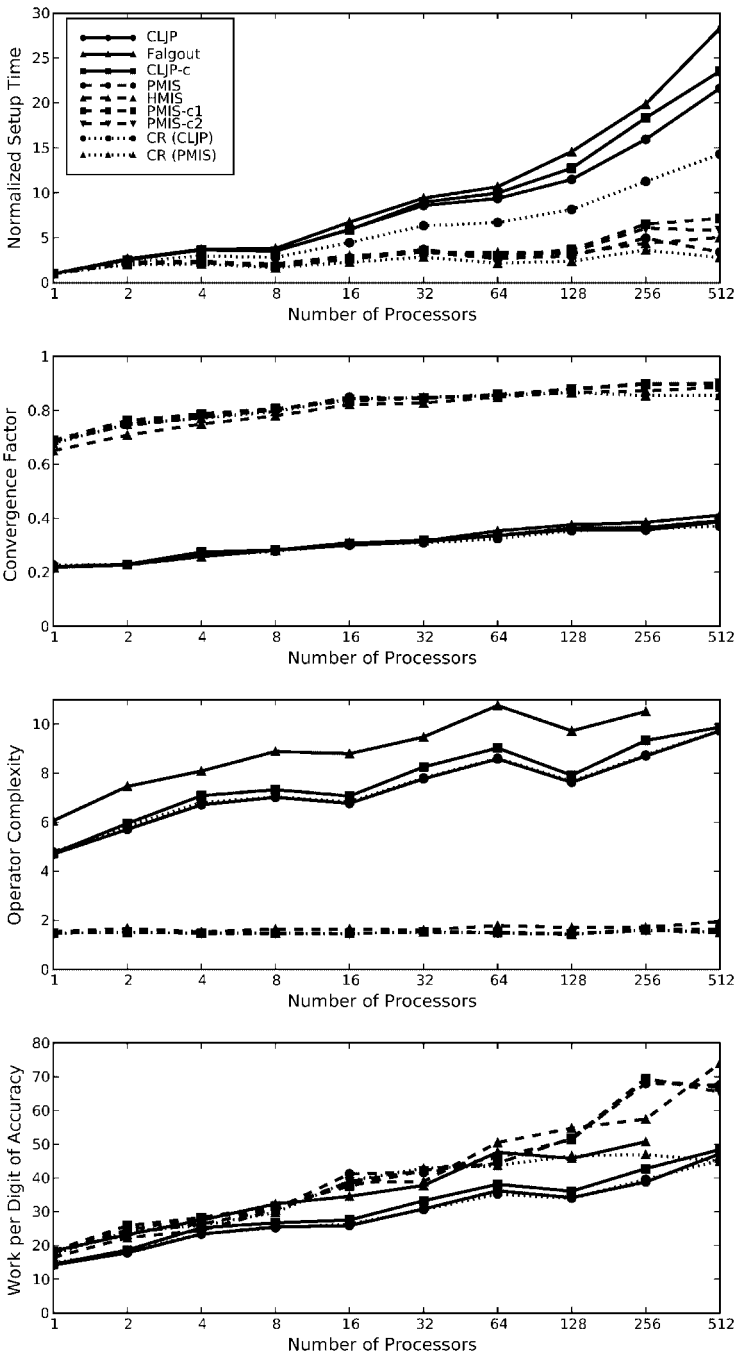

Figure 13. Results for the scaled 3D unstructured Laplacian problem (Section 4.3.2). The legend from the first plot applies to all four plots. The final data point for the Falgout line was removed from the final two plots because the operator complexity data were corrupted by overflow.

