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

Mark Adams *

November 3, 2000

## 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 grid problems, in which it is often difficult or impossible for the application to provide coarse grids, is not as well understood. In particular methods that are designed to require only data that is easily available in most finite element applications (ie, fine grid data), constructing the grid transfer operators and coarse grid operators internally, are of practical interest. We investigate three unstructured multigrid methods that show promise for challenging problems in 3D elasticity: 1) non-nested geometric multigrid, 2) smoothed aggregation and 3) plain aggregation algebraic multigrid. This paper evaluates the effectiveness of these three methods on several unstructured grid problems in 3D elasticity with up to 76 million degrees of freedom.

Key words: unstructured multigrid, algebraic multigrid, parallel sparse solvers, finite element 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 are often 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 are still, popular because 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 as well understood and is currently an active area of research. Multigrid theory can not provide hard bounds on the convergence rate of multilevel methods on unstructured grid 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 iterative 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 $h$) 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 provably $O(n)$ 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).

Note, truly optimal convergence is actually $O(1)$ iterations, or work per unknown, to attain accuracy to the *discretization* error of the problem. 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, $10^{-6}$ for most problems in §6). We see some signs of this truly optimal convergence with the non-nested geometric method in §6.

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 aggregation multigrid methods.

## 1.1 Unstructured multigrid methods

A multigrid method can be defined by the method's coarse grid function space (see §2 or [27]). Unstructured non-nested *geometric* multigrid methods construct explicit coarse grids from the application provide fine grid and use standard finite element function spaces to construct the grid transfer operators. *Algebraic* multigrid methods are unstructured by definition and work only with the graphs of the "grids" and usually use the actual values in the operator matrices to construct the grid transfer operators. We are aware of two classes of multigrid methods that show promise for elasticity problems: non-nested geometric [2, 18, 17, 24] and aggregation algebraic based on the rigid body modes [10, 29]. We are interested in 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 grid transfer operators or coarse grid operators [18, 17]. This leaves two classes of unstructured multigrid methods: rigid body mode based algebraic multigrid and non-nested geometric multigrid [2, 24].

We investigate the rigid body mode based algebraic method "smoothed aggregation" by Vanek, Mandel and Brezina as it has superior theoretical convergence characteristics over plain aggregation methods [30]. We test both smoothed and plain aggregation in §6. These methods require the kernel of the operator without Dirichlet boundary conditions for the fine grid; these are the rigid body modes for elasticity problems.

We are aware of two non-nested geometric methods: hybrid (non-nested linear to trilinear piecewise continuous coarse grid spaces) and remeshing methods (non-nested linear piecewise continuous coarse grid spaces). The *hybrid* (or incremental, or annealing) approach to coarsening grids is a promising method [24], as is the *remeshing* method [19, 11, 2]. Both methods automatically construct coarse grids from fine grids, use standard finite element shape functions to construct the grid transfer operators and construct the coarse grid operators algebraically. Both of these methods require the fine grid mesh (ie, element connectivities and nodal coordinates) be provided. This investigation uses the remeshing method as described in [2] as a representative non-nested geometric method.

### 1.1.1 Complexity theory

This section introduces the results of multigrid convergence theory that can provide some insight into the expected performance of our three methods on the symmetric positive definite matrices that arise from finite element discretizations of elliptic problems. Optimal convergence results are well known for (nested) geometric multigrid from Brandt, Douglas and others [7, 14], and algebraic multigrid with M matrices from Brandt, Ruge and Stübin and others [8, 26]. These early algebraic results assume M matrices which we do not have in elasticity; nested geometric multigrid theory is not directly applicable as we can not dependably constructed nested coarse grids.

The theories developed by Hackbusch and others and by Bramble, Pasciak and Xu can be applied to our three unstructured multigrid methods to provide some indication of the performance that we can hope to

achieve [20, 6].

- Smoothed aggregation multigrid preconditioning can guarantee a condition number of $O(\log^3(n))$ for 3D elasticity and $O(\log(n))$ with some assumptions about the problem [31, 30]. This is not optimal but it is close, it is only an upper bound, but our experiments in §6 indicate that in practice we can expect $O(1)$ iterations on elasticity problems.

- Condition number results for plain aggregation is more than polylogarithmic in $n$ for 3D problems: $O(n^{\frac{1}{3}}\log(n))$ [9]. This is far from optimal and our experiments in §6.8 (Figure 18 left) do indeed show some indication of this super-polylogarithmic complexity.

- Zhang extended the work of Bramble in developing multigrid algorithms with optimal theoretical results for non-nested non-quasi-uniform meshes coarse grids with some modified multigrid algorithms [33, 34]. Our construction of coarse grids can not guarantee some of the assumptions of these theories so these results can only provide some indication of the performance that we can expect.

## 2  Multigrid introduction

This section provides a brief introduction to multigrid, defining terms and providing comments on the structure of multigrid relevant to its implementation on high performance (ie, parallel) computers for unstructured grid 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, damped Jacobi and block Jacobi, are effective at reducing the high frequency error, but are ineffectual in reducing the low frequency content of the error [12]. 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 way that the finite element method 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) the grid transfer operators (ie, the *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 in this study 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 on the fine grid of the coarse grid function space of grid $i + 1$).

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

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

Figure 1: Multigrid *V-cycle* Algorithm

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 the multigrid methods considered in this study 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 because 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. This structure has the advantage that the job of the solver is decoupled into two distinct parts: 1) the multigrid method and 2) one or more of the many smoothers that have been developed as one level iterative solvers (eg, Gauss-Seidel, conjugate gradients preconditioned with block Jacobi, overlapping Schwarz, incomplete factorizations, approximate inverses ). 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 often only the construction of the restriction operators need be replaced.

# 3   Remeshing multigrid

The remeshing multigrid method was first proposed by Guillard and independently by Chan and Smith, and latter extended to 3D and in parallel by Adams [19, 11, 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 functions to construct the grid transfer operators. 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 sets (MISs) to produce a subset of vertices.

2. The new node set is then automatically remeshed with tetrahedra with a standard Delaunay algorithm.

3. Standard linear finite element shape functions, $N_I(x)$, are used to produce the restriction operator $R$. The weight used to transfer a fine grid residual for node $j$ at the coordinate $j.x$ to the coarse grid node $I$ is given by $R_{Ij} = N_I(j.x) \cdot I_{d \times d}$ where $I_{d \times d}$ is a $d$ by $d$ identity matrix, $d$ being the number of degrees of freedom per node.

4. The restriction operator is then used to construct the algebraic coarse grid operator from the fine grid operator: $A_{coarse} \leftarrow RA_{fine}R^T$.

   This non-nested geometric methods is not able to guarantee all of the assumptions that are used in the theoretical results of §1.1.1 and still maintain a minimal interface with the application (eg, coarse grid construction may not be as automatic as one would like, see §6.2 for an example). To try to minimize this difficulty and improve convergence rates in general we employ a series of optimizations to the MIS algorithm to preserve the geometric representation of the fine mesh, see Adams for details [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 an 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]. Note, the aggregation methods, discussed in the next section, also modify the matrix graph for use in an MIS algorithm to aggregate (or partition) the fine grid vertices.

# 4   Smoothed and plain aggregation multigrid

Unlike methods that use geometric coarse grids spaces, the aggregation methods do not construct explicit coarse grids. The main advantage of aggregation methods is that they need not construct coarse grid finite element meshes but only need to partition graphs and use the operator. 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.

