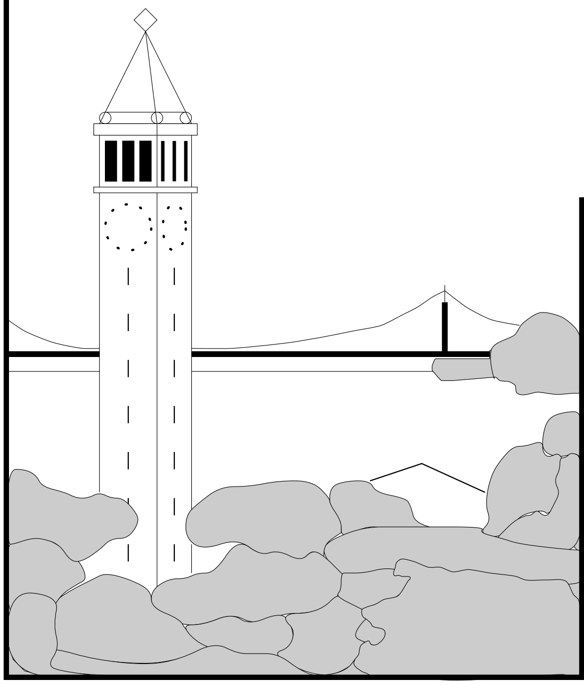
Evaluation of Three Unstructured Multigrid Methods on 3D Finite Element Problems in Solid Mechanics

Mark Adams [1]



Report No. UCB/CSD-0-1103

May 2000

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Abstract

Multigrid has been a popular solver method for finite element and finite difference problems with regular grids for over 20 years. The application of multigrid to unstructured problems is, however, not well understood and has been an active area of research in recent years. The two most promising categories of unstructured multigrid methods are 1) "geometric" methods that use standard finite element coarse grid function spaces, and 2) rigid body mode based coarse grid space "algebraic" methods. This paper evaluates the effectiveness of three promising multigrid methods (one geometric and two rigid body mode algebraic) on several unstructured problems in 3D elasticity with up to 76 million degrees of freedom.

Key words: unstructured multigrid, algebraic multigrid, parallel sparse solvers

1 Introduction

The availability of large high performance computers is providing scientists and engineers with the opportunity to simulate a variety of complex physical systems with ever more accuracy and thereby exploit the advantages of computer simulations over laboratory experiments. The finite element method is widely used for these simulations. The finite element method requires that one or several linearized systems of sparse unstructured algebraic equations (the stiffness matrix) be solved at each time step when implicit time integration is used. These linear system solves become the computational bottleneck (once the simulation has been setup and before the results are interpreted) as the scale of problems increases. Direct solution methods have been, and still are, popular since they are dependable; however the asymptotic complexity of direct methods, or any fixed level method, is high in comparison to optimal iterative methods (ie, multigrid).

Multigrid is well known to be an optimal solution method for Poisson's equation on structured meshes with a serial complexity of $O(n)$ (with n degrees of freedom) and $O(\log(n))$ or polylogarithmic complexity in parallel. The application of multigrid to unstructured meshes, that are the hallmark of the finite element method and the reason for its widespread use, is not well understood and is currently an active area of research. Multigrid theory (or domain decomposition theory [23]) can not provide hard bounds on the convergence rate of multilevel methods on unstructured problems, as it can for structured problems, but can provide expressions for bounds on the condition number of the preconditioned system in terms of coarse grid subspace quantities (eg, characteristic subdomain sizes H and characteristic discretization size h), as well as assumptions about the underlying operator and discretization. These bounds can be used to compare relative convergence rate of different methods so as to anticipate scalability problems for a particular method.

The expected complexity of using most methods can be estimated by combining the expected number of iterations (assumed proportional to a function of the bound on the condition number of the preconditioned system) with the complexity of each iteration in terms of the number of unknowns n, or characteristic discretization size h, and relevant quantities used in the condition number bound (eg, subdomain size H). Multigrid is optimal on some model problems as 1) the condition number of a system preconditioned with multigrid is bounded by a constant (ie, is not a function of H or h) and 2) the cost of each iteration (eg, number of floating point operations) is proportional to n. Thus, multigrid applied to some model problems has $O(n)$ serial complexity and polylogarithmic parallel complexity as some processors remain idle on the coarsest grids in the limit (see §1.1.1 for more details and references). Theory can provide the order of complexity for a multigrid method but is not able to prove the superiority of one multigrid method over another, or even differentiate between many multigrid methods. The goal of this paper is to evaluate three promising unstructured multigrid methods via experimental analysis with several 3D problems in solid mechanics with up to 76 million degrees of freedom. We also introduce a new aggregation method for the algebraic multigrid methods.

1.1 Multigrid methods

A multigrid method can best be defined by the method's coarse grid function space (see §2 or Smith [23]). There are two basic categories of functions spaces: geometric and algebraic. We define an *algebraic* method as a method that explicitly uses the values of a grid's stiffness matrix in the construction of the next coarse grid space. We define *geometric* multigrid methods as those that use an explicit coarse grid mesh to form standard finite element function spaces (thus, the coarse grid spaces have an explicit geometric representation). Note, non-nested coarse grid space methods are included in this definition of a geometric method.

We are aware of two classes of highly scalable linear solver methods for unstructured problems: geometric multigrid [2, 17, 15, 21] and algebraic multigrid based on rigid body modes [9, 25]. We wish to investigate methods that are not only theoretically optimal but that require only information that is readily available in most applications. To this end we do not investigate methods that require the user provide the coarse grid spaces or coarse grid operators [17, 15]; this leaves two geometric methods [2, 21] and the rigid body mode algebraic methods.

We investigate the algebraic method called "smoothed aggregation", by Vanek et. al., as it has superior theoretical convergence characteristics over plain aggregation methods [26]. We test both smoothed and plain aggregation in §6. Note, plain aggregation does not require the stiffness matrix (but it is useful) to construct the aggregates and the coarse grid spaces have an explicit geometric representation (ie, the rigid body modes). Thus, plain aggregation could be called a geometric method with our definition. We refer to plain aggregation as an algebraic method as we use the aggregation method of Vanek et.al., which does use the stiffness matrix to select the aggregates. Additionally, plain aggregation is closely related to smoothed aggregation, which is clearly an algebraic method (see §4).

We are aware of two classes of geometric methods for unstructured grids: hybrid and remeshing methods. The *hybrid* (or incremental, or annealing) geometric approach to coarsening grids is a promising method [21], as is the *remeshing* method [18, 10, 2]. Both methods automatically construct coarse grids from fine grids and use standard finite element shape functions to construct restriction operators for standard multigrid algorithms. We use the remeshing method as described in [2] as a representative geometric method in this investigation.

1.1.1 Complexity theory

This section provides a brief introduction and references to the results of convergence theory of our three unstructured multigrid methods, to complement and inform our numerical experiments. The most important result of the results of convergence analysis is that the geometric and smoothed aggregation have the potential of being optimal (ie, $O(1)$ or polylogarithmic $O(Poly(\log(n)))$ iterations to a fixed relative residual), and plain aggregation can only be shown to have polynomial complexity.

Truly optimal convergence is actually $O(1)$ iterations, or work per unknown, to attain accuracy to the discretization error of the problem. So called *full* multigrid (see §2), which we use in our experiments, attains this complexity on model problems [5, 4], but we will assume that all problems are being solved to a fixed reduction in the residual (eg, 10^{-6}). Note, we see some signs of this truly optimal convergence in some of our scalability studies in §6 with the geometric method.

Theory indicates that plain aggregation is not optimal in that the condition number of the preconditioned system is more than polylogarithmic in n - $O(\log(n)n^{\frac{1}{3}})$ for 3D problems [7]. Our experiments in §6 show some indication of super-polylogarithmic complexity, but are not conclusive without access to larger test problems and computers. Smoothed aggregation can provide us with a guarantee of $O(\log^3(n))$ iterations [26], and $O(\log(n))$ iterations with some assumptions about the problem [27]. Our experiments in §6 indicate $O(1)$ iterations on an elasticity problem with continuum elements with up to 76 million equations with full multigrid (note, full multigrid adds a $\log(n)$ term to the parallel complexity).

There is a body of unstructured geometric multigrid work that provides support for the optimal convergence rates that we observe in our numerical experiments. First, it is known that with some assumptions about the regularity of the problem that unstructured geometric multigrid, with nested quasi-uniform grids which articulate all material interfaces and boundary conditions, has optimal complexity (ie, $O(1)$ iterations for convergence, or convergence rates independent of problem size, with V-cycle multigrid [23]). Theoretically optimal convergence rates have been shown in 2D with some assumption about the problem and coarsening rates, and with multiple smoothing steps, on quasi-uniform meshes [30, 6]. Optimal convergence rates have also been proven for non-quasi-uniform grids with W-cycles and using a modified multigrid algorithm [31]. These results are not directly applicable to our test problems as they assume some properties that we can not guarantee in practice and use some modified multigrid algorithms, but they do provide some support for the optimal convergence rates that we observe.

2 Multigrid introduction

This section provides a brief introduction to multigrid [8], defining terms and comments on the structure of multigrid relevant to its implementation on high performance (ie, parallel) computers for unstructured problems. Multigrid has been an active area of research for almost 30 years and much literature can found on the subject. Multigrid is motivated by the observation that simple (and inexpensive) iterative methods like Gauss-Seidel, Jacobi, and block Jacobi, are effective at reducing the high frequency error effectively, but are ineffectual in reducing the low frequency content of the error [11]. These simple solvers are called *smoothers* as they render the error smooth by reducing the high frequency content of the error (actually they reduce high energy components of the error, leaving the low energy components which are smooth in, for example, Poisson's equation with constant material coefficients). The ineffectiveness of simple iterative methods can be ameliorated by projecting the solution onto a smaller space, that can resolve the low frequency content of the solution in exactly the same manner as the finite element method which projects the continuous solution onto a finite dimensional subspace to compute an approximation to the solution. Multigrid is practical because this projection can be prepared and computed reasonably cheaply and has $O(n)$ complexity. The coarse grid correction (the solution projected onto a coarser grid) does not eliminate the low frequency error exactly, but it "deflates" the low frequency error to high frequency error by removing an approximation to the low frequency components from the error.

Multigrid requires three types of operators: 1) *restriction* and *prolongation* operators, which can be implemented with a rectangular matrix (R and $P = R^T$ respectively); 2) the PDE operator, a sparse matrix, for each coarse grid (the fine grid matrix is provided by the finite element application); and 3) cheap (one level) iterative solvers that can effectively eliminate high frequency error in the problem at hand. The coarse grid matrix can be formed in one of two ways - either algebraically to form Galerkin (or variational) coarse grids ($A_{coarse} \leftarrow RA_{fine}P$), or by creating a new finite element problem on each coarse grid (if an explicit coarse grid is available), thereby letting the finite element implementation construct the matrix. Only algebraic coarse grids are considered as we wish to maintain a narrow interface with the finite element application so as to place a minimal burden on the application. Additionally we believe algebraic coarse grids are inherently more robust as, for instance, Jacobians need not be formed with the coarse grid elements.

