Parallel Smoothed Aggregation Multigrid : Aggregation Strategies on Massively Parallel Machines

Ray S. Tuminaro[1]
Sandia National Laboratories

and

Charles Tong[2]
Lawrence Livermore National Laboratory

Abstract

Algebraic multigrid methods offer the hope that multigrid convergence can be achieved (for at least some important applications) without a great deal of effort from engineers and scientists wishing to solve linear systems. In this paper we consider parallelization of the smoothed aggregation multigrid method. Smoothed aggregation is one of the most promising algebraic multigrid methods. Therefore, developing parallel variants with both good convergence and efficiency properties is of great importance. However, parallelization is nontrivial due to the somewhat sequential aggregation (or grid coarsening) phase. In this paper, we discuss three different parallel aggregation algorithms and illustrate the advantages and disadvantages of each variant in terms of parallelism and convergence. Numerical results will be shown on the Intel Teraflop computer for some large problems coming from nontrivial codes: quasi-static electric potential simulation and a fluid flow calculation.

[1]Supported by the Applied Mathematical Sciences program, U.S. Department of Energy, Office of Energy Research, and was performed at Sandia National Laboratories, operated for the U.S. Department of Energy under contract No. DE-AC04-94AL85000.

[2]This work was performed under the auspices of the US Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

0-7803-9802-5/2000/$10.00 (c) 2000 IEEE.

1 Introduction

Multilevel methods offer the best promise to balance fast convergence and parallel efficiency in the numerical solution of elliptic partial differential equations. Multigrid methods are scalable and relatively suitable for parallel implementation. The idea of multigrid is to capture errors spanning a range of spaces via several levels of fineness. By traversing the levels, optimal convergence rates are frequently observed (independent of the number of mesh points) with overall solution times that are much faster than other methods.

For a large set of applications, algebraic multigrid methods provide many of the convergence/efficiency benefits of multigrid without a large time investment from application engineers. Most algebraic methods automatically construct coarse grids and grid transfer operators using a a modest amount of application information. In this paper we consider the smoothed aggregation multigrid method [11]. Smoothed aggregation is one of the more promising algebraic multigrid methods. Various forms of aggregation have been used on different applications [8] including some very difficult elasticity simulations that are notoriously difficult for iterative methods [13] [1]. To our knowledge the only other massively parallel smoothed aggregation work is being developed concurrently by [1]. There is also work in parallelizing classical algebraic multigrid[7]. There are many similarities in parallelizing the coarsening phase of classical multigrid and parallelizing the aggregation phase of smoothed aggregation. However, the specific coarsening algorithms and convergence behavior are different.

In this paper we consider the development of variants of smoothed aggregation suitable for massively parallel computer architectures. While most smoothed aggregation algorithm stages parallelize easily, the aggregation (or coarsening) phase is more problematic. In particular, the original coarsening algorithm progresses by making each aggregate one after another (updating information used to determine the next aggregate). In this paper, we present three parallel aggregation schemes. In one variant coarsening occurs in each processor independently. While highly parallel, this variant may coarsen somewhat irregularly and is constrained by the number of processors and the partitioning of the original data. The second algorithm tries to rectify this by taking into account inter-processor matrix coupling. Specifically, it coarsens near inter-processor boundaries first (requiring some processors to wait for other processors). Once a processor's inter-processor boundaries are aggregated, it may coarsen the interior independently. The third variant is

based on parallel maximally independent sets (MIS). In this MIS aggregation all processors coarsen in parallel. There are, however, some restrictions near processor boundaries (i.e. processors may need to wait for other processors before acting on inter-processor boundary points). While the decoupled variant often performs satisfactorily, accounting for inter-processor coupling is sometimes needed on complex applications in order to avoid deteriorating convergence or increasing execution time. Numerical results will be shown on some problems coming from nontrivial simulation codes on the Intel Teraflop computer.

2 Smoothed Aggregation Multigrid Method

We begin with