   The *rigid body modes* (constant translations in, and rotations about, the three coordinate axis) can, in principle, be constructed from the stiffness matrix (without Dirichlet boundary conditions imposed) [15]. However, these spaces are easily constructed from the nodal coordinates for elasticity problems. Many rigid body mode methods have been developed [10, 15]; the addition of smoothing can provides significant improvement to these methods [29, 23], especially on more challenging problems. These aggregates are constructed so that the nodes within an aggregate are "strongly connected" (see §4.1 and [29]).

   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 [32]. 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 probably a good basis even if there is no kernel for the operator (as with dynamic problems with a mass matrix added into the stiffness matrix). These rigid body modes are used exclusively for elasticity problems and thus an $n$ (with $n$ degrees of freedom on the fine grid) by 6 block vector $B_0$ is constructed with the nodal coordinates 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: $B_i^j \rightarrow \bar{P}_i^j B_{i+1}^J$

4. $B_{i+1}^J$ is the kernel for coarse grid node $J$ 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 [23] 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 memory 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 [29], and 2) graph partitioners with some post-processing to remove tiny aggregates as discussed in §4.1.1. Vanek recommend that aggregates should be "connected by a path of strong coupling" as a means of automatically achieving semi-coarsening (semi-coarsening is effective for anisotropic problems such as some fluid flow problems [13, 20]). 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 $j$ - $d$ 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 $\epsilon$; 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 [29]. Note, this is similar in spirit to the heuristics used in the non-nested 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 on 3D problems (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 node 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 [22]. 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. 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 also useful as they are designed to provide load balancing which translates into having uniformly sized aggregates (which is desirable). Thus, 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 recommends 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 to date result in better solve times on any of our test problems; we have seen solve times of about 15% higher on the smallest version (run on one processor) of the concentric sphere problem in §6.8. 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 methods. The parallel finite element system is composed of a parallel finite element program built on a serial finite element code (FEAP [16]) and a parallel graph partitioner (ParMetis [22]) and our solver *Prometheus* (Prometheus is freely available in a publicly domain library [25]). Prometheus can be further decomposed into three parts: 1) general unstructured multigrid support built on PETSc [3]; 2) the non-nested geometric multigrid method; and 3) the aggregation multigrid methods.

The system reads a large "flat" finite element mesh input file in parallel, 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. Thus, these tests are solved with very general unstructured software so that, even if the problems are not very complex, the solver is not taking advantage of any of its underlying structure.

Explicit message passing (MPI) is used for performance and portability and all parts of the algorithm have been parallelized for scalability. 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. Prometheus assumes that the provide fine grid is partitioned well but does repartition the internally constructed coarse grids with ParMetis to maintain load balance.

As mentioned in §1.1.1 the parallel application of multigrid adds a $\log(n)$ term to the parallel complexity

as some processor must remain idle on the coarsest grids on very large problems. Given the number of degrees of freedom that we have per processor in our numerical experiments (about 27,000 degrees of freedom per processor in our scalability experiment that measures solve times on the Cray §6.8) and the number processors uses (640 Cray processors) the log term is not theoretically significant and most of the flops are done on grids that are "active" on all grids. We have found though that it is advantageous to reduce the number of processors on the coarsest grids when there are very few equations per processor (below about 100) particularly on machines with poor communication infrastructure. The reasons for this are two fold: 1) it is difficult to implement the parallel construction of the coarse grid spaces to have exactly the same semantics in the regions between processors as they do in serial and 2) the machines that we use are not modeled accurately with the PRAM complexity model (ie, the coarsest grids on large problems can actually run faster if fewer processors are used). Our solver implementation thus, reduce the number of active processors on the coarsest grids to try to keep a minimum of about 200 equations per processor.

# 6 Numerical results

This section investigates the performance of the three multigrid methods: non-nested geometric multigrid (NNGMG), plain aggregation and 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 [16, 28]. 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 along with their sizes and condition numbers.

| Test problem | Sect. | $n$ | Cond. | Procs. | thin | jumps | incomp. | scale | shell |
|--------------|-------|-----|-------|--------|------|-------|---------|-------|-------|
| Cone | §6.1 | 21,600 | $3.5 \cdot 10^7$ | 1 | $\sqrt{}$ | | | | |
| Beam-column | §6.2 | 34,460 | $1.0 \cdot 10^8$ | 4 | $\sqrt{}$ | | | | |
| Plate | §6.3 | 155,280 | $4.0 \cdot 10^{10}$ | 4 | | | | | $\sqrt{}$ |
| Wing | §6.4 | 617,760 | $1.0 \cdot 10^9$ | 16 | | | | | $\sqrt{}$ |
| Cylinder with cutouts | §6.5 | 135,318 | $1.0 \cdot 10^9$ | 4 | | | | | $\sqrt{}$ |
| Cantilever | §6.6 | 62,208 | $5 \cdot 10^{10}$ | 3 | | $\sqrt{}$ | $\sqrt{}$ | | |
| Sphere in a cube (Sp.) | §6.7 | 7,534,488 | $3 \cdot 10^8$ | 512 | | $\sqrt{}$ | $\sqrt{}$ | | |
| Concentric Sp. (C.Sp.) | §6.8 | 80K - 76M | NA | 1-1024 | $\sqrt{}$ | $\sqrt{}$ | $\sqrt{}$ | $\sqrt{}$ | |

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 using a direct subdomain solver. The Jacobi blocks (sub-domains) are constructed with METIS. Most problems are solved to a reduction in the residual of $10^{-6}$.

The value of these experiments is 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 methods with fully parallelized implementations 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 get 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 "illconditioned" 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 20% in convergence rate between runs with multiple processors for each of the three multigrid methods that we investigate on most of our test problems. 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 perhaps incur some performance penalty.

  The non-nested geometric method only meshes the coarse grids locally, hence the global coarse grid has overlapping regions and is not a valid finite element mesh. We do not see this as a major problem as the coarse grids need only approximate the low frequency error and these overlapping meshes do capture the rigid body modes. The local problems that can be formed by the parallel finite element code (especially on thin body problems like the cone below in §6.1) can effect the quality of the coarse grid spaces that are formed with our non-nested geometric method - resulting a different convergence rates (sometimes better and sometimes worse) as more processors are used.

  The aggregation method does not maintain the exact serial semantics in parallel in the interprocessor regions and we have noticed increases in number of iterations required for convergence of up to about 30% when multiple processors are used for both the aggregation methods on the smallest version of the concentric spheres problem in §6.8.

- We have worked with the non-nested geometric method for several years and the aggregation method for about a year; the aggregation method is much simpler (and is implemented with about half as much code) and we have done less algorithmic development on the aggregation method. We believe that we have optimized the aggregation 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" $R_{max}$ [21]).