Figure 1 shows the standard multigrid *V-cycle* and uses a smoother $x \leftarrow S(A, b)$, and restriction operator R_{i+1} that maps residuals from the fine grid space i to the coarse grid space $i + 1$ (the rows of R_{i+1} are the discrete representation of the coarse grid function space of grid $i + 1$).

function $MGV(A_i, r_i)$
  if there is a coarser grid $i + 1$
    $x_i \leftarrow S(A_i, r_i)$
    $r_i \leftarrow r_i - Ax_i$
    $r_{i+1} \leftarrow R_{i+1}(r_i)$ - - restriction of residual to coarse grid
    $x_{i+1} \leftarrow MGV(R_{i+1}A_iR_{i+1}^T, r_{i+1})$ - - the recursive application of multigrid
    $x_i \leftarrow x_i + R_{i+1}^T(x_{i+1})$ - - prolongation of coarse grid correction
    $r_i \leftarrow r_i - A_ix_i$
    $x_i \leftarrow x_i + S(A_i, r_i)$
  else
    $x_i \leftarrow A_i^{-1}r_i$ - - direct solve of coarsest grid
  return x_i

Figure 1: Multigrid *V-cycle* Algorithm

Many multigrid algorithms have been developed; the "full" multigrid algorithm is used in the numerical experiments. One full multigrid cycle applies the V-cycle to each grid, by first restricting the residual (b) to the coarsest grid and applying a V-cycle (simply a direct solve), interpolating the new solution to the next finer grid as an initial guess, computing the residual, applying the V-cycle to this finer grid, adding the coarse grid correction to the initial guess, and so on until the finest grid is reached. One iteration of full multigrid is used as a preconditioner for the Krylov method "accelerator".

An important, and advantageous, characteristic of multigrid is that the solver has several distinct parts: the restriction/prolongation operators, the smoother, the Krylov method accelerator (the actual solver), multigrid algorithms such as V-cycles, F-cycles, W-cycles, and other standard multigrid infrastructure (ie, sparse matrix triple products for algebraic coarse grids). The only difference between multigrid methods is the construction of the coarse grid spaces or restriction/prolongation operators - the rest of the infrastructure for multigrid is standard and can be reused by multiple multigrid methods. This is useful as a well designed multigrid implementation can facilitate the use of multiple multigrid methods easily.

The reason for using multigrid is to insure that the convergence rate is independent of the scale of the problem and the cost, in floating point operations (flops), asymptotes to a constant as the scale of the problem increases. The smoother can have an important impact on this cost, especially on challenging problems, and is the primary parameter in optimizing the solve time for a particular problem by reducing the constant in the complexity (ie, the cost per iteration divided by the convergence rate). This structure has the advantage that the job of the solver is decoupled into two distinct parts: 1) the multigrid method and 2) one of the many smoothers that have been developed as one level iterative solvers (eg, conjugate gradients preconditioned with: block Jacobi, overlapping Schwarz, incomplete factorizations, approximate inverses, etc.). This allows two aspects of the solver (ie, the scale and solving for the operator at hand) to be attacked independently and allows for new multigrid algorithms to added easily to existing multigrid codes as only the construction of the restriction operator need be replaced.

3 Remeshing geometric multigrid

The geometric multigrid method was first proposed by Guillard [18] and independently by Chan and Smith [10], and latter extended to 3D and in parallel by Adams [2]. The purpose of this algorithm is to automatically construct an explicit coarse grid from a finer grid for use with standard finite element shape function spaces. This method is applied recursively to produce a series of coarse grids, and their attendant operators, from a "fine" (application provided) grid. A high level view of the algorithm is as follows:

1. The node set at the current level (the "fine" mesh) is uniformly coarsened, using an maximal independent set (MIS) algorithm to produce a subset of vertices.
2. The new node set is then automatically remeshed with tetrahedra with a standard Delaunay algorithm [16].
3. Standard linear finite element shape functions are used to produce the restriction operator R.
4. The restriction operator is then used to construct the (algebraic) coarse grid operator from the fine grid operator: $A_{coarse} \leftarrow RA_{fine}R^T$.

These operators are then used in a standard multigrid algorithm.

We employ a series of optimizations to the MIS algorithm to preserve the geometric representation of the fine mesh as described in [2]. The crux of these optimizations is to automatically define faces of the mesh and use these faces to modify the graph and order the nodes in standard greedy MIS algorithms. These faces are used to implicitly improve the representation of the original geometry by 1) modifying the graph for the MIS algorithm to avoid two separate features from interacting and hence potentially decimating each other (eg, the two sides of a thin body meshed with continuum elements) and 2) by emphasizing geometrically "important" nodes (ie, corners and edges) by reordering the nodes in the standard greedy MIS algorithm [1]. The algebraic methods also modify the matrix graph for use in an MIS algorithm to aggregate (or partition) the fine grid vertices.

4 Smoothed and plain aggregation algebraic multigrid

Unlike geometric methods, algebraic methods do not construct explicit coarse grid meshes (ie, tetrahedra and coordinates). The main advantage of algebraic methods is that they need not construct coarse grid finite element meshes but only need to work with graphs. Working with graphs is much easier than working with meshes - especially in parallel - as mesh coarsening and mesh generation are much more complex algorithmically than graph partitioning. For example, we do not need to work with floating point arithmetic in any significant way with the graph algorithms of the algebraic methods as we do with the Delaunay mesh generation of the geometric method.

Plain aggregation can be called "algebraic" as the coarse grid spaces (the *rigid body modes* or constant translations in, and rotations about, the three coordinate axis) can, in principle, be constructed from the stiffness matrix (without Dirichlet boundary conditions imposed) [13]. However, these spaces are most easily constructed from the nodal coordinates (ie, geometry) for elasticity problems. Rigid body mode methods could thus be called "geometric", with our definition, as the stiffness matrix does not need to be used to construct the coarse grid spaces and they do have a concrete geometric interpretation (though a representation is never explicitly constructed). Many rigid body mode methods have been developed [9, 13]; the addition of smoothing can provides significant improvement to these methods [25, 20], especially on more challenging problems in solid mechanics such as shells. This smoothing requires the stiffness matrix and thus classifies this method as algebraic. These aggregates are constructed so that the nodes within an aggregate are "strongly connected" (see §4.1 and Vanek et. al. [25]); this aggregation strategy requires the stiffness matrix and thus classifies this method as an algebraic method when this aggregation method is used.

For elasticity problems, the smoothed aggregation multigrid method starts with the rigid body modes of the finite element problem as its initial coarse grid space, and then applies an iterative solver (ie, smoother) to the coarse grid functions themselves to reduce the energy of the coarse grid functions [28]. The fact that this smoothing can be done without any additional input from the user is the key to the practicality of this algorithm. Smoothed aggregation, as the name suggests, begins by aggregating fine grid nodes into strongly connected aggregates as described below. Each of these aggregates will produce a node on the coarse grid. This process is applied recursively until the top grid is small enough to solve quickly with the available direct solver. The user supplies the kernel of the stiffness matrix of the problem without any essential boundary conditions (a rectangular block vector, or tall skinny matrix, B_0). For most static, small deformation, 3D finite element formulations in solid mechanics there are six such kernel vectors - the rigid body modes. These rigid body modes are trivial to construct with the nodal coordinates and are probably a good basis even if there is no kernel for the operator (as with dynamic problems). These rigid body modes are used exclusively for elasticity problems and thus an n by 6 block vector B_0 is provided (with n degrees of freedom on the fine grid).

A high level view of the algorithm is as follows, starting on fine grid $i = 0$ with provided kernel block vector B_0.

1. Construct aggregates (nodal partitions) on the current (fine) grid i, as described in §4.1
2. For each aggregate j extract the submatrix B_i^j of B_i associated with the nodes in aggregate j
3. On each aggregate j construct the initial prolongator $\bar{P}_i^j$ with a QR factorization: $B_i^j \rightarrow \bar{P}_i^j B_{i+1}^I$
4. B_{i+1}^I is the kernel for coarse grid node I on the next grid $i+1$
5. The initial prolongator $\bar{P}_i$ for grid i is a (tall skinny) block diagonal matrix with $\bar{P}_i^j$ in the j^{th} diagonal block
6. Each column of the initial prolongator $\bar{P}_i$ is then "smoothed" with one iteration of a simple iterative method to provide the prolongator $\hat{P}_i$ for grid i: $\hat{P}_i \leftarrow (I - \omega D_i^{-1} A_i)\bar{P}_i$
7. The next grid operator is constructed algebraically: $A_{i+1} \leftarrow \hat{P}_i^T A_i \hat{P}_i$

This algorithm gives us all of the operators: $P_i = \hat{P}_i$, $R_i = P_i^T$, and A_i for grids 1 thru L (assuming L coarse grids). Note that each node on the coarse grids will have 6 degrees of freedom even if the original problem only has three degrees of freedom per node. D_i above can be any symmetric positive definite preconditioner matrix; the diagonal of A_i is used in this paper and $\omega = \frac{1.5}{\bar{\lambda}_i}$ where $\bar{\lambda}_i$ is an estimate of the highest eigenvalue of $D_i^{-1}A_i$ (see [20] for details). The construction of the aggregates in step 1 above is the last item in the algorithm that remains to be specified.

4.1 Construction of aggregates

The construction of the aggregates in smoothed aggregation is an important aspect of the implementation as the choice of aggregates can significantly effect the convergence rate and the complexity of the coarse grids which significantly effects the complexity of each iteration and the setup cost (ie, coarse grid construction). We have investigated two aggregation approaches: 1) maximal independent set (MIS) with post-processing similar to the original algorithm proposed by Vanek et.al. [25], and 2) graph partitioners with some post-processing to remove tiny aggregates as discussed in §4.1.1. Vanek et.al. recommend that aggregates should be "connected by a path of strong coupling" as a means of automatically achieving semi-coarsening (semi-coarsening has been empirically found to be effective for anisotropic problems such as some fluid flow problems [12]). Our experience indicates that the selection of strongly connected aggregates is effective for solid mechanics problems with large jumps in material coefficients as well.

Given a norm $\|\cdot\|$, the edge weight w_{ij} between two nodes i and j is computed with $w_{ij} = \frac{\|a_{ij}\|}{\sqrt{\|a_{ii}\|\|a_{jj}\|}}$, where a_{ij} is the d by d sub-matrix of the stiffness matrix associated with the degrees of freedom of node i and node j and d is the number of degrees of freedom per node on the grid (eg, 3 or 6 for our test problems). Note, for the symmetric positive definite problems $w_{ij} \leq 1.0$, if the two norm $\|a_{ij}\|_2$ is used, we use a combination of the one and infinity norms for simplicity $\|a_{ij}\| = (\|a_{ij}\|_1 + \|a_{ij}\|_\infty)/2.0$. The MIS is computed with a graph that has been modified by dropping edges that have weights that fall below a certain threshold ϵ; we use $\epsilon = 0.08 \cdot (\frac{1}{2})^l$, where l is the grid number (the fine grid is zero) as suggested by Vanek et.al [25]. Note, this is similar in spirit to the heuristics used in the geometric multigrid method that identifies nodes that should not be connected to each other in the MIS computation.

Common "greedy " MIS algorithms naturally construct a nodal partitioning; these partitions, however, tend to be too large (ie, the aggregates are too small) for smoothed aggregation as the complexity of the coarse grids tend to be larger than that which is optimal for the overall complexity of the solver. A post-processing step is thus advisable to increase the size of the aggregates. We iterate over the aggregates and coalesce the nodes in the smallest aggregates with nearby aggregates with which each has the largest sum of edge weights, constrained by requiring that the resulting aggregate be less than two times the minimum degree of nodes in the original aggregate. This heuristic is used to limit the size of aggregates as large aggregates would create a "bottleneck" in the convergence of the solver as the residual would be relatively poorly "covered" by the coarse grid correction on these large aggregates. The minimum degree term is meant to reflect the lower rate of coarsening that the MIS provides in "thin body" regions of the mesh and that is desirable for the convergence rate of the solver (see below).