DOI: 10.1002/nla

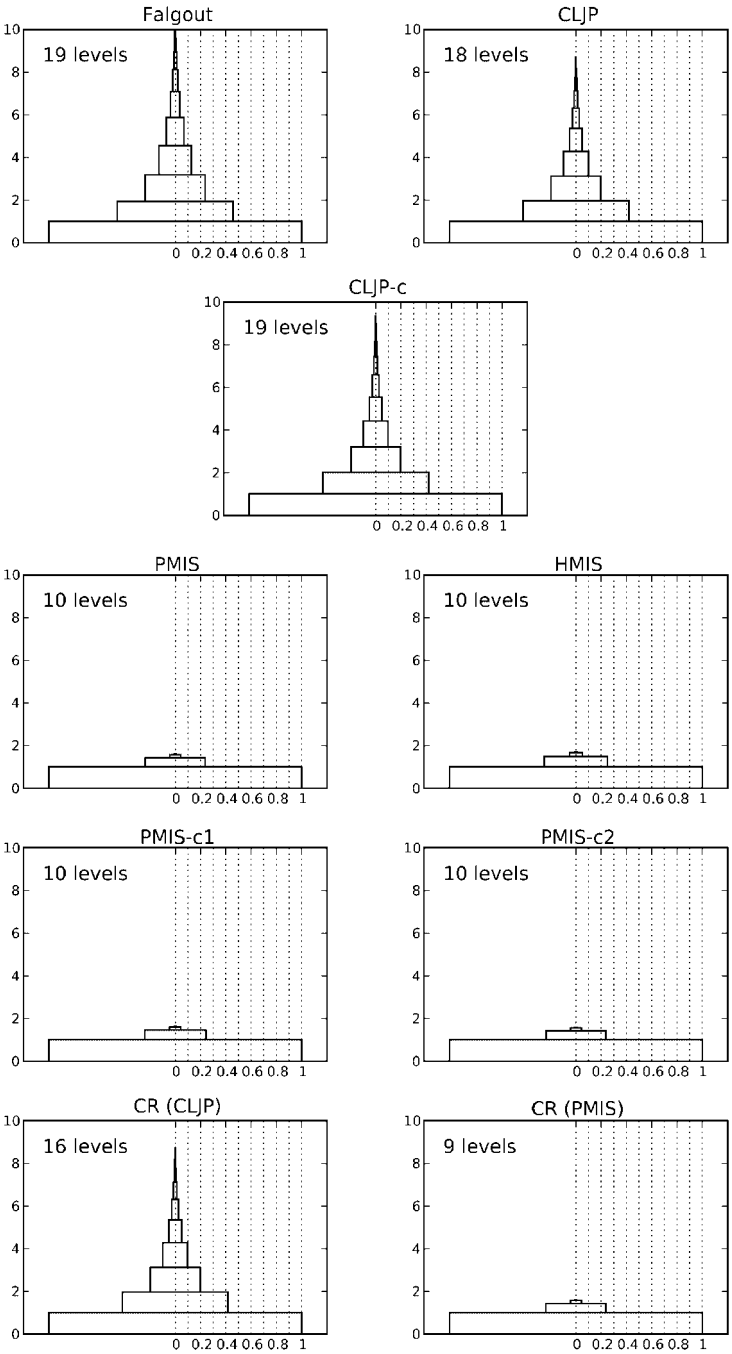

Figure 14. Tower plots for the scaled 3D unstructured Laplacian problem (Section 4.3.2). The towers shown are for the 256 processor trials.