## 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 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.
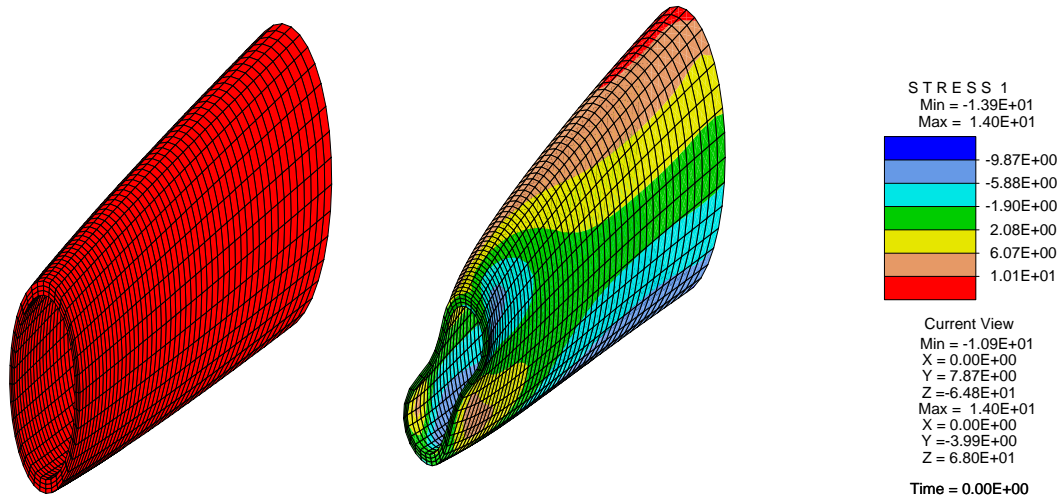


Figure 2: Truncated cone

Table 3 shows the sizes of the coarse grids for this problem along with the iteration counts.

| Solver | Smoothed aggregation (45) | | | Plain agg. (86) | | | NNGMG (54) | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 (iteration count)

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 NNGMG 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 grid takes 0.08 seconds on one PowerPC processor.
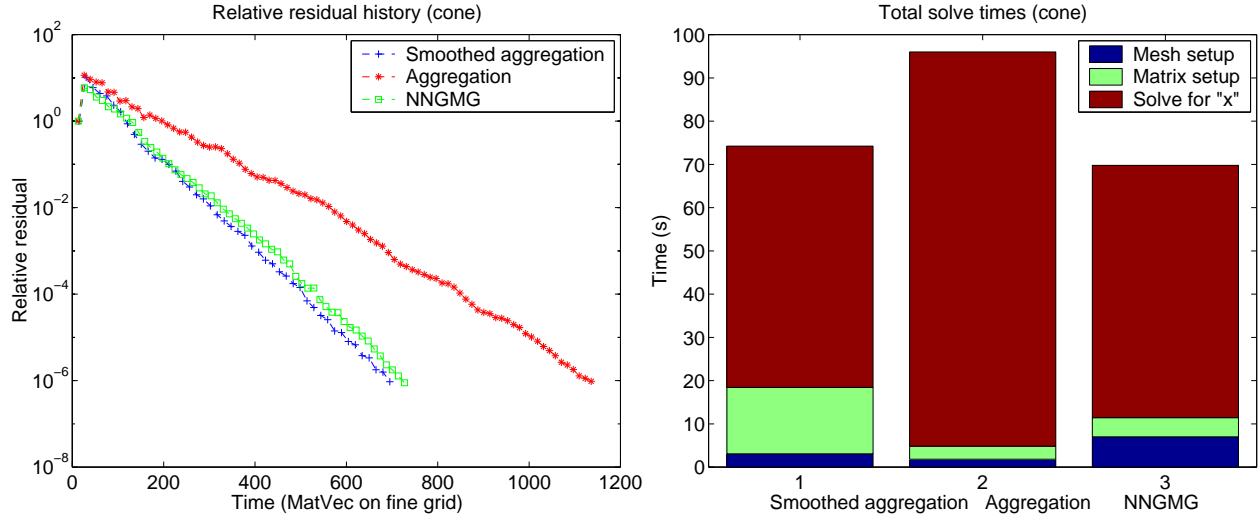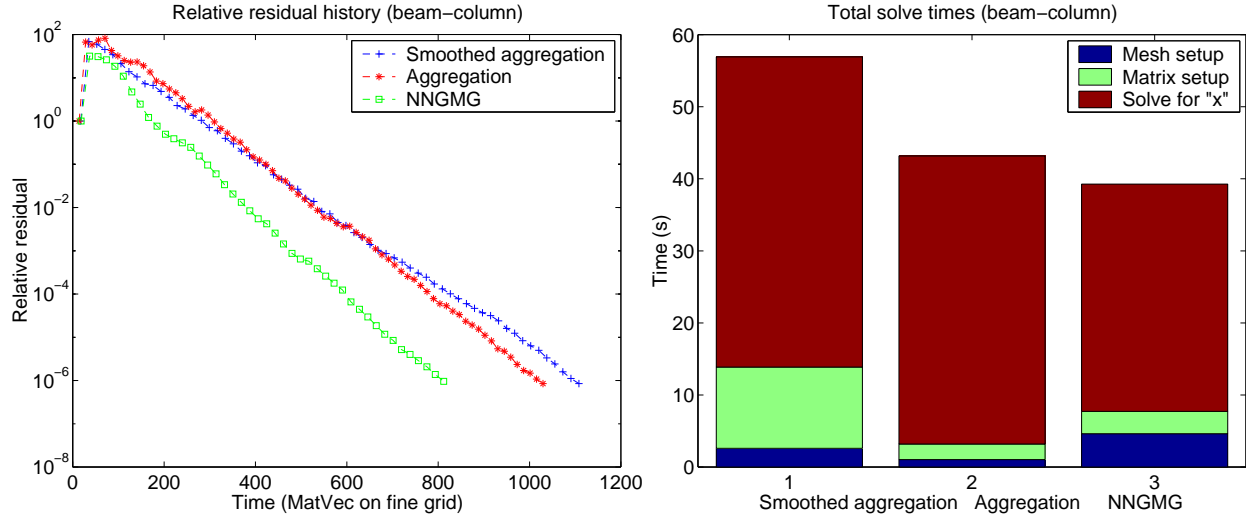
Figure 3: (left) Truncated cone residual history vs. solve times and (right) sum of mesh setup, matrix setup and solve times on one PowerPC processor

## 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 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.
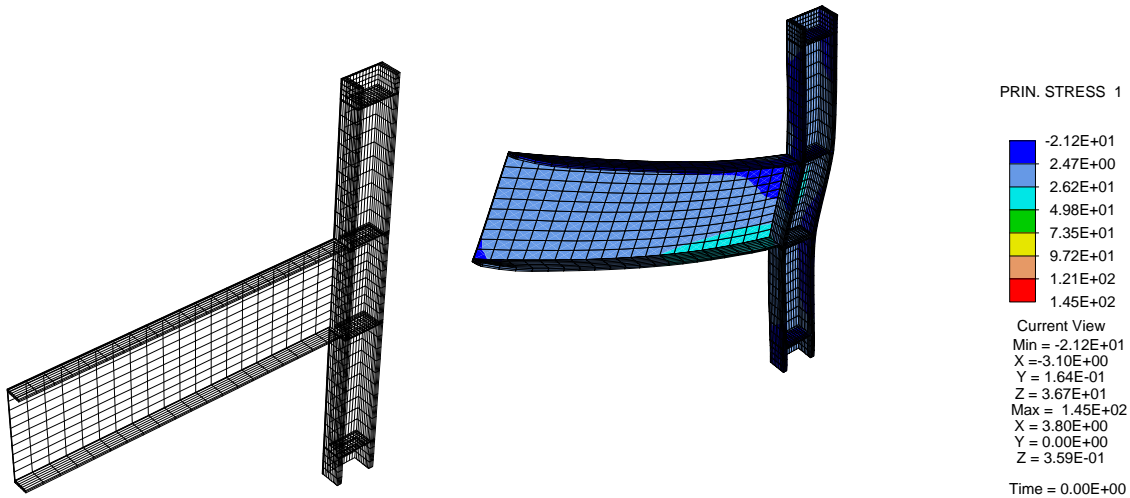


Figure 4: Beam Column

Table 4 shows the sizes of the coarse grids along with the iteration counts 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 NNGMG 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 grid takes 0.0394 seconds on four PowerPC processors.