4.1.1 Graph partitioner construction of aggregates

We have investigated an alternative to maximal independent set (MIS) based aggregates which recognizes that the aggregation problem can be formulated as a standard graph partitioning problem. This approach allows us to construct aggregates with a graph partitioner such as ParMetis [19]. Partitioning has the advantage that the *global* rate of reduction in coarse grid points can be explicitly specified - a potentially useful feature for the optimization of the solve time (ie, the convergence rate of the solver and the complexity of each iteration). Another advantage of graph partitioners is that one can leverage the expertise of algorithm development efforts in a general purpose graph partitioner that has been designed to minimize "edge cuts" and thereby maximize connectivity within each aggregate - the purpose of the aggregation heuristics in smoothed aggregation. General purpose graph partitioners are useful as the aggregation problem can be formulated as the standard graph partitioning problem of minimizing edge cuts and evenly partitioning node weights. We select node weights to appropriately coarsen *locally*; we select the number of aggregates to appropriately coarsen globally.

MIS partitions are effective at automatically coarsening thin bodies (plates or line geometries) *appropriately* (ie, reduction in nodes of about a factor of 8 to 27 for 3D geometries, 4 to 9 for 2D geometries, and 2 or 3 for 1D geometries). Vanek et. al. recommend a high coarsening ratio to reduce the complexity of the coarse grids; the upper limit on the coarsening ratio for MIS partitions is about the average number of neighbors (*degree*) for each node. As the number of edges in the graph is easily available we can chose a coarsening ratio of $\lfloor \frac{2E}{n} \rfloor + 1$, where E is the number of edges in the graph and n is the number of nodes in the graph. Note, as ParMetis has high memory complexity with respect to the number of partitions (aggregates) we are limited to constructing aggregates on each processor with the serial graph partitioner METIS which has the disadvantage of not maintaining the serial semantics in parallel (ie, partitions that would be computed with one processor can not, in general, be generated on multiple processors as the aggregate partitions must be nested in the processor partitions).

To aggregate planar (or linear) regions appropriately - locally - we can set the node weights in the graph partitioner so as to give a higher weight to planar regions. Node i with $|i|$ degree is given a weight $w_i = \frac{1}{|i|}$; this provides about the same weight per partition in a 27 node aggregate of interior nodes of a regular 3D mesh of hexahedra and a 9 node aggregate in the interior of a regular quadrilateral mesh. The partitioning method does not result in better solve times on any of our test problems; we have seen solve times of about 15

Thus, as we can not use ParMetis and must use METIS locally on each processor and as we do not see an increase in performance, we do not use the partitioning aggregation method for any of our numerical experiments. Though, we see the graph partitioner approach as potentially useful method and the development of effective graph partitioners for this purpose as an area of potential future work.

5 Parallel architecture

A highly scalable implementation of the algorithms and of a finite element application are used to test the solvers. The parallel finite element system is composed of two basic parts: 1) *Athena*, a parallel finite element program built on a serial finite element code (FEAP [14]) and a parallel graph partitioner (ParMetis [19]) and 2) our solver *Prometheus*. Prometheus can be further decomposed into three parts: 1) *Epimetheus*, general unstructured multigrid support (built on PETSc [3]); 2) the geometric multigrid method Prometheus; and 3) the algebraic multigrid method *Atlas*. Prometheus, Atlas, and Epimetheus are available as a publicly domain library called Prometheus [22].

Athena reads a large "flat" finite element mesh input file in parallel (ie, each processor seeks and reads only the part of the input file that it, and it alone, is responsible for), uses ParMetis to partition the finite element graph, and then constructs a complete finite element problem on each processor. These processor sub-problems are constructed so that each processor can compute all rows of the stiffness matrix, and entries of the residual vector, associated with vertices that have been partitioned to the processor. This negates the need for communication in the finite element operator evaluation at the expense of a little bit of redundant work.

Explicit message passing (MPI) is used for performance and portability and all parts of the algorithm have been parallelized for scalability. All components of multigrid can scale reasonably well.

Clusters of symmetric multi-processors (SMPs) are targeted as this seems to be the architecture of choice for future large machines. Clusters of SMPs are accommodated by first partitioning the problem onto the SMPs and then the local problem is partitioned onto each processor. This approach implicitly takes advantage of any increase in communication performance within each SMP, though the numerical kernels (in PETSc) are "flat" MPI codes.

6 Numerical results

This section investigates the performance of the three multigrid methods (geometric, aggregation, smoothed aggregation) on several problems in solid mechanics with up to 76 million degrees of freedom (dof). The goal of these experiments is to test a wide variety of features found in challenging finite element simulations so as to best inform application developers of the potential value of these algorithms and implementations. FEAP's trilinear hexahedra mixed linear elastic element and linear elastic four node quadrilateral shell element are used [14, 24]. Two linear elasticity continuum element material constitutions are used: a "hard" steel like material and a "soft" incompressible rubber-like material. Table 1 lists these material properties. The shell elements have a thickness of $\frac{1}{2000}$ of the principle (longest orthogonal) dimension of the mesh (eg, the "wing" problem in §6.4 is 50.0 units long and has a thickness of 0.025 units). Continuum element problems will use the "hard" material unless otherwise stated.

Material	Elastic mod. (E)	Poisson ratio
soft	10^{-4}	0.49
hard	1	0.3
shell	1	0.3

Table 1: Continuum materials

The test problems are designed to include several features of challenging finite element simulations: thin body domains (thin), large jumps in material coefficients (jumps), incompressible materials (incomp.), large scale problems with parameterized meshes for scalability studies (scale), and shell problems (shell). Table 2 lists the test problems and their properties.

Test problem	Section	n	Processors	thin	jumps	incomp.	scale	shell
Cone	§6.1	21,600	1	$\surd$				
Beam-column	§6.2	34,460	4	$\surd$				
Plate	§6.3	155,280	4					$\surd$
Wing	§6.4	617,760	16					$\surd$
Cylinder with large cutouts	§6.5	135,318	4					$\surd$
Cantilever	§6.6	62,208	3		$\surd$	$\surd$		
Sphere in a cube (Sp.)	§6.7	7,534,488	512		$\surd$	$\surd$		
Concentric Sp. (C.Sp.)	§6.8	80K - 76M	1 - 1024	$\surd$	$\surd$	$\surd$	$\surd$	

Table 2: Test problems and characteristics

All of the test problems are symmetric positive definite and thus preconditioned conjugate gradient (PCG) is used. One pre-smoothing and one post-smoothing step are used within multigrid, preconditioned with block Jacobi. The Jacobi blocks (sub-domains) are constructed with METIS.

The goal of this paper is to evaluate the effectiveness of the three multigrid methods: smoothed aggregation, (plain) aggregation, and geometric multigrid. These experiments are, to our knowledge, unique in that several scalable solver methods are tested head-to-head on the same problems, on the same machines and with the same multigrid infrastructure and numerical kernels, thus providing comparable performance data for these methods as the only parameter that is tested is the construction of the coarse grid spaces (manifested in the prolongation and restriction operators). This is the first time, to our knowledge, that multiple highly scalable, and fully parallelized, solvers have been tested end-to-end. Parallel unstructured solvers are inherently complex; conducting controlled experiments to compare methods is a challenge. Thus a few observation are in order to aid in interpreting this data:

- The nature of unstructured parallel computing today is such that results are non-deterministic. That is, we usually get slightly different performance results from one run to the next. This is primarily due to the fact that the parallel graph partitioner is not deterministic and we are in general allocated a different set of processors each time a job is run which can effect the performance. We observe variations of about 10% in Mflop rates in the solver from one run to the next.

- The nature of unstructured iterative solvers is such that the performance results are somewhat "ill-conditioned" in that small algorithmic changes (eg, changes in the number of processors used, small code changes) can noticeably change the performance of the solver. In general we have seen a variance of up to about 10% in convergence rate between runs with multiple processors for each of the three multigrid methods that we investigate. The parallel semantics are not identical to the serial semantics for either method (ie, the space of all possible coarse grid spaces for a problems is smaller or different when multiple processors are used). We have maintained nearly identical parallel semantics, but the price of maintaining identical semantics in parallel, which would be desirable, would significantly complicate the implementation and incur some performance penalty. The geometric method only meshes the coarse grids locally, hence the global coarse grid has overlapping regions and is not a valid finite element mesh. Aggregation for the parallel algebraic methods has some slight inconsistencies with the serial semantics. These inconsistencies are small and we have not observed any systematic decrease in convergence rate as more processors are used in most of the test problems with the exception of the geometric method when many more processors are used than is common practice on typical machines of today (ie, machines with 256 Mb or more of memory per processor). We have found, on a few of our test problems, that we can get increases in convergence rate of up to about 20% when one processor is used for both the algebraic methods and geometric method.

- We have worked with the geometric method for several years and the algebraic method for about a year; the algebraic method is much simpler (and is implemented with about half as much code) and we have done less algorithmic development on the algebraic method. We believe that we have optimized the algebraic method reasonably well but it may not be as highly developed as the geometric method.

There are three primary parts of a linear solve: 1) "mesh setup": setup for the mesh, 2) "matrix setup": setup for the matrix, and 3) the actual "solve for x". The mesh setup is work that needs to be done for each mesh (for a direct solver this is the symbolic factorization). Note, that all three of the solvers require essentially the same data for the mesh setup phase (ie, the fine grid mesh with coordinates). The setup for each matrix includes the construction of the coarse grid stiffness matrices, setup for the smoothers, the factorization for the coarsest grid, the smoothing of the coarse grid spaces for the smoothed aggregation method (for direct solvers this is the factorization). The matrix setup work need not be repeated in solving for multiple right hand sides, as in linear dynamic problem, and the only cost incurred are those of the solve phase (for direct solver this is the front substitution and back solve phase). The times for each of these three phases is reported for each of the test problems that follow.

Two machines are employed for these experiments:

- Cray T3E at NERSC has 640 single processor nodes total, 450 MHz., 900 Mflop/sec theoretical peak, 256 Mb memory per processor, and a peak Mflop rate of 662 Mflop/sec (1/2 of 2 processor Linpack "toward perfect parallelism" R_{max}).
- IBM PowerPC cluster at LLNL, has about 256 4-way-SMPs available to users, 332 MHz PowerPC 604e processors, 664 Mflop/sec theoretical peak, 512 Mb of memory per node, and a peak Mflop rate of 258 Mflop/sec (1/2 of 2 processor Linpack "toward perfect parallelism" R_{max} [29]).

6.1 Cone

The "cone" problems is mesh of linear hexahedral elements of a truncated cone with 21,600 degrees of freedom, fixed at the base and loaded at the end with a twisting load (see Figure 2). This problem is challenging as it has poorly proportioned elements with aspect ratios as high as 12:1, and has thin body features. This problem is run with a 64-block block Jacobi preconditioner for the PCG smoother. This problem has a condition number of about $3.5 \cdot 10^7$ and Figure 2 (left) shows the undeformed mesh and Figure 2 (right) shows the deformed mesh with the stress in the direction of the primary axis of the problem.

Table 3 shows the sizes of the coarse grids for this problem.

Solver	Smoothed aggregation			Plain agg.			Geometric		
Grid	0	1	2	0	1	2	0	1	2
Equations (eq.)	21,600	2,160	126	21,600	2,766	342	21,600	2,424	606
Ave. non-zeros per eq.	65	144	89	65	77	46	65	79	126

Table 3: Grid sizes for cone

Figure 3 (left) shows the residual history of this problem on one PowerPC processor and solved to a tolerance of 10^{-6} in units of the time to do one matrix-vector product on the fine grid.

This data shows that geometric multigrid and smoothed aggregation are the fastest and the aggregation method is the slowest. Figure 3 (right) shows the times for the three primary parts of a linear solve: setup for the mesh, setup for the matrix, and the actual solve. One matrix vector product on the fine 21,600 degrees of freedom grid takes 0.08 seconds on one PowerPC processor.