DOI: 10.1002/nla

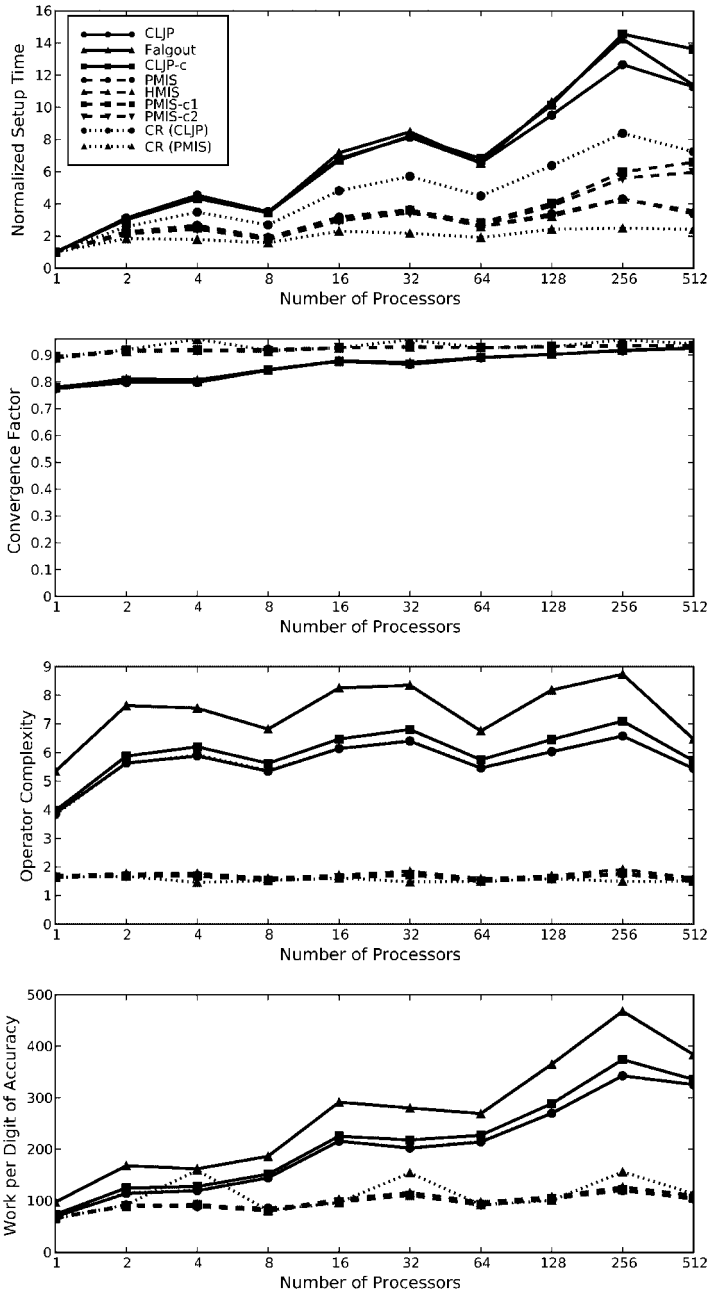

Figure 15. Results for the 3D unstructured anisotropic problem (Section 4.3.3). The legend from the first plot applies to all four plots.

DOI: 10.1002/nla

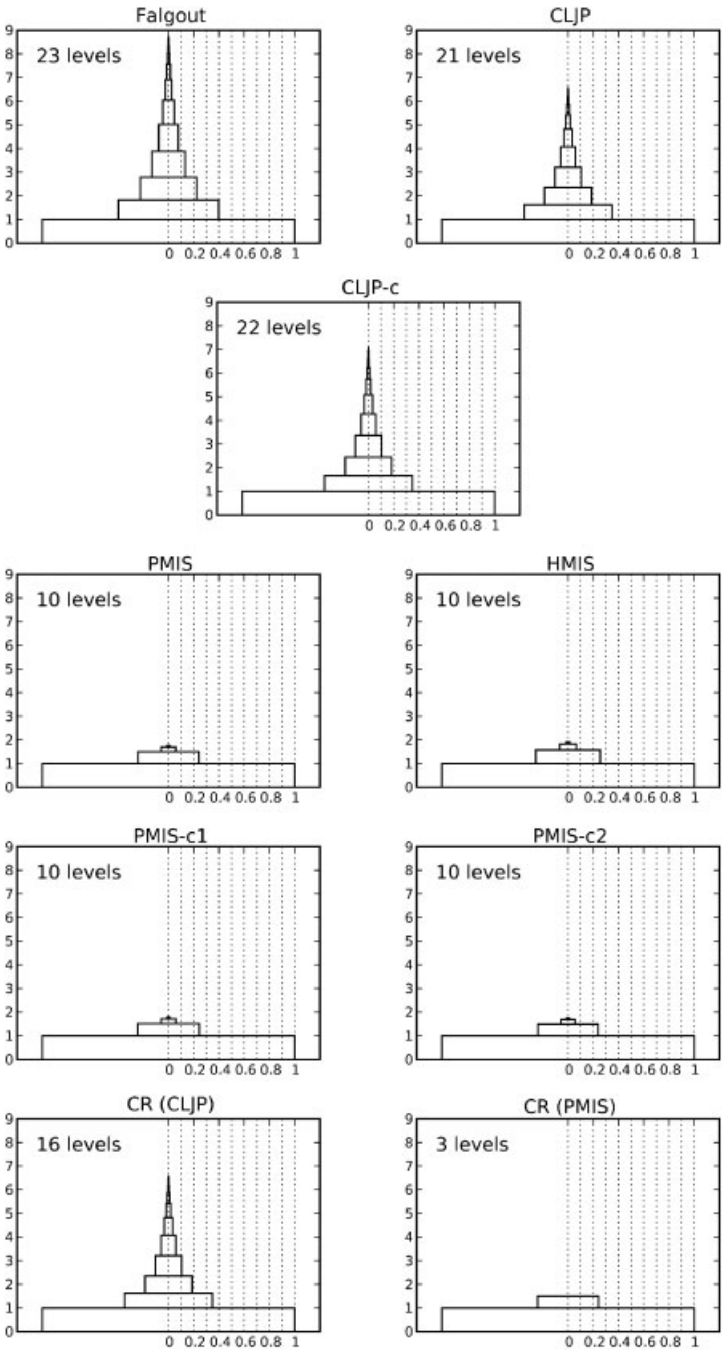

Figure 16. Tower plots for the 3D unstructured anisotropic problem (Section 4.3.3). The towers shown are for the 256 processor trials.