11

| Solver | Smoothed aggregation (62) | | | Plain agg. (72) | | | NNGMG (43) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 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 (iteration count)


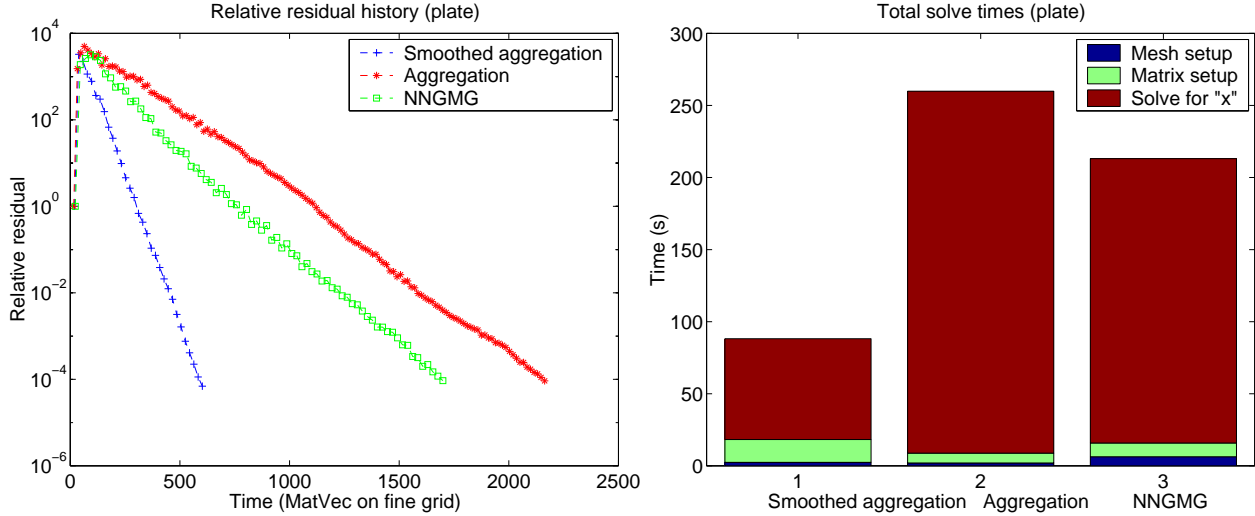
Figure 5: (left) Beam-column residual history vs. solve times and (right) sum of mesh setup, matrix setup and solve times on four PowerPC processors

## 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 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.
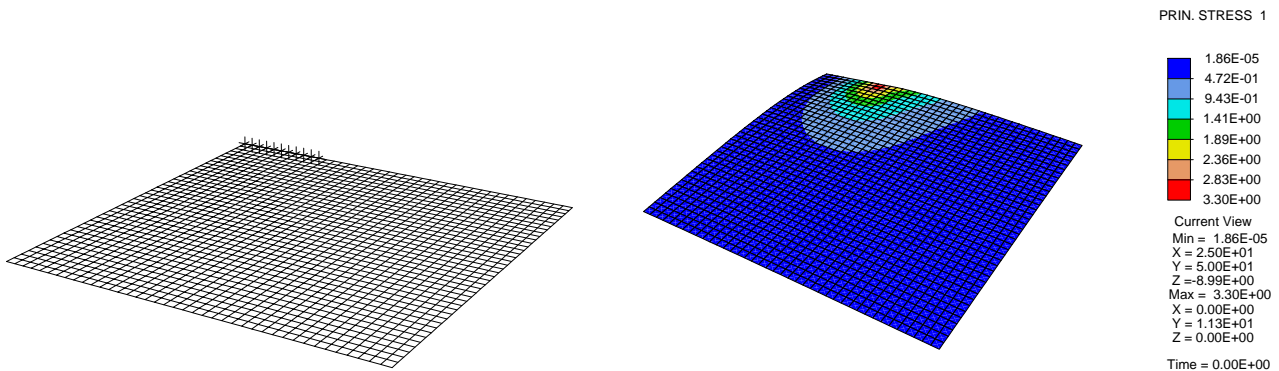


Figure 6: Flat plate

Table 5 shows the sizes of the coarse grids along with the iteration counts for this problem.

| Solver | Smoothed aggregation (30) | | | Plain agg. (134) | | | | NNGMG (73) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 (iteration count)

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 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.

Table 6 shows the sizes of the coarse grids along with the iteration counts for this problem.

| Solver | Smoothed aggregation (80) | | | | Plain agg. (199) | | | | NNGMG (93) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 (iteration count)

13

PRIN. STRESS 1

3.16E+00
2.60E+02
5.17E+02
7.75E+02
1.03E+03
1.29E+03
1.55E+03
1.80E+03

Current View
Min = 3.16E+00
X = 5.00E+01
Y = 2.98E+00
Z = -6.23E+00
Max = 1.80E+03
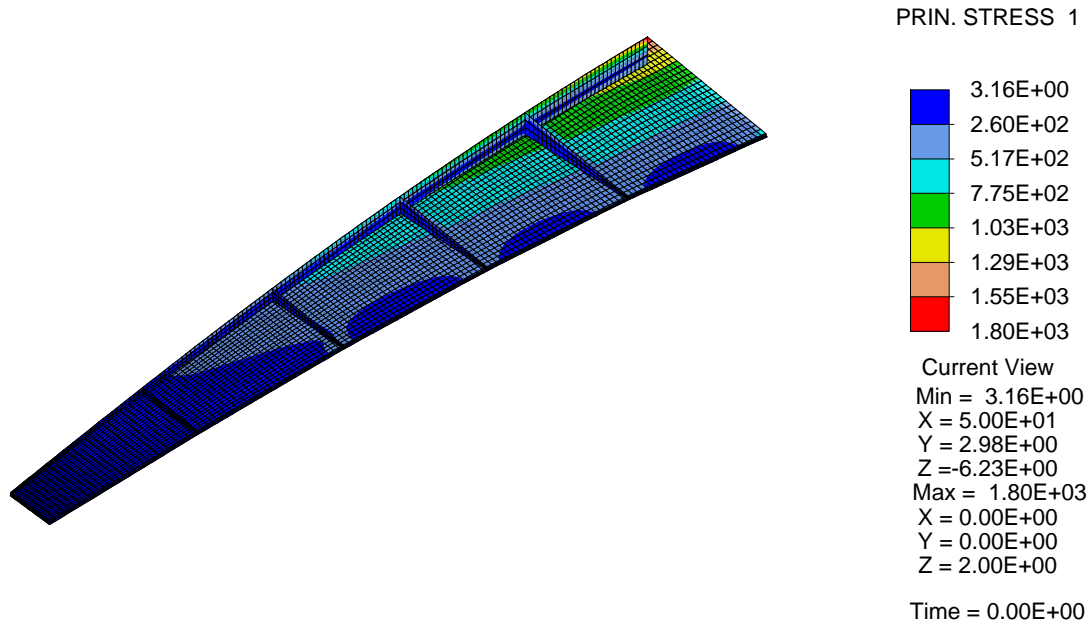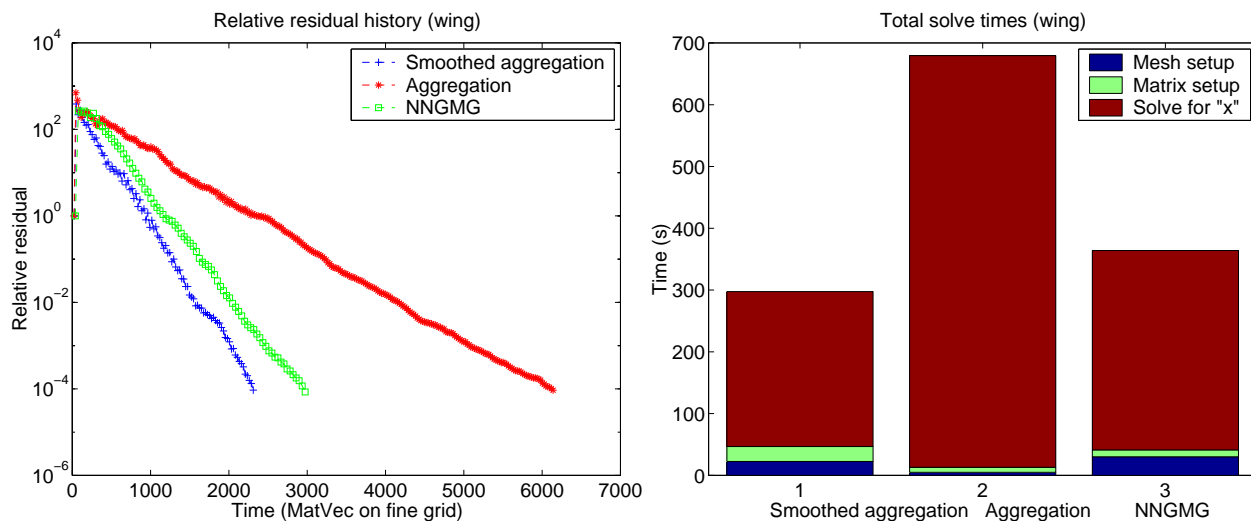X = 0.00E+00
Y = 0.00E+00
Z = 2.00E+00