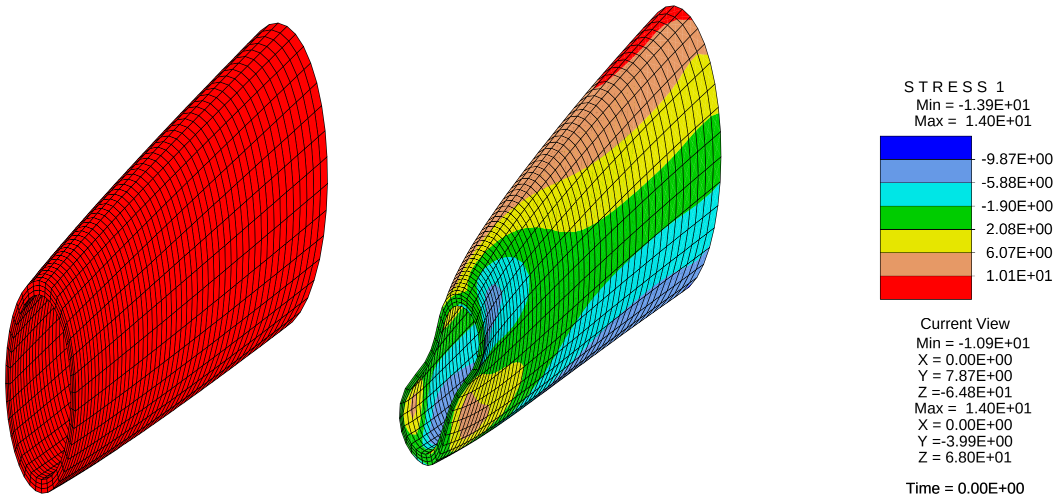


Figure 2: Truncated cone

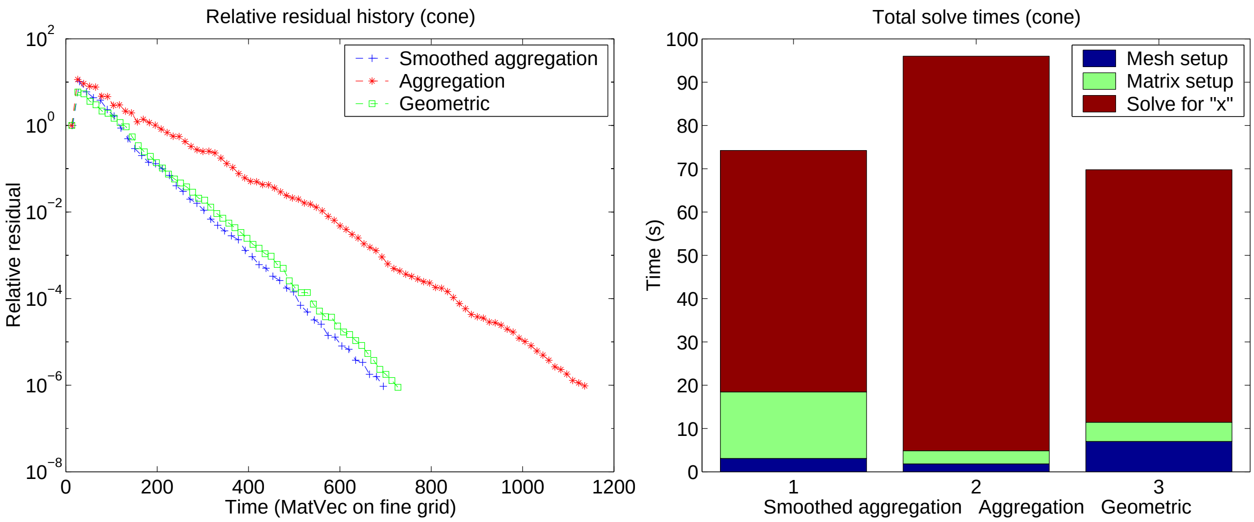


Figure 3: (left) Truncated cone residual history vs. solve times and (right) sum of mesh setup, matrix setup and solve times on one PowerPC processor

6.2 Beam-column

The next problem is that of a steel "beam-column" connection with 34,460 degrees of freedom and is meshed with thin (poorly proportioned) hexahedral elements. This problem is solved with a 64-block block Jacobi preconditioner for the PCG smoother. This problem has a condition number of about $1.0 \cdot 10^8$ and Figure 4 (left) shows the undeformed mesh and Figure 4 (right) shows the deformed mesh with the first principle stress.

Table 4 shows the sizes of the coarse grids for this problem.

Figure 5(left) shows the residual history of this problem on four PowerPC processors and solved to a tolerance of 10^{-6} in units of the time to do one matrix-vector product on the fine grid.

This data shows that the geometric multigrid method is the fastest by a small margin and the aggregation method is the slowest. Figure 5 (right) shows the times for the three primary parts of a linear solve: setup for the mesh, setup for the matrix, and the actual solve. One matrix vector product on the fine 34,460 dof grid takes 0.0394 seconds on four PowerPC processors.

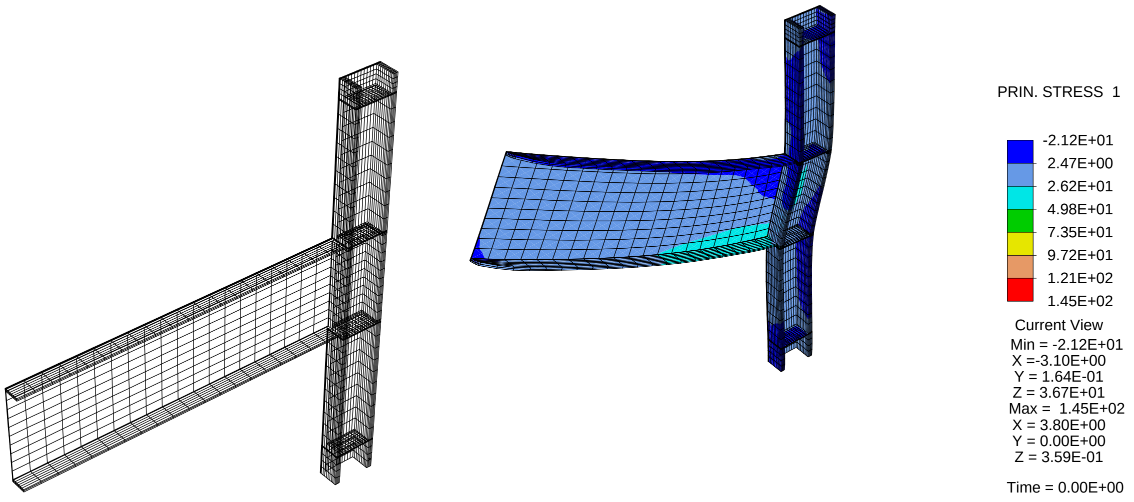


Figure 4: Beam Column

Solver	Smoothed aggregation			Plain agg.			Geometric			
Grid	0	1	2	0	1	2	0	1	2	3
Equations (eq.)	34,460	3,456	180	34,460	3,402	396	34,460	4,539	789	144
Ave. non-zeros per eq.	65	168	108	65	67	45	65	89	115	89

Table 4: Grid sizes for beam-column

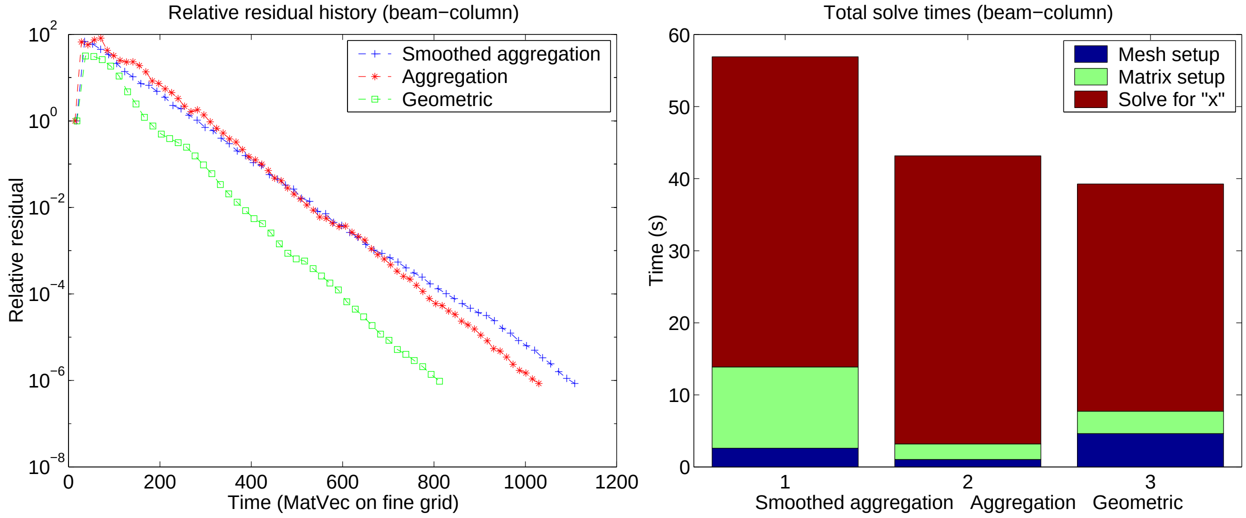


Figure 5: (left) Beam-column residual history vs. solve times and (right) sum of mesh setup, matrix setup and solve times on four PowerPC processors

6.3 Plate

The next problem is that of a flat square "plate" with fully clamped boundary condition on one quarter of one side and a uniform load down. The plate is meshed with four node quadrilateral shell elements and has 155,280 degrees of freedom (note, as this problem is linear it has no in plane stress nor "drilling" rotations and thus has one half as many true degrees of freedom, ie, three per node). This problem is run with a 257-block block Jacobi preconditioner for the PCG smoother. This problem has a condition number of over

$4.0 \cdot 10^{10}$ and Figure 6 (left) shows the undeformed mesh of an 8K dof version of this problem and Figure 6 (right) shows the deformed mesh with the first principle stress.

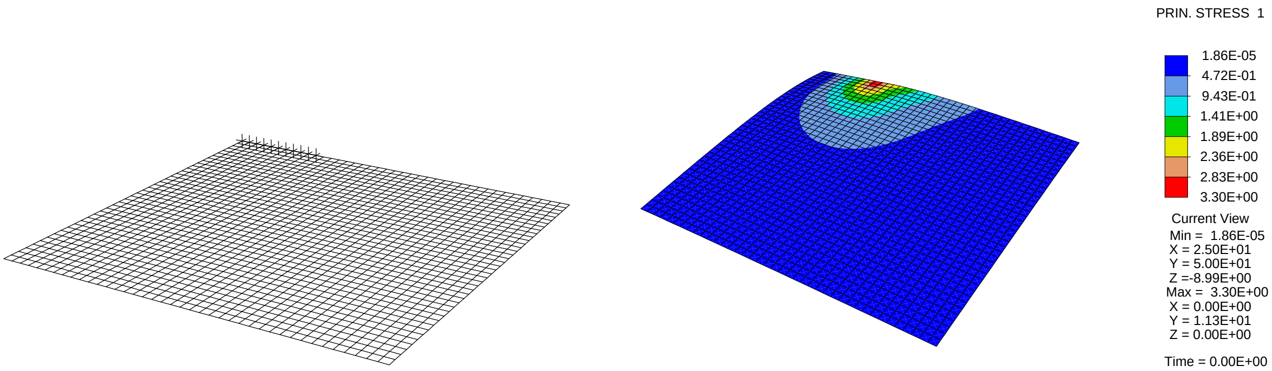


Figure 6: Flat plate

Table 5 shows the sizes of the coarse grids for this problem.

Solver	Smoothed aggregation			Plain agg.				Geometric			
Grid	0	1	2	0	1	2	3	0	1	2	3
Equations (eq.)	155,280	18,840	1,764	155,280	20,922	3,408	564	155,280	29,550	4,872	576
Ave. non-zeros per eq.	53	84	124	53	43	41	38	53	72	102	94

Table 5: Grid sizes for the flat plate

Figure 7(left) shows the residual history of this problem on four PowerPC processors and solved to a tolerance of 10^{-4} in units of the time to do one matrix-vector product on the fine grid.