DOI: 10.1002/nla

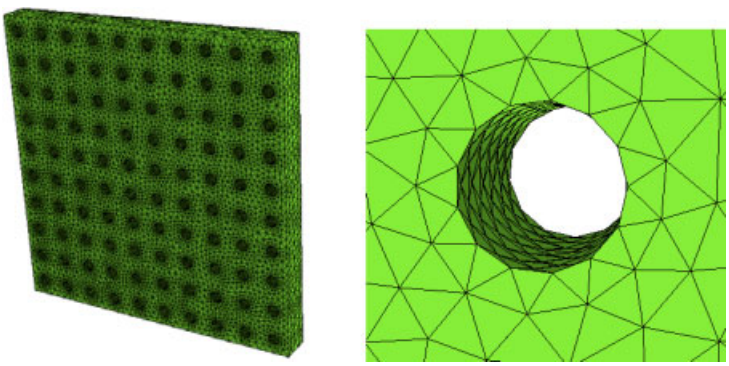

Figure 17. The problem domain for the 3D Laplacian holes test problem (Section 4.3.4). The right image is a close-up view of one of the holes.

4.3.3. 3D unstructured anisotropic problem. This test is on a 3D unstructured anisotropic problem defined as follows:

$$\begin{aligned} -(0.01u_{xx} + u_{yy} + 0.0001u_{zz}) = 0 \quad &\text{on } \Omega \ (\Omega = (0, 1)^3) \\ u = 0 \quad &\text{on } \partial\Omega \end{aligned} \tag{8}$$

The problem sizes are identical to the sizes in the 3D unstructured Laplacian. On one processor, the problem has approximately 211 000 unknowns. On 512 processors, the problem has about 100 million unknowns, which gives an average of 198 000 unknowns per processor. Figure 15 plots the observed normalized setup times, convergence factors, operator complexities, and work per digit of accuracy for this experiment. The effects of the non-uniform partitioning is most clear in this problem. Compare the pattern of growth in setup time in Figure 15 with the partition data in Figure 12. The fluctuations in work per processor appear to affect both the setup time and the operator complexity.

The normalized setup time results for this problem are similar to the results from the 3D unstructured Laplacian setup time data. The order of the algorithms by how quickly their cost grows is very close to the order from the previous problem. Though, the rate of growth is lower in this problem compared to the isotropic problem. The convergence factors in this problem are higher than in any other problems tested—in each case, the convergence factors approach one. The operator complexities for the anisotropic problem are similar, but slightly smaller than the complexities observed in the isotropic problem. The tower plots in Figure 16 show the complexities on each level in more detail. Finally, the work needed to get one more digit of accuracy in the residual is large compared with all other problems examined. This is due to the very slow convergence observed for this problem. The CR-based methods produce AMG solvers as slow to converge as any of the other algorithms. Recall theory states that these CR methods select coarse grids that are adequate to represent the algebraically smooth error. Furthermore, we know there is a prolongation operator which makes a multigrid method that converges quickly. From the slow convergence, we know the prolongation operator is not performing nearly as well as the theory-based optimal prolongation operator, which implies it is possible to construct a prolongation operator leading to a fast multigrid method on this problem.