Time = 0.00E+00

Figure 8: Simple wing (50K dof version)

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.



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

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.

14

## 6.5  Tube with holes

The next problem is a 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 Jacobi preconditioner for the PCG smoother. This problem has a condition number of over $1.0 \cdot 10^9$. 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.
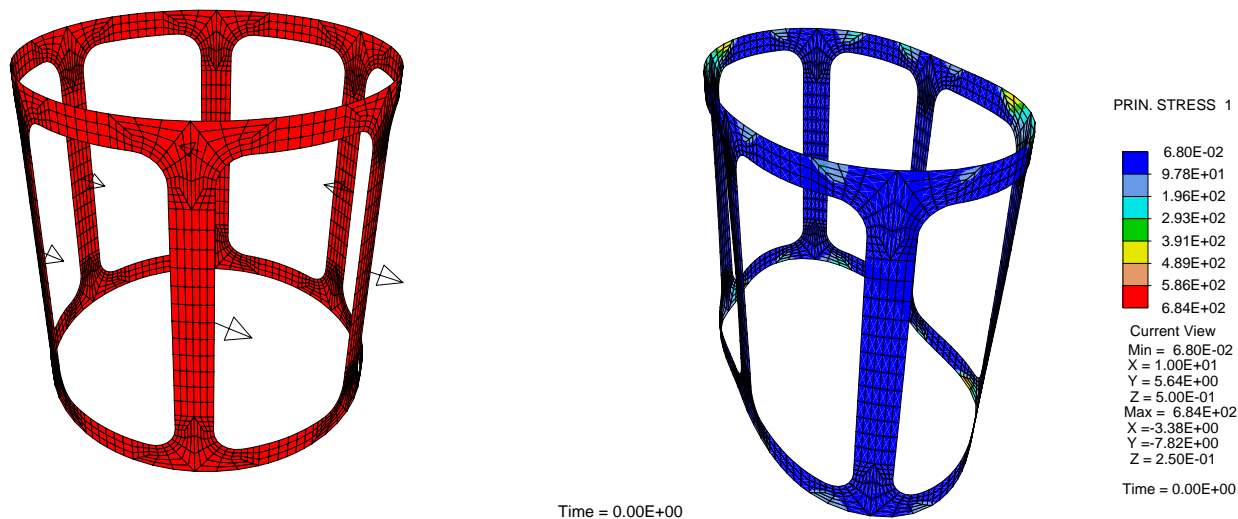


Figure 10: Tube with holes (13K dof version)

Table 7 shows the sizes of the coarse grids along with the iteration counts for this problem.

| Solver | Smoothed aggregation (132) | | | Plain agg. (127) | | | NNGMG (200+) | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 (iteration count)

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.

This data shows that smoothed aggregation multigrid is the fastest and that the NNGMG method is the slowest and essentially fails on this problem. 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 NNGMG 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 Jacobi preconditioner for the PCG smoother. This problem has a condition number of over $5 \cdot 10^{10}$.

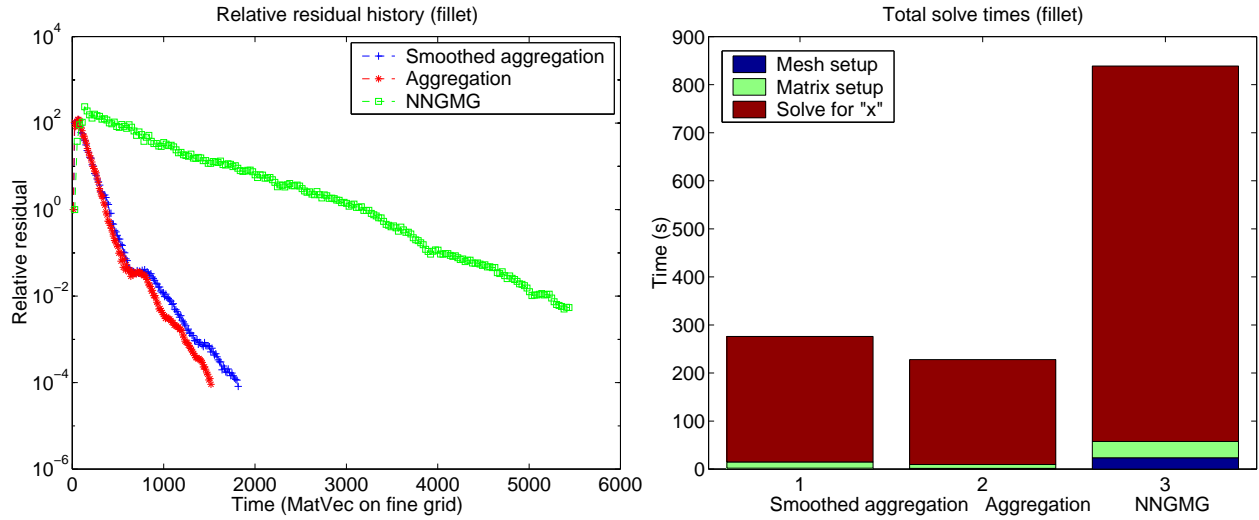Table 8 shows the sizes of the coarse grids for along with the iteration counts this problem.