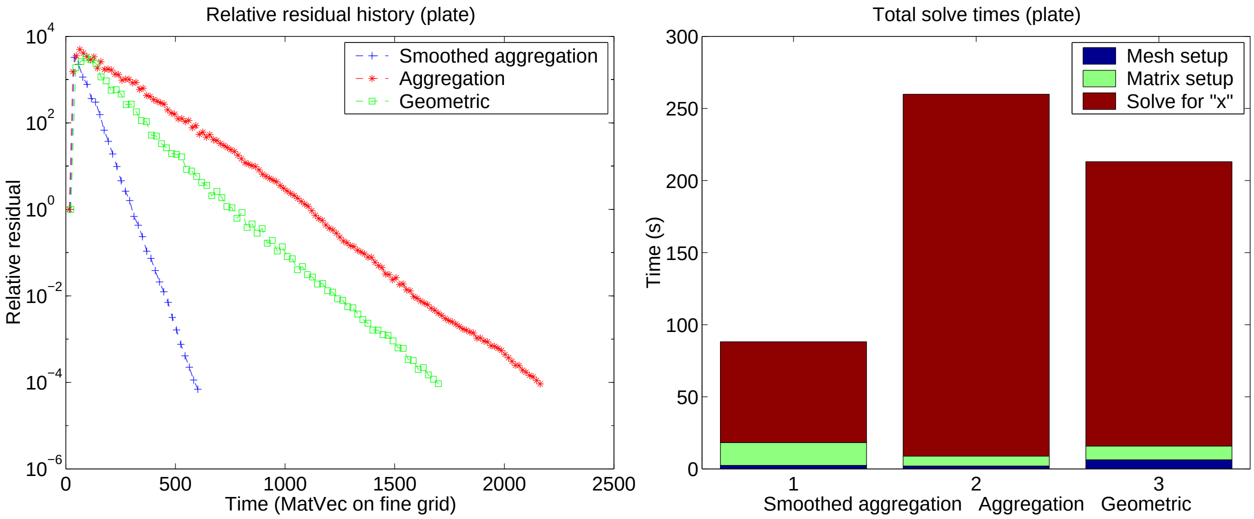


Figure 7: (left) Flat plate residual history vs. solve times and (right) sum of mesh setup, matrix setup and solve times on four PowerPC processors

This data shows that smoothed aggregation multigrid is the fastest by far and that the aggregation method is the slowest. Figure 7 (right) shows the times for the three primary parts of a linear solve: setup for the mesh, setup for the matrix, and the actual solve. One matrix vector product on the fine 155,280 dof grid takes 0.1160 seconds on four PowerPC processors.

6.4 Wing

The next problem is a "wing" like mesh with fully clamped boundary condition at the base and a uniform load down on the under side of the wing. The wing is meshed with four node quadrilateral shell elements, has 561,330 degrees of freedom, and four internal stiffener plates. This problem is run with a 560-block block Jacobi preconditioner for the PCG smoother. This problem has a condition number of about $1.0 \cdot 10^9$ and Figure 8 (left) shows the undeformed mesh of an 22,000 dof version of this problem and Figure 8 (right) shows the deformed mesh with the first principle stress.

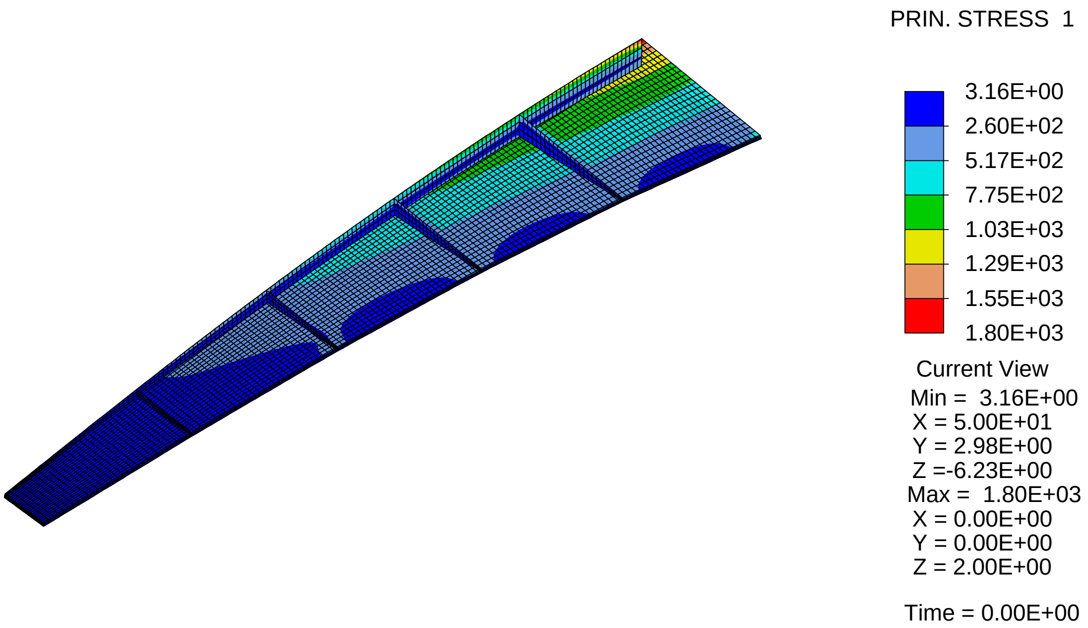


Figure 8: Simple wing (50K dof version)

Table 6 shows the sizes of the coarse grids for this problem.

Solver	Smoothed aggregation				Plain agg.				Geometric				
Grid	0	1	2	3	0	1	2	3	0	1	2	3	4
Equations (eq.)	561K	52K	4,506	270	561K	53K	6,768	876	561K	129K	25K	7K	1,116
Ave. non-zeros per eq.	54	79	130	99	54	44	43	43	54	67	113	161	201

Table 6: Grid sizes for the simple wing

Figure 9(left) shows the residual history of this problem on 16 PowerPC processors and solved to a tolerance of 10^{-4} in units of the time to do one matrix-vector product on the fine grid.

This data shows that the smoothed aggregation method is the fastest by a small margin and the aggregation method is the slowest. Figure 9 (right) shows the times for the three primary parts of a linear solve: setup for the mesh, setup for the matrix, and the actual solve. One matrix vector product on the fine 617,760 dof grid takes 0.109 seconds on 16 PowerPC processors.

6.5 Tube with holes

The next problem is shell element mesh of a "cylinder with large cutouts", roller boundary conditions on the base (with minimal constraints for stability), and several point loads. It is meshed with four node quadrilateral shell elements and has 135,318 degrees of freedom. This problem is run with a 67-block block Jacobi preconditioner for the PCG smoother. This problem has a condition number of over $1.0 \cdot 10^9$ and

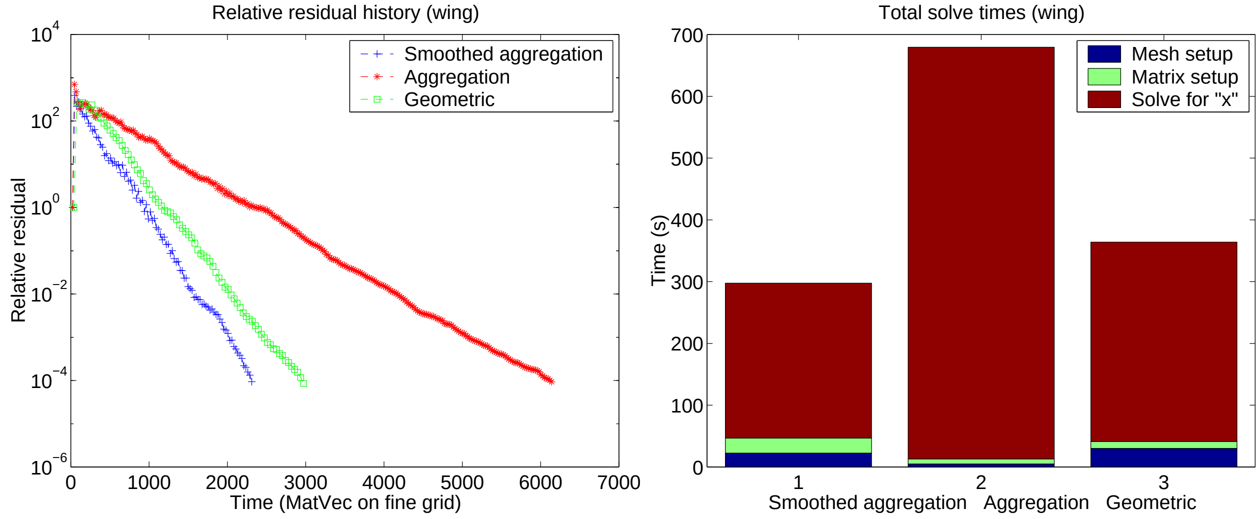


Figure 9: (left) Simple wing residual history vs. solve times and (right) sum of mesh setup, matrix setup and solve times on 16 PowerPC processors

Figure 10 (left) shows the undeformed mesh of an 13,000 dof version of this problem and Figure 10 (right) shows the deformed mesh with the first principle stress.

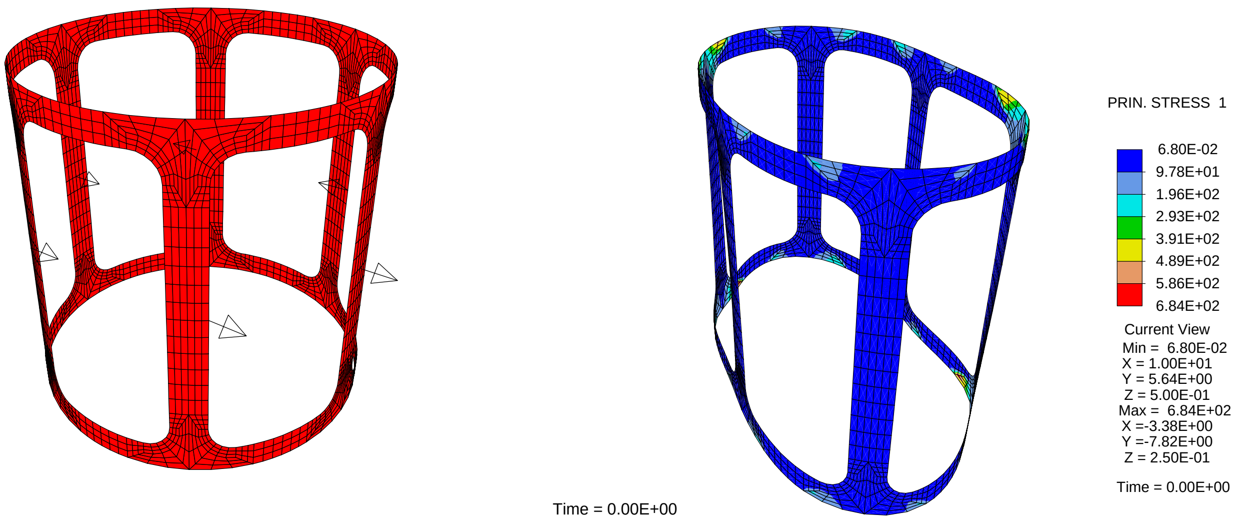


Figure 10: Tube with holes (13K dof version)

Table 7 shows the sizes of the coarse grids for this problem.

Solver	Smoothed aggregation			Plain agg.			Geometric		
Grid	0	1	2	0	1	2	0	1	2
Equations (eq.)	135K	14K	1,488	135K	14K	2,034	135,318	34K	5,448
Ave. non-zeros per eq.	52	67	75	52	39	32	52	88	135

Table 7: Grid sizes for the tube with holes

Figure 11(left) shows the residual history of this problem on four PowerPC processors and solved to a tolerance of 10^{-4} in units of the time to do one matrix-vector product on the fine grid.