DOI: 10.1002/nla

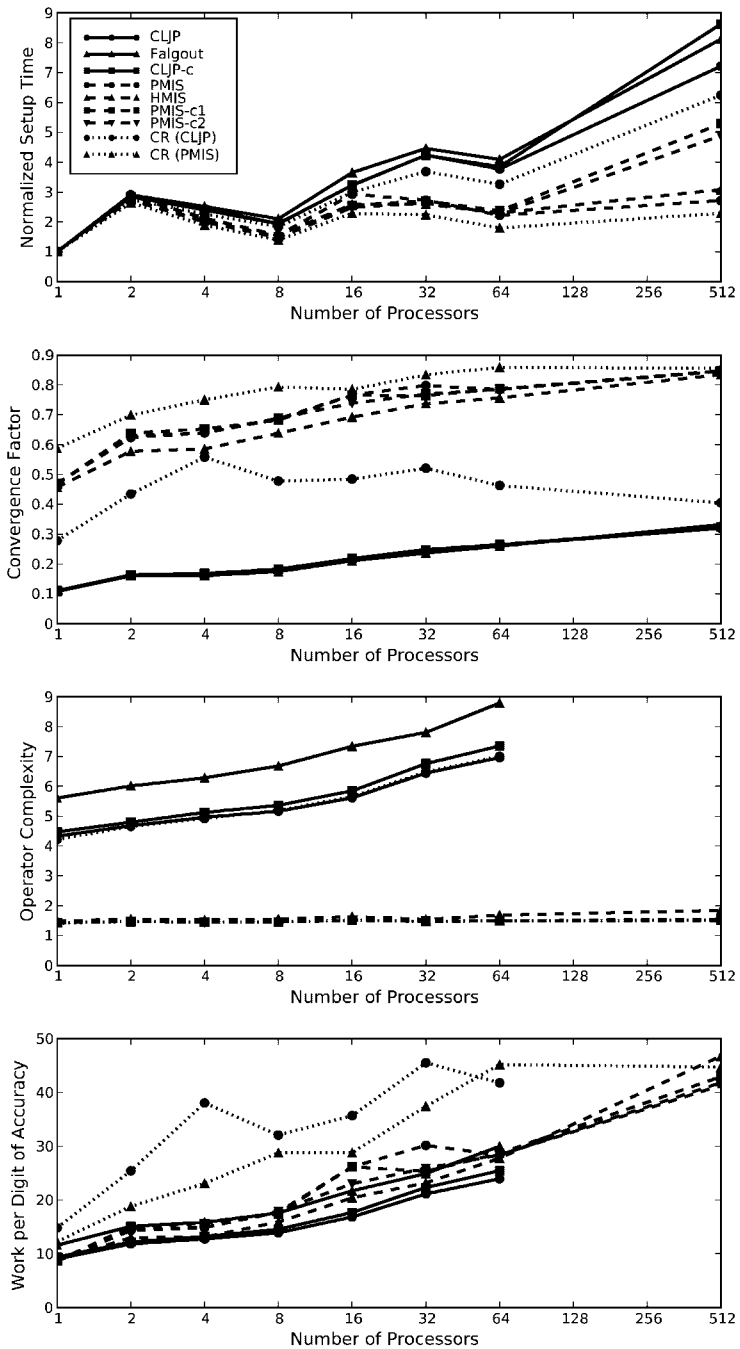

Figure 18. Results for the 3D unstructured Laplacian problem on the holes geometry (Section 4.3.4). The legend from the first plot applies to all four plots. The final data points have been removed from several of the lines on the operator complexity and work per digit of accuracy plots due to overflow errors in computing the operator complexity for those algorithms.