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
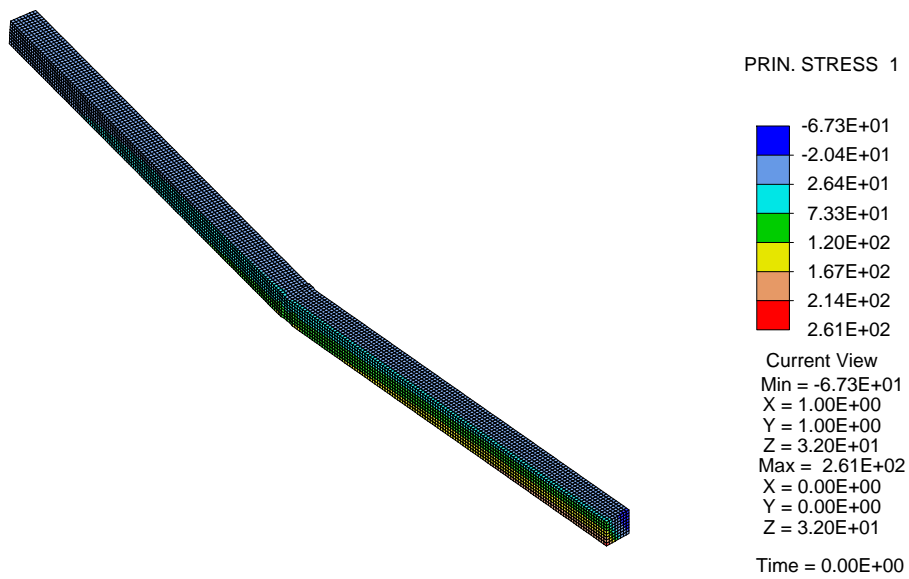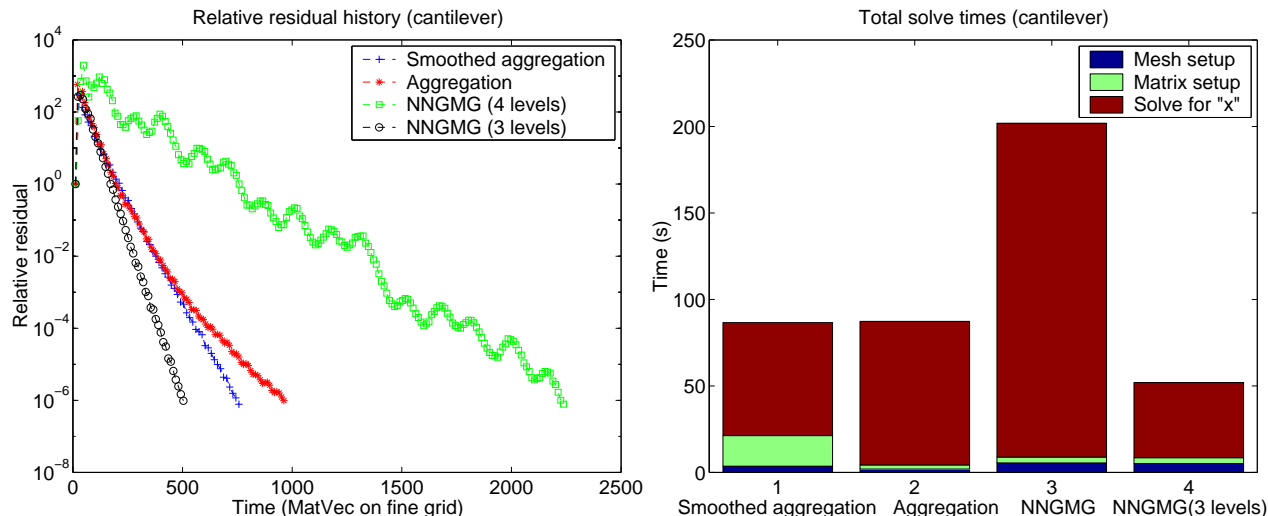


Figure 12: Soft section cantilever

| Solver | Smoothed agg. (53) | | | Plain agg. (88) | | | NNGMG - 4 levels (185) | | | | NNGMG - 3 levels (43) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 (iteration count)

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 NNGMG method on pathological problems. It is well know that

16

if the coarse grids can capture the material discontinuities on all of the coarse meshes that the convergence rate of NNGMG multigrid will not deteriorate with large jumps in material coefficients [27]. The NNGMG method can not guarantee that important 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 NNGMG 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 ("NNGMG (3 levels)" in Figure 13).



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 NNGMG multigrid method is the fastest by a small margin and the aggregation method is the slowest, but that the NNGMG method is not robust on this problem. 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 7.5 million 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 Jacobi preconditioner for the PCG smoother. This problem has a condition number of about $3 \cdot 10^8$.

Table 9 shows the sizes of the coarse grids along with the iteration counts for this problem.

| Solver | Smoothed agg. (68) | | | | Plain agg. (120) | | | | | NNGMG (55) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 (iteration count)

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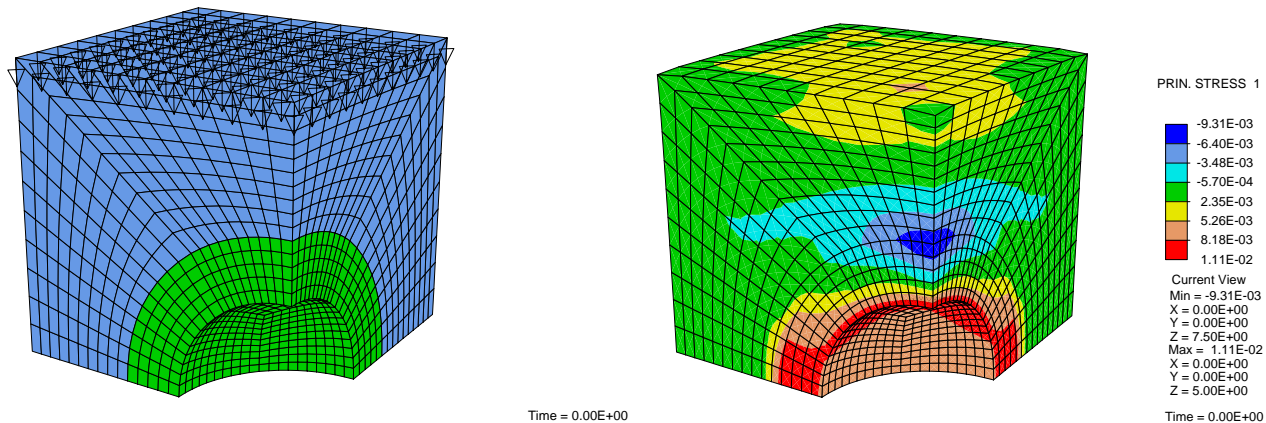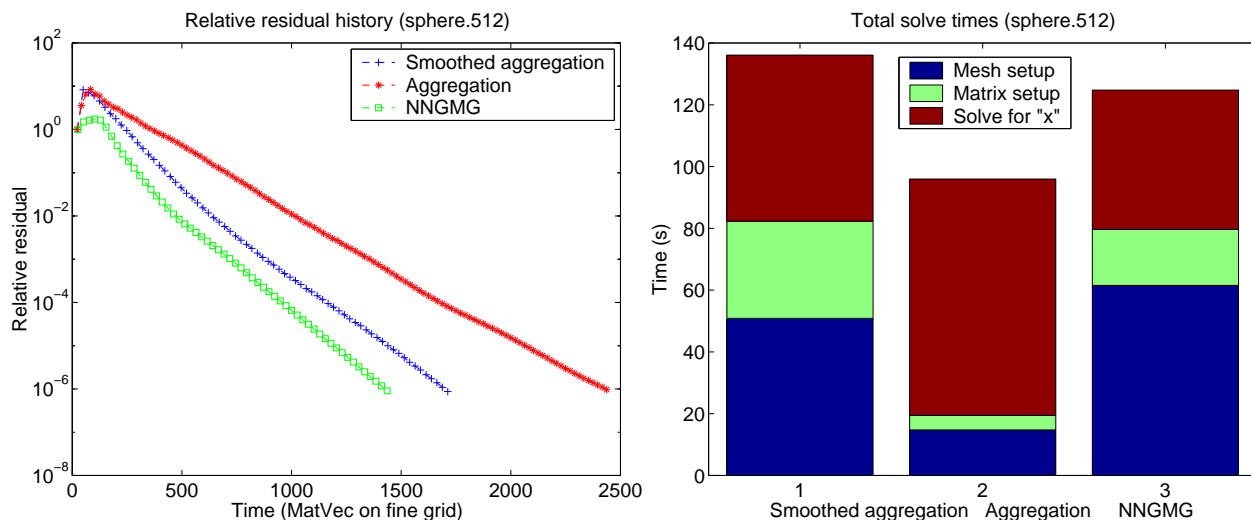
Figure 14: Sphere in a cube (15K dof version)



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 NNGMG 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 shells 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.