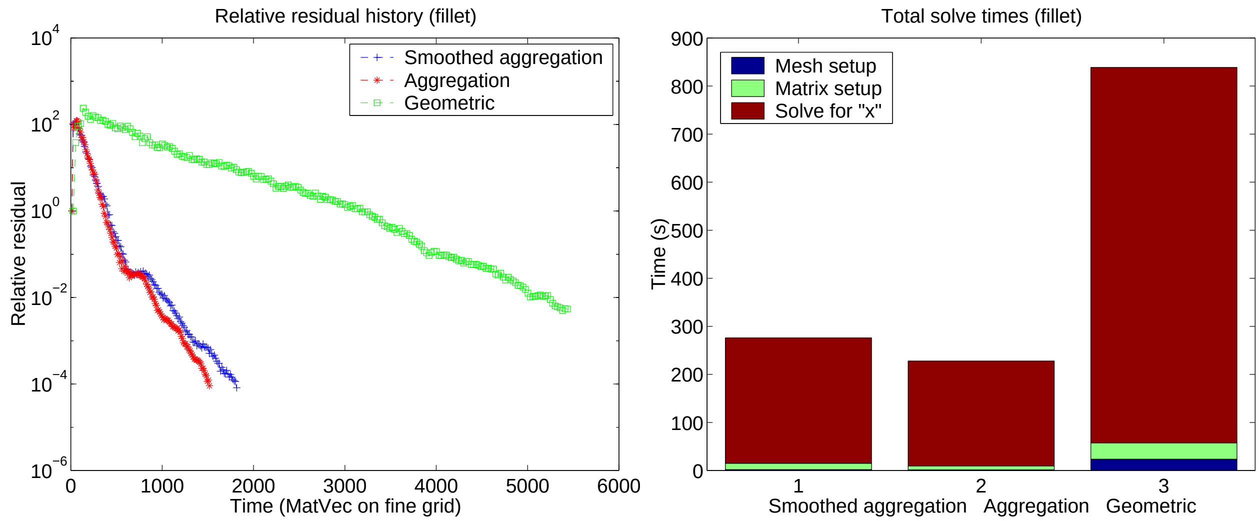


Figure 11: (left) Tube with holes residual history vs. solve times and (right) sum of mesh setup, matrix setup and solve times on four PowerPC processors

This data shows that smoothed aggregation multigrid is the fastest and that the geometric method is the slowest. Figure 11 (right) shows the times for the three primary parts of a linear solve: setup for the mesh, setup for the matrix, and the actual solve (note, the geometric method was not run to completions as shown in the left plot of Figure 11). One matrix vector product on the fine 135,318 dof grid takes 0.0992 seconds on four PowerPC processors.

6.6 Soft section cantilever

The "cantilever" problem is 8 by 8 by 256 mesh of linear hexahedral (cube) elements of a "cantilever" with 62,208 degrees of freedom and three layers of the "soft" material in the middle, fixed at the base and loaded at the end (see Figure 12). This problem is challenging as it poorly conditioned with a large jump in material coefficients and as the elastic modulus of the soft material goes to zero the problem becomes singular.

This problem is run with a 512-block block Jacobi preconditioner for the PCG smoother. This problem has a condition number of over $5 \cdot 10^{10}$.

Table 8 shows the sizes of the coarse grids for this problem.

Solver	Smoothed aggregation			Plain agg.			Geometric				Geometric (3 levels)		
Grid	0	1	2	0	1	2	0	1	2	3	0	1	2
Equations (eq.)	62K	5,550	282	62K	5,406	576	62K	6,807	744	189	62K	7,335	954
Ave. non-zeros per eq.	69	180	56	69	72	44	69	89	65	49	69	84	72

Table 8: Grid sizes for soft section cantilever

Figure 13(left) shows the residual history of this problem on three PowerPC processors and solved to a tolerance of 10^{-6} in units of the time to do one matrix-vector product on the fine grid. This problem is a good example of robustness problems of the geometric method on pathological problems. It is well know that if the coarse grids can capture the material discontinuities on all of the coarse meshes that the convergence rate of geometric multigrid will not deteriorate with large jumps in material coefficients [23]. The geometric method can not guarantee that material discontinuities are articulated on the coarse mesh (in fact material discontinuities need to be neglected on problems with many fine geometric features for complexity reasons). This problem thus "breaks" the geometric method as it does not maintain a material boundary on the coarsest grid (that should be maintained) as can be inferred by the performance with one fewer coarse grids which show the faster results that we expect ("Geometric (3 levels)" in Figure 13).

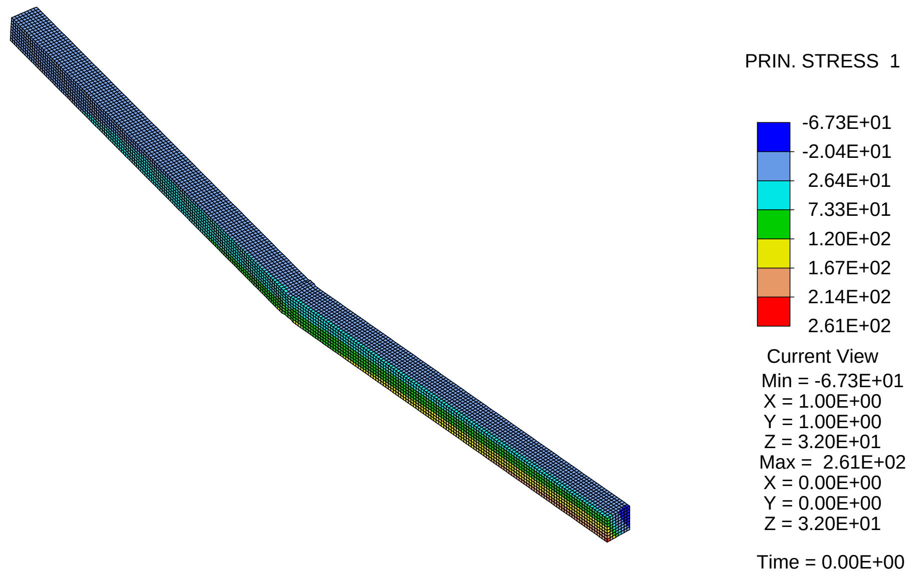


Figure 12: Soft section cantilever

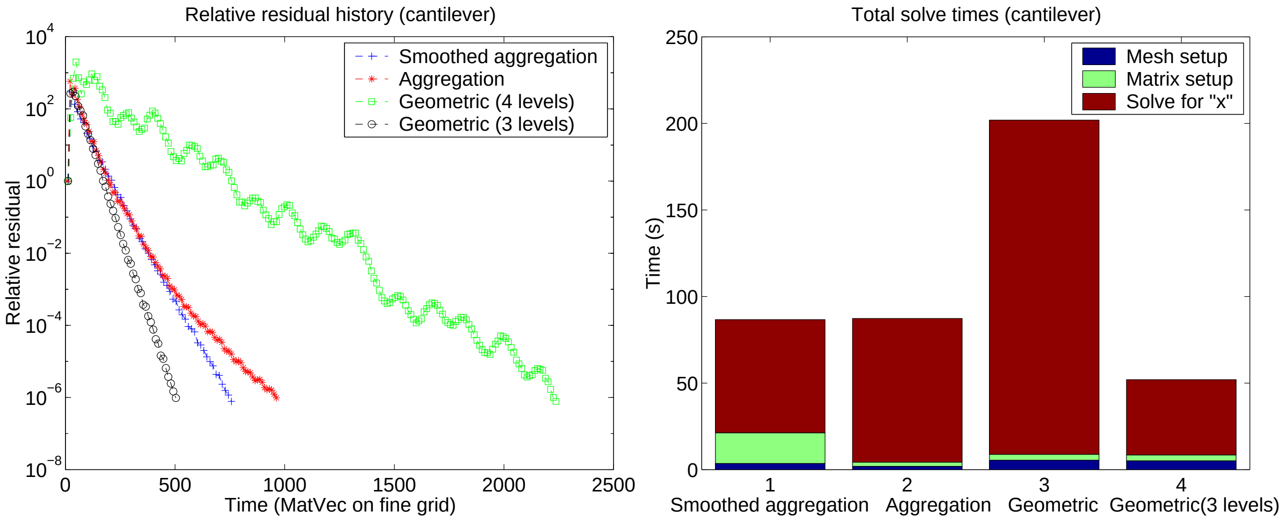


Figure 13: (left) Soft section cantilever residual history vs. solve times and (right) sum of mesh setup, matrix setup and solve times on three PowerPC processors

This data shows that the geometric multigrid method is the fastest by a small margin and the aggregation method is the slowest, but that the geometric method has robustness problems. Figure 13 (right) shows the times for the three primary parts of a linear solve: setup for the mesh, setup for the matrix, and the actual solve. One matrix vector product on the fine 62,208 dof grid takes 0.0915 seconds on three PowerPC processors.

6.7 Sphere in a cube

The next problem is hard sphere embedded in a soft cube and crushed with a uniform pressure load on two sides. One octant is meshed with linear hexahedral elements with 7,534K degrees of freedom and with symmetric boundary conditions (see Figure 14). This problem is challenging as it has a large jump in material coefficients. This problem is run with a 59,904-block block Jacobi preconditioner for the PCG smoother. This problem has a condition number of about $3 \cdot 10^8$.

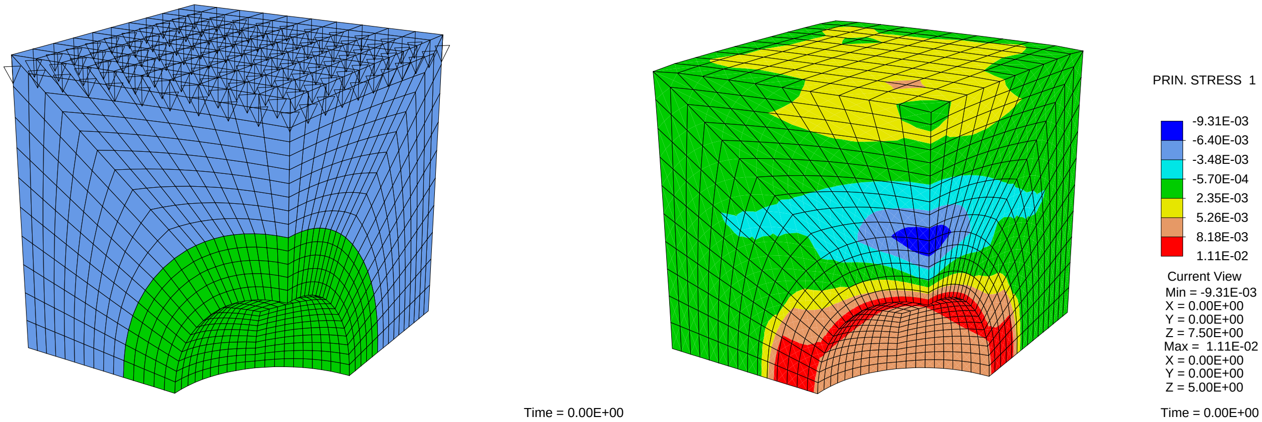


Figure 14: Sphere in a cube (15K dof version)

Table 9 shows the sizes of the coarse grids for this problem.

Solver	Smoothed aggregation				Plain agg.					Geometric				
Grid	0	1	2	3	0	1	2	3	4	0	1	2	3	4
Equations (eq.)	7.5M	492K	12K	240	7.5M	492K	32K	2,322	204	7.5M	642K	32K	1,938	216
Ave. non-zeros per eq.	80	333	574	219	80	106	98	86	60	80	176	216	170	121

Table 9: Grid sizes for sphere

Figure 15(left) shows the residual history of this problem on 512 processors of a Cray T3E and solved to a tolerance of 10^{-6} in units of the time to do one matrix-vector product on the fine grid.