DOI: 10.1002/nla

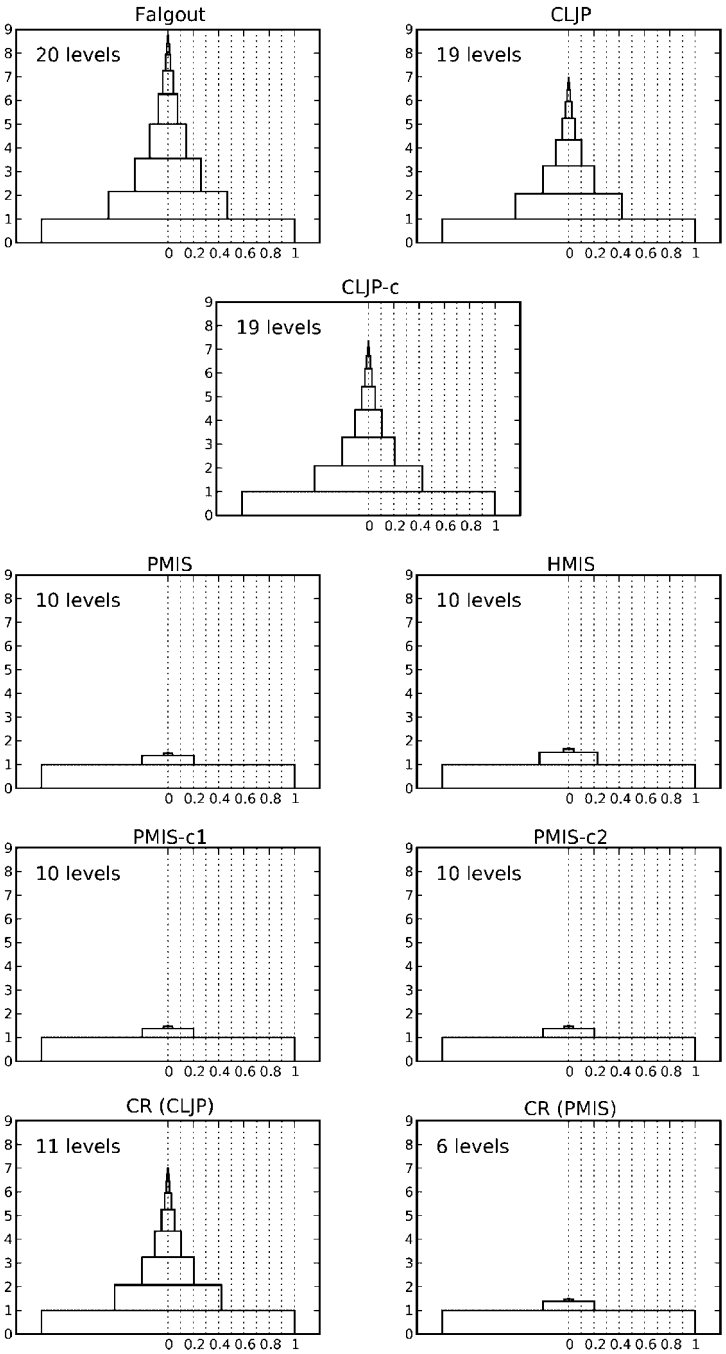

Figure 19. Tower plots for the 3D unstructured Laplacian problem on the holes geometry (Section 4.3.4). The towers shown are for the 64 processor trials.

DOI: 10.1002/nla

4.3.4. 3D Laplacian holes. The purpose of this experiment is to examine the effect, if any, on the performance of the coarsening algorithms on a problem with a more complicated geometry. In this instance, a thin slab with many holes drilled completely through the material is used as the problem geometry. With this problem, many more boundaries have been created. Figure 17 shows a mesh of the geometry used in this problem.

The problem solved on this domain is the Laplacian:

$$\begin{aligned} -\Delta u &= 0 \quad \text{on } \Omega \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned} \tag{9}$$

On one processor, the problem has approximately 380 000 unknowns. On 512 processors, the problem has about 167 million unknowns. This gives an average of 327 000 unknowns per processor. Figure 18 plots the normalized setup time, convergence factor, operator complexity, and work per digit of accuracy data from these tests. Below, we compare the results obtained to the data from the 3D unstructured Laplacian on the unit cube (Section 4.3.2). The normalized setup time results are similar for both tests, except the growth in time is much less in this problem than in the former case. The RS-like methods experience the greatest increase in setup times.

The convergence factors for this problem are initially lower for each coarsening algorithm than in the unstructured Laplacian problem. On the largest problems, the convergence factors are nearly the same as in the previous test. This is a result of the large increase of the interior nodes relative to the boundary nodes, which makes the two problems more similar for large numbers of unknowns.

The operator complexities in the two problems are very similar, as well. In this problem, the operator complexities are lower, but the rates of growth and the performance of the algorithms relative to one another are similar. The tower plots in Figure 19 show significant differences compared to the tower plots for the problem in Section 4.3.2.

With the complexities and convergence factors behaving similarly between the two problems, the work per digit of accuracy results are also similar. A clear difference is that this problem is cheaper to solve than the unstructured Laplacian on the unit cube, due to slightly lower convergence factors and lower operator complexities. Also, the CR methods both perform differently than before, compared with their respective groups.

Between this problem and the unstructured Laplacian on the unit cube, the most noticeable difference is that the problem with holes is cheaper to solve and has less growth in setup time. Overall, creating a larger 'surface area' makes the job of the coarsening algorithms cheaper to complete, but the general characteristics of the solver's performance do not change significantly.

5. CONCLUSIONS

In this paper, we introduced two new parallel coarse-grid selection algorithms: PMIS-c1 and PMIS-c2. The concept behind both algorithms developed from the pre-colouring technique used in the CLJP-c algorithm [10]. These algorithms join a growing set of parallel coarsening algorithms, many of which are tested in this paper. In addition to PMIS-c1 and PMIS-c2, two parallel compatible relaxation algorithms were introduced.