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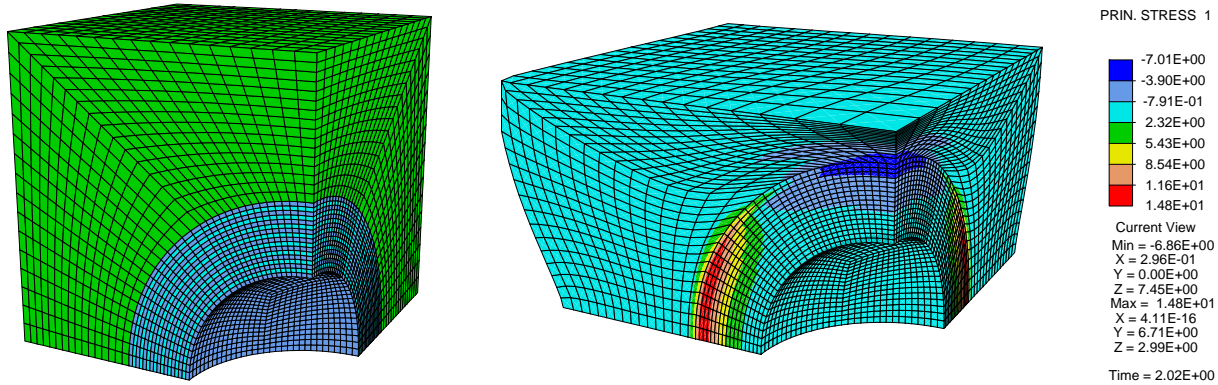
Figure 16: Concentric spheres in a cube (80K dof version)

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).
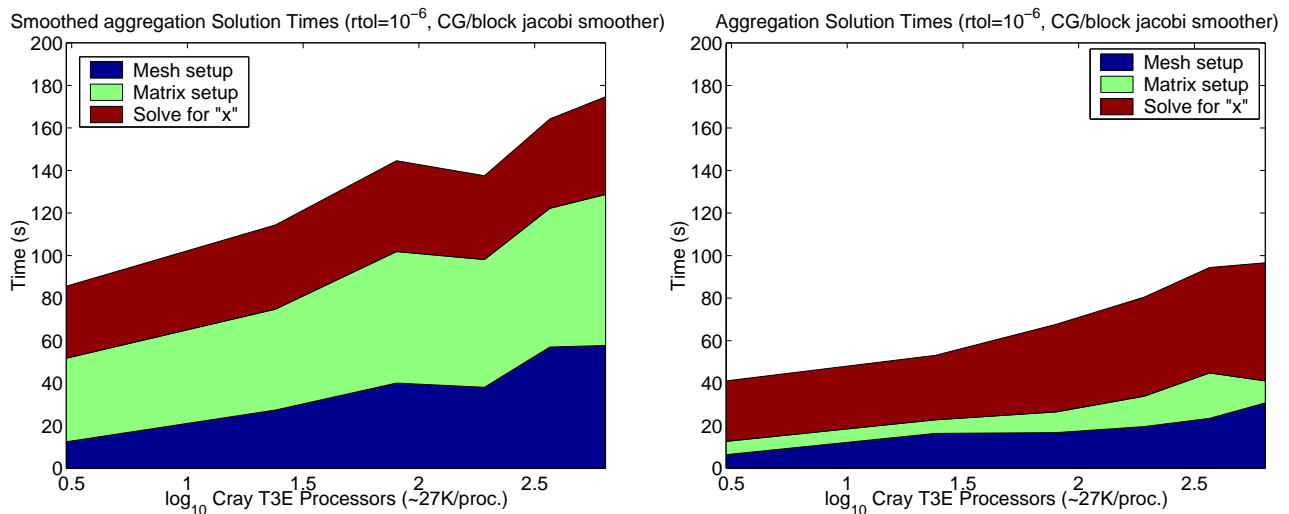


Figure 17: Concentric spheres in a cube, sum of mesh setup, matrix setup and solve times for smoothed aggregation (left) and plain aggregation (right), vs. $\log_{10}$ of number of Cray T3E (from 3 to 640) processors

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 NNGMG 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. These problems are run with a $\lfloor \frac{n}{128} \rfloor$ block Jacobi preconditioner for the PCG smoother. Figure 18 (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 NNGMG method is decreasing, the smoothed aggregation method iterations are about constant, and the plain aggregation method iterations are increasing. This data reflects what the theory predicts - namely the convergence rate (both asymptotic and total) is decreasing for the plain aggregation method, staying relatively steady for the smoothed aggregation method and increasing for the NNGMG method.

Figure 20 (right) and Table 10 show the average number of non-zeros per row of the stiffness matrices, along with the iteration counts, for each solver on the largest problem. From this data we can see a large
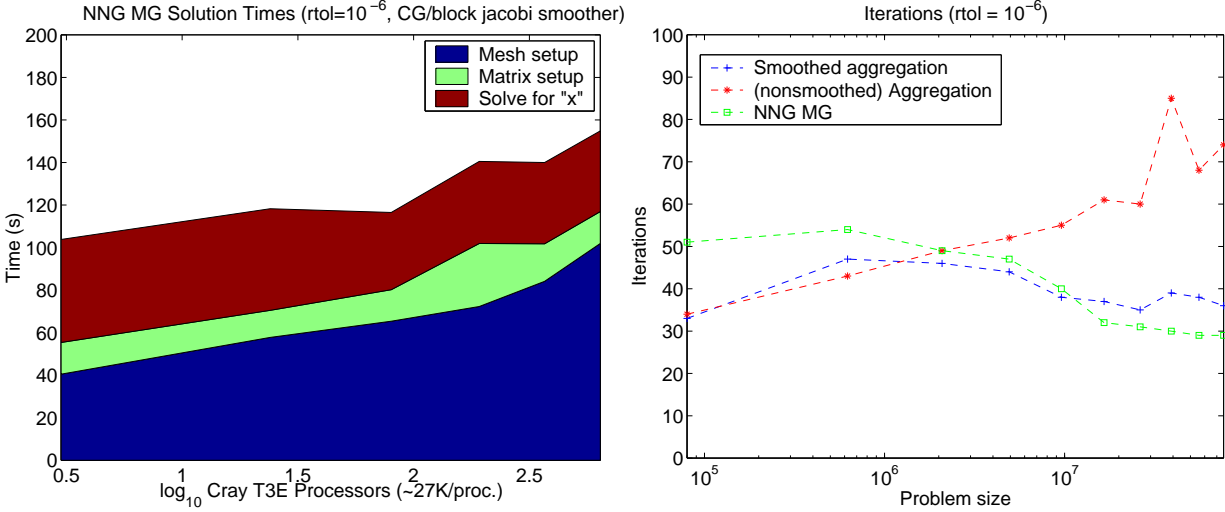
Figure 18: (left) Concentric spheres in a cube, sum of mesh setup, matrix setup and solve times vs. $\log_{10}$ of number of Cray T3E (from 3 to 640) processors for the NNGMG method and (right) iteration counts from IBM with up to 76M dof vs. $\log_{10}$ of number of PowerPC processors
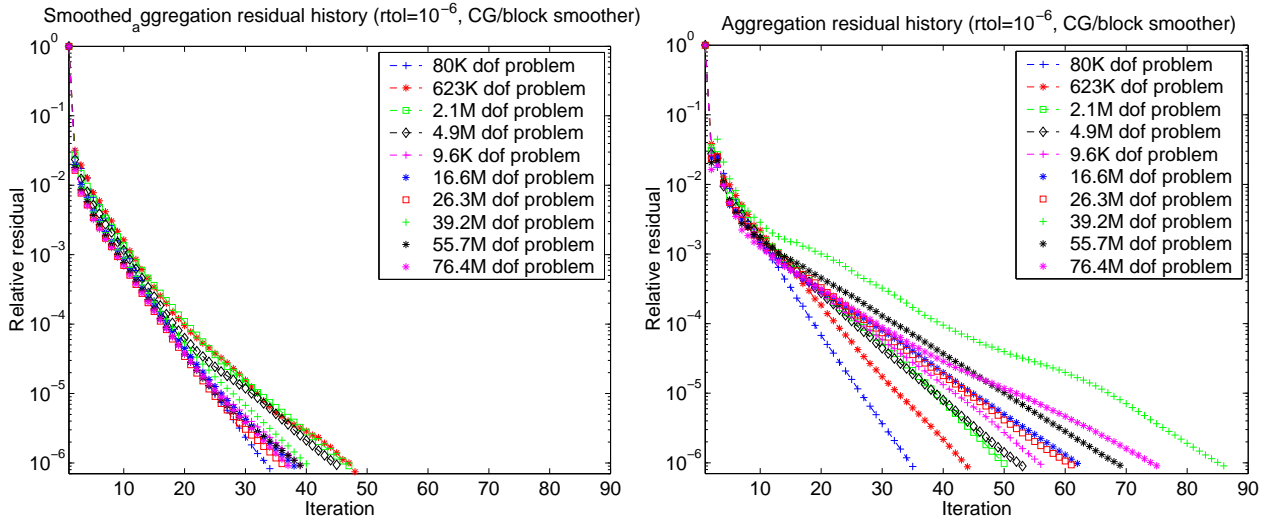