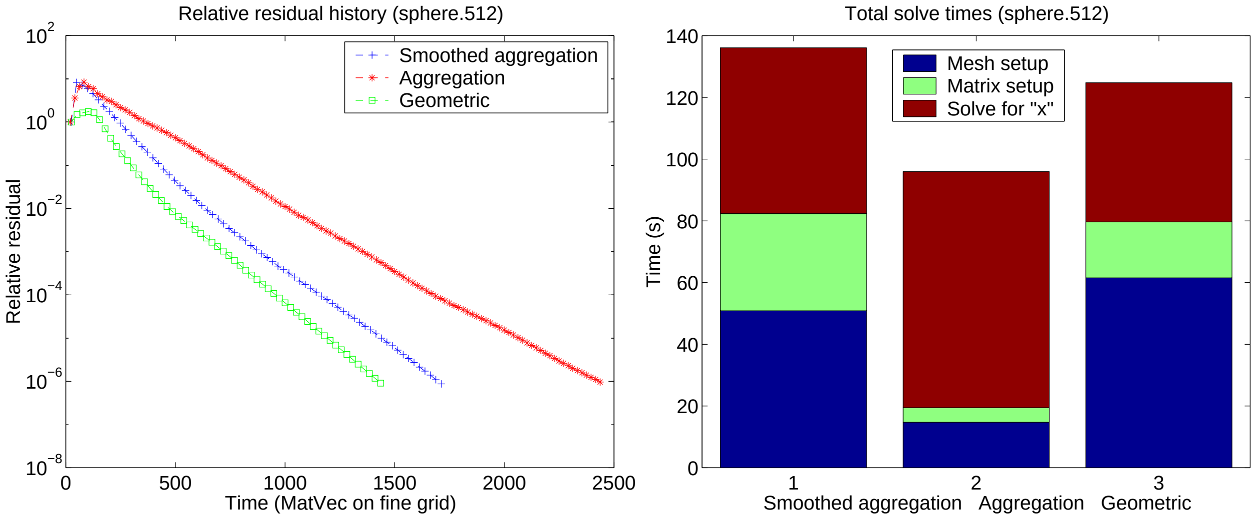


Figure 15: (left) Sphere in a cube residual history vs. solve times and (right) sum of mesh setup, matrix setup and solve times on 512 Cray T3E processor

This data shows that the geometric multigrid method is the fastest and the aggregation method is the slowest. Figure 15 (right) shows the times for the three primary parts of a linear solve: setup for the mesh, setup for the matrix, and the actual solve.

One matrix vector product on the fine 7,534,488 dof grid takes 0.0314 seconds on 512 processors of a Cray T3E.

6.8 Concentric spheres in cube

The "concentric sphere" problem is a series of 17 alternating hard and soft concentric spheres embedded in a soft cube and crushed with displacement boundary conditions on two sides. Like the problem in the previous section, one octant is meshed with linear hexahedral elements with symmetric boundary conditions (see Figure 16). This problem is challenging as it has many large jumps in material coefficients and thin body features. This problem has a parameterized mesh so as to perform scalability studies. The smallest version of the problem has 79,679 degrees of freedom and the largest version has 76,395,359 degrees of freedom.

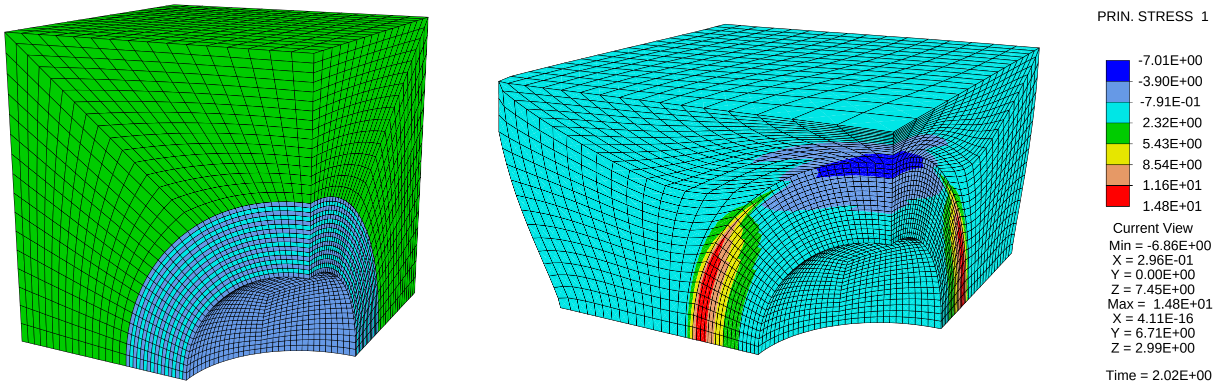


Figure 16: Concentric spheres in a cube (80K dof version)

Figures 17 and 18 (left) show the times for the three major phases of one linear solve (setup for the mesh, setup for the matrix, and the actual solve), for each of the three solver methods on 3 to 640 processors of a Cray T3E solved to a tolerance of 10^{-6} plotted against the $\log_{10}$ of the number of processors. There are about 27,000 dof per processor and the largest problem has 16,553,759 dof. This problems are run with a block diagonal preconditioner for the PCG smoother with about $\lfloor \frac{n}{125} \rfloor$ blocks (with n equal to the number of degrees of freedom).

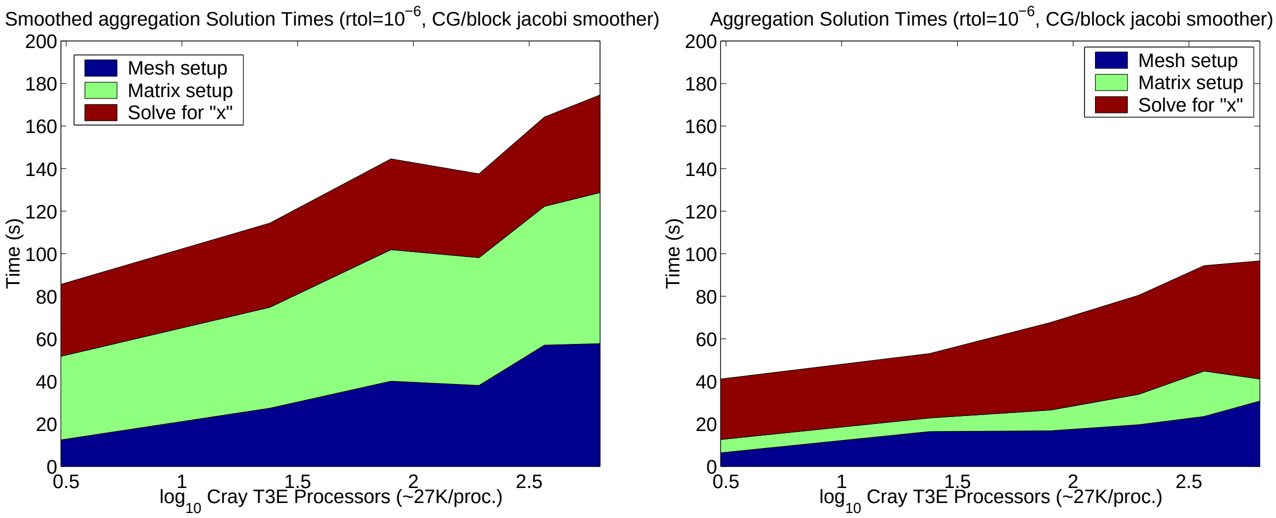


Figure 17: Concentric spheres in a cube, sum of mesh setup, matrix setup and solve times for algebraic methods, vs. $\log_{10}$ of number of Cray T3E (from 3 to 640) processors

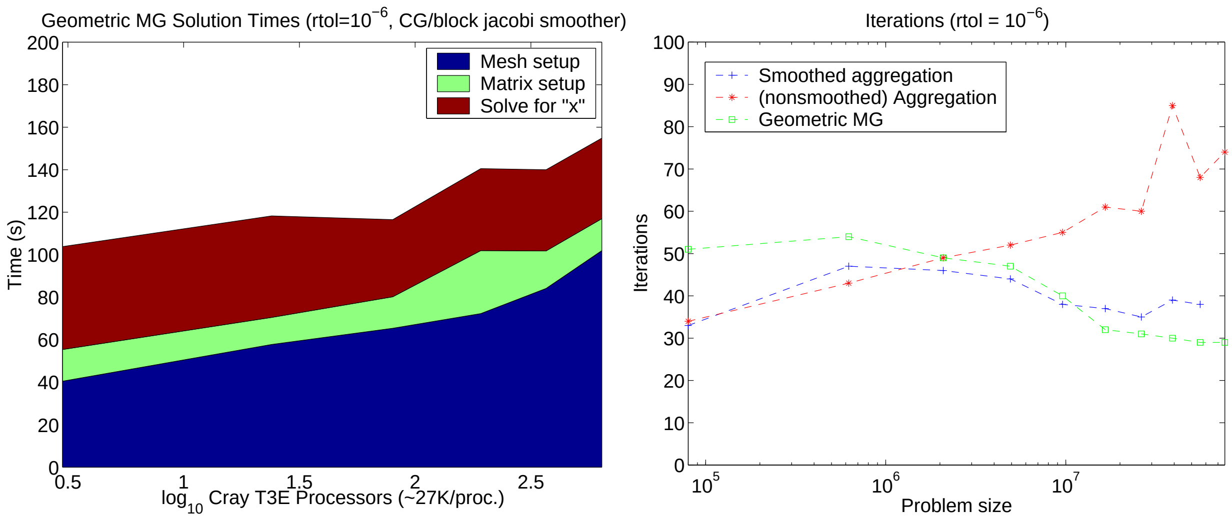


Figure 18: (left) Concentric spheres in a cube, sum of mesh setup, matrix setup and solve times vs. $\log_{10}$ of number of Cray T3E (from 3 to 640) processors for the geometric method and (right) iteration counts from IBM with up to 76M dof vs. $\log_{10}$ of number of PowerPC processors

This data shows that the aggregation multigrid method is the fastest and the smoothed aggregation method is the slowest for one complete linear solve. However, the geometric method is fastest for multiple solves (as the setup phase is amortized). Figures 19 and 20 (left) show the residual history, in terms of iterations, of the three solvers, run on one to 1024 processors of an IBM SP with about 80,000 dof per processor. Note, the largest smoothed aggregation data point is not available. This problems are run with a $\lfloor \frac{n}{128} \rfloor$ block block Jacobi preconditioner for the PCG smoother. Figure 17 (right) shows the iteration counts of the three solver methods plotted against the log of the problem size and shows that the number of iteration for the geometric method is decreasing, the smoothed aggregation method iterations are about constant, and the aggregation method iterations are increasing. This data reflects what the theory predicts - namely the convergence rate (both asymptotic and total) is decreasing for the aggregation method, staying relatively steady for the smoothed aggregation method and increasing for the geometric method. Thus, this data suggests that the geometric method is scaling well and the (plain) aggregation method is scaling poorly.

Figure 20 (right) and Table 10 show the average number of non-zeros per row of the stiffness matrices for each solver on the largest problem. From this data we can see a large growth in the number of non-zeros per row of the smoothed aggregation coarse grids (which results in relatively high memory complexity for the smoothed aggregation method) that is partially ameliorated by the faster rate of coarsening (recall that as we use MIS coarsening, denser graphs and hence higher average node degree will result in faster coarsening, see Table 10). Table 10 shows the sizes of the coarse grids for the largest problem.