We introduced new coarse-grid hierarchy visualization tools, which help highlight differences between the algorithms. The tower plot (Figure 3) is used to see the level-by-level contribution to the grid and operator complexities.

DOI: 10.1002/nla

A series of experiments were used to examine the behaviour of the coarsening algorithms under different conditions. Run time, convergence factors, operator complexities, and work per digit of accuracy were reported for all algorithms in every test, revealing unique behaviours. In general, PMIS-like algorithms always produce grid hierarchies with lower operator complexities, and RS-like algorithms usually yield methods with faster convergence factors.

In some tests, such as the anisotropic diffusion problem (Section 4.3.3), AMG convergence was prohibitively slow. From recently developed theory [18], it is known that better prolongation operators exist for each of the problems where parallel CR was used. Since AMG was slow for some test problems, the restriction and prolongation operators must not be effectively transferring information from one level to the next. Recent work has been done on approximations to the ideal interpolation operator [16]. That work, however, has not focused on producing an effective and efficient parallel prolongation construction algorithm. This is a goal for our future work.

REFERENCES

1. Briggs WL, Henson VE, McCormick SF. *A Multigrid Tutorial* (2nd edn). SIAM: Philadelphia, PA, 2000.
2. Trottenberg U, Oosterlee C, Schüller A. *Multigrid*. Academic Press: London, 2001.
3. Ruge JW, Stüben K. Algebraic multigrid (AMG). In *Multigrid Methods*, McCormick SF (ed.), Frontiers in Applied Mathematics, vol. 3. SIAM: Philadelphia, PA, 1987; 73–130.
4. Stüben K. An introduction to algebraic multigrid. *Multigrid*. Academic Press: New York, 2000; 413–532.
5. Falgout RD, Yang UM. Hypre: a library of high performance preconditioners. *ICCS '02*: *Proceedings of the International Conference on Computational Science—Part III*. Springer: London, U.K., 2002; 632–641.
6. Tong C, Tuminaro RS. ML 2.0 smoothed aggregation user's guide. *Technical Report SAND2001-8028*, Sandia National Laboratories, December 2000.
7. Henson VE, Yang UM. BoomerAMG: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics* 2002; **41**:155–177.
8. Cleary AJ, Falgout RD, Henson VE, Jones JE. Coarse-grid selection for parallel algebraic multigrid. *Proceedings of the 5th International Symposium on Solving Irregularly Structured Problems in Parallel*. Springer: Berlin, 1998; 104–115.
9. Sterck HD, Yang UM, Heys JJ. Reducing complexity in parallel algebraic multigrid preconditioners. *SIAM Journal on Matrix Analysis and Applications* 2006; **27**:1019–1039.
10. Alber D. Modifying CLJP to select grid hierarchies with lower operator complexities and better performance. *Numerical Linear Algebra with Applications* 2006; **13**:87–104.
11. Ek PV, Mandel J, Brezina M. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing* 1996; **56**(3):179–196.
12. Brezina M, Falgout R, MacLachlan S, Manteuffel T, McCormick S, Ruge J. Adaptive smoothed aggregation (αSA) multigrid. *SIAM Review*: *SIGEST* 2005; **47**:317–346.
13. Krechel A, Stüben K. Parallel algebraic multigrid based on subdomain blocking. *Parallel Computing* 2001; **27**:1009–1031.
14. Brandt A. General highly accurate algebraic coarsening. *Electronic Transactions on Numerical Analysis* 2000; **10**:1–20.
15. Livne OE. Coarsening by compatible relaxation. *Numerical Linear Algebra with Applications* 2004; **11**:205–227.
16. Brannick J. Adaptive algebraic multigrid coarsening strategies. *Ph.D. Thesis*, University of Colorado, Boulder, 2005.
17. Luby M. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing* 1986; **15**(4):1036–1055.
18. Falgout RD, Vassilevski PS. On generalizing the AMG framework. *SIAM Journal on Numerical Analysis* 2004; **42**:1669–1693.
19. Kolev T. *aFEM Software Library*, 2006.
20. Karypis G, Schloegel K, Kumar V. *ParMETIS*: *Parallel Graph Partitioning and Sparse Matrix Ordering Library*, 2003.

DOI: 10.1002/nla