Figure 19: Concentric spheres in a cube residual histories, vs. iterations, on one to 1024 PowerPC processors for smoothed aggregation (left) and plain aggregation (right)

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.

# 7   Conclusion

We have shown that the three multigrid methods 1) smoothed aggregation, 2) plain aggregation and 3) non-nested geometric multigrid, are effective to varying degrees on our test problems in 3D solid mechanics.
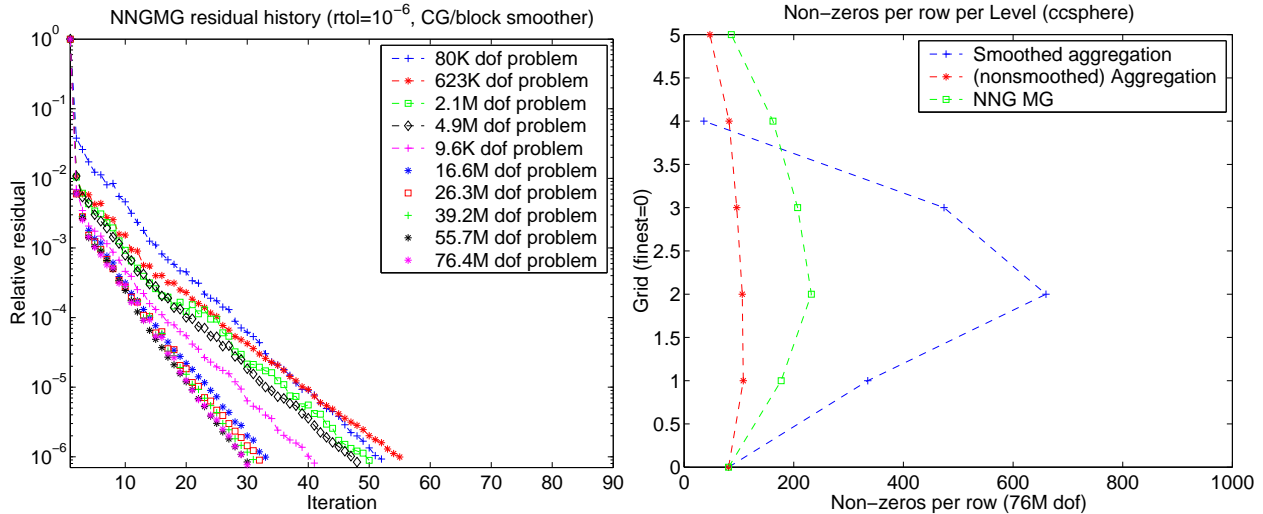
Figure 20: (left) Concentric spheres in a cube residual histories, vs. iterations, on one to 1024 PowerPC processors for NNGMG method and (right) average non-zeros per row for all grids of largest problem

| Solver | Smoothed agg. (36) | | | | | Plain agg. (74) | | | | | | NNGMG (29) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Grid | 0 | 1 | 2 | 3 | 4 | 0 | 1 | 2 | 3 | 4 | 5 | 0 | 1 | 2 | 3 | 4 | 5 |
| Eeqs. | 76M | 4.9M | 124K | 1.7K | 36 | 76M | 4.9M | 310K | 22K | 1.8K | 156 | 76M | 6.4M | 295K | 13K | 990 | 96 |
| nz/eq. | 81 | 335 | 660 | 474 | 36 | 81 | 108 | 106 | 96 | 82 | 47 | 81 | 177 | 232 | 207 | 167 | 86 |

Table 10: Grid sizes for concentric spheres (iteration count on the 76M dof problem)

We have also introduced a new method for constructing the aggregates for the aggregation 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 often 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 non-nested geometric multigrid method has been shown to be effective on continuum element problems with most of the smallest solve times, but with the highest setup costs of the three methods. However, the non-nested geometric method has 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. The only drawback of the smoothed aggregation method is the relatively large times for the algebraic construction of the coarse grids.

For the solve times, non-nested geometric multigrid was the fastest in four of our eight test problems, smoothed aggregation was the fastest in three and plain aggregation was fastest in one. For one complete linear solve, the non-nested 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.

We conclude that for problems that are not very large or if only one solve is to be run for each mesh configuration than the plain aggregation method can be superior. For continuum element problems in which many solves are to be run for each mesh configuration the non-nested geometric if often superior, with the caveat that it is less robust than the other two methods. For shell problems, or problems with multiple solves for each stiffness matrix, the smoothed aggregation method is superior. Additionally the smoothed aggregation method always seem to be close to optimal, thus, we conclude that the overall superior general purpose method for 3D elasticity problems is smoothed aggregation.

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. More work can also be done in tracking down the precise sources of the performance variations that we see in some parallel cases.

# 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(8):1241–1262, 2000.

[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] A. Brandt. Multi-level adaptive solutions to boundary value problems. *Math. Comput.*, 31:333–390, 1977.

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

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

[10] 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.

[11] 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.

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

[13] 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.

[14] C. C. Douglas. Multi–grid algorithms with applications to elliptic boundary–value problems. *SIAM J. Numer. Anal.*, 21:236–254, 1984.

[15] 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.

[16] FEAP. www.ce.berkeley.edu/~rlt.

[17] 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.

[18] 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.

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

[20] W. Hackbusch. *Multi-grid Methods and Applications*. Springer-Verlag, Berlin, 1985.

[21] IBM. www.rs6000.ibm.com/hardware/largescale.

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

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

[24] 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.

[25] Prometheus. www.cs.berkeley.edu/~madams.

[26] J. Ruge and Stübin K. *Multigrid methods*, chapter 4, pages 73–130. SAIM, 1987.

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

[28] 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.

[29] 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.

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

[31] 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.

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

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

[34] 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.