Solver	Smoothed aggregation					Plain agg.						Geometric					
Grid	0	1	2	3	4	0	1	2	3	4	5	0	1	2	3	4	5
Eeqs.	56M	3.9M	91K	3.2K	168	56M	4.1M	356K	35K	4.1K	576	56M	4.6M	212K	10K	840	66
nz/eq.	80	364	834	1351	168	80	116	123	111	102	576	80	178	230	209	159	63

Table 10: Grid sizes for concentric spheres

7 Conclusion

We have shown that the three multigrid methods geometric, smoothed aggregation and plain aggregation are effective on many of our test problems in 3D solid mechanics. We have also introduced a new method for

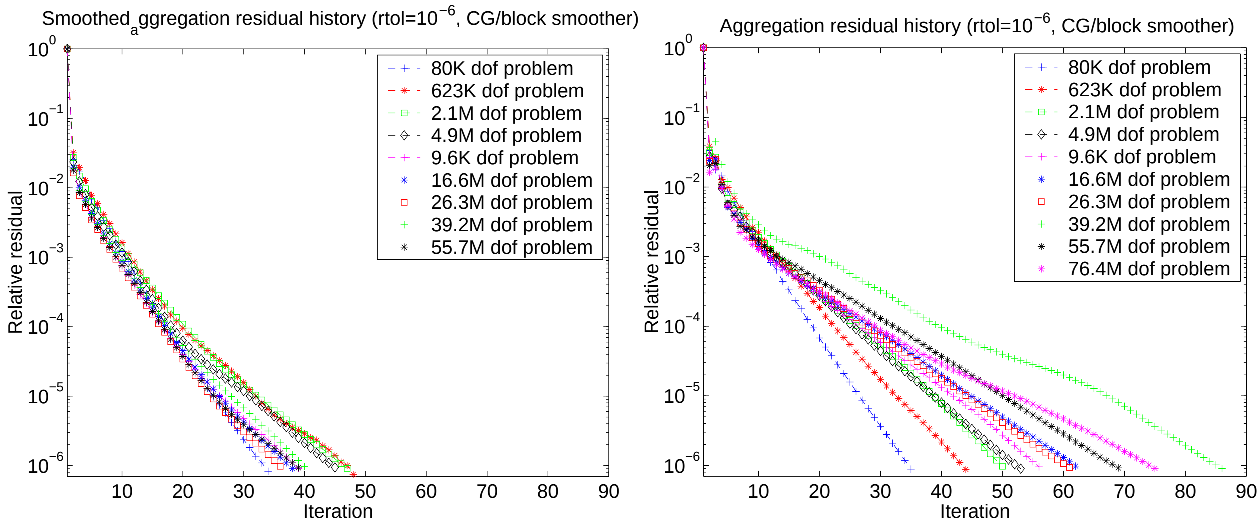


Figure 19: Concentric spheres in a cube residual histories, vs. iterations, on one to 1024 PowerPC processors for algebraic methods

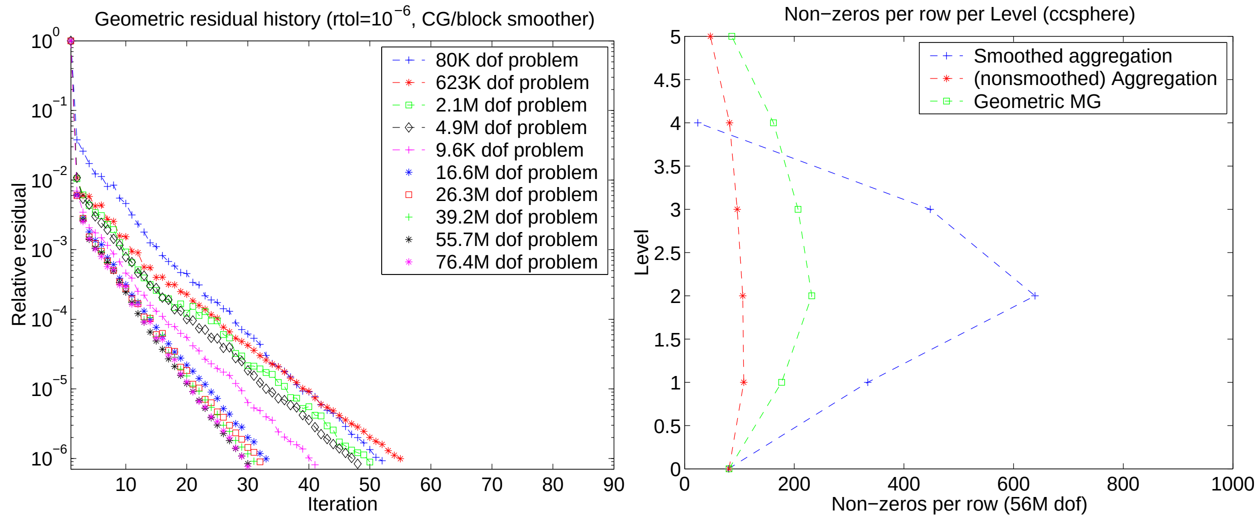


Figure 20: (left) Concentric spheres in a cube residual histories, vs. iterations, on one to 1024 PowerPC processors for geometric method and (right) average non-zeros per row for all grids of largest problem

constructing the aggregates for the algebraic methods that exploits the fact that the aggregation problem can be formulated as a standard graph partitioning problem. Plain aggregation has been shown to have the smallest setup times but has the longest solve times. Additionally, plain aggregation has demonstrated its scalability problems as theory predicts (as described in §1.1.1 and the references therein). Our geometric multigrid method has been shown to be effective on continuum element problems with many of the smallest solve times, but with the highest setup costs of the three methods. The geometric method also has some potential robustness problems stemming from the complex nature of grid coarsening. The smoothed aggregation method is shown to be an effective, scalable and robust method, particularly on shell problems. For the solve times, the smoothed aggregation and geometric methods were each the fastest in four of our eight test problems. For one complete linear solve, the geometric method was the fastest in three of the test problems, the smoothed aggregation method was fastest in two, and the plain aggregation method the was fastest in three of the eight test problems.

Future work includes continuing to test a variety of linear elasticity problems and to explore other application areas such as nonlinear elasticity and plasticity, fluid dynamics problems and multi-physics problems.

Acknowledgments. I would like thank my dissertation advisor and research supervisor Jim Demmel for providing invaluable critical feedback in the development of this work. I would also like to thank Marian Brezina for helping with the background in multigrid convergence theory. I would like to thank the many people that have contributed libraries to this work: R.L. Taylor for providing FEAP, the PETSc team for providing PETSc, George Karypis for providing ParMetis. I would also like to thank Steve Ashby and the ALE3D group at LLNL for proving support for this work. Lawrence Livermore National Laboratory for providing access to its computing systems and to the staff of Livermore Computing for their support services. Lawrence Berkeley National Laboratory for the use of their Cray T3E, and their helpful support staff - this research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Energy Research of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. An allocation of computer time from the Center for High Performance Computing at the University of Utah is gratefully acknowledged. This research was supported in part by the Department of Energy under DOE Contract No. W-7405-ENG-48 and DOE Grant Nos. DE-FG03-94ER25219, DE-FG03-94ER25206.

References

[1] M. F. Adams. A parallel maximal independent set algorithm. In *Proceedings 5th copper mountain conference on iterative methods*, 1998.

[2] M. F. Adams. Parallel multigrid solvers for 3D unstructured finite element problems in large deformation elasticity and plasticity. *International Journal for Numerical Methods in Engineering*, 48(0000-0000), 2000. To appear.

[3] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. PETSc 2.0 users manual. Technical report, Argonne National Laboratory, 1996.

[4] R.E. Bank and T. Dupont. An optimal order process for solving finite element equations. *Math. Comp.*, 36:35–51, 1981.

[5] J.H. Bramble. *Multigrid methods.* Longman Scientific and Technical, 1993.

[6] J.H. Bramble, J.E. Pasciak, and J Xu. The analysis of multigrid algorithms with nonnested spaces or noninherited quadratic forms. *Math. Comp.*, 56(193):1–34, January 1991.

[7] Marian Brezina. *Robust Iterative Solvers on Unstructured Meshes.* PhD thesis, University of Colorado at Denver, Denver, CO 80217-3364, 1997.

[8] W. Briggs. *A Multigrid Tutorial.* SIAM, 1987.

[9] V. E. Bulgakov and G. Kuhn. High-performance multilevel iterative aggregation solver for large finite-element structural analysis problems. *International Journal for Numerical Methods in Engineering*, 38:3529–3544, 1995.

[10] T. F. Chan and B. F. Smith. Multigrid and domain decomposition on unstructured grids. In David F. Keyes and Jinchao Xu, editors, *Seventh Annual International Conference on Domain Decomposition Methods in Scientific and Engineering Computing.* AMS, 1995. A revised version of this paper has appeared in ETNA, 2:171-182, December 1994.

[11] J. Demmel. *Applied Numerical Linear Algebra.* SIAM, 1997.

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

[13] C Farhat and F-X Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. *International Journal for Numerical Methods in Engineering*, 32:1205–1227, 1991.

[14] FEAP. www.ce.berkeley.edu/$\sim$ rlt.

[15] Y. T. Feng, D Peric, and D. R. J. Owen. A non-nested multi-grid method for solving linear and nonlinear solid mechanics problems. *Compute. Meth. Mech. Engng.*, 144:307–325, 1997.

[16] D. A. Field. Implementing Watson's algorithm in three dimensions. In *Proc. Second Ann. ACM Symp. Comp. Geom.*, pages 246–259, 1986.

[17] J. Fish, M. Pandheeradi, and V. Belsky. An efficient multilevel solution scheme for large scale non-linear systems. *International Journal for Numerical Methods in Engineering*, 38:1597–1610, 1995.

[18] H. Guillard. Node-nested multi-grid with Delaunay coarsening. Technical Report 1898, Institute National de Recherche en Informatique et en Automatique, 1993.

[19] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. *Proceedings of SC96: High Performance Networking and Computing*, 1996.

[20] J. Mandel, M. Brezina, and P. Vaněk. Energy optimization of algebraic multigrid bases. Technical Report 125, UCD/CCM, February 1998.

[21] J.D. Muller and M. B. Giles. Edge-based multigrid schemes for hybrid grids. *Numerical Methods for Fluid Dynamics*, VI, ICFD 1998. M. J. Baines ed.

[22] Prometheus. www.cs.berkeley.edu/$\sim$madams.

[23] B. Smith, P. Bjorstad, and W. Gropp. *Domain Decomposition.* Cambridge University Press, 1996.

[24] R. L. Taylor. Finite element analysis of linear shell problems. In J.R. Whiteman, editor, *The Mathematics of Finite Elements and Applications VI*, pages 191–203. Academic Press, London, 1988.

[25] P. Vanek, J. Mandel, and M. Brezina. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. In *7th Copper Mountain Conference on Multigrid Methods*, 1995.

[26] P. Vaněk, J. Mandel, and M. Brezina. Convergence of algebraic multigrid based on smoothed aggregation. *Numer. Math.*, 2000. To appear.

[27] P. Vaněk, Jan Mandel, and Marian Brezina. Algebraic multigrid on unstructured meshes. UCD/CCM Report 34, Center for Computational Mathematics,University of Colorado at Denver, December 1994.

[28] W. L. Wan. An energy minimizing interpolation for multigrid methods. Technical Report 97-18, UCLA, April 1997. Department of Mathematics.

[29]

[30] S. Zhang. Optimal-order nonnested multigrid methods for solving finite element equations i: on quasi-uniform meshes. *Math. Comp.*, 55:23–36, July 1990.

[31] S. Zhang. Optimal-order nonnested multigrid methods for solving finite element equations ii: on non-quasi-uniform meshes. *Math. Comp.*, 55:439–450, October 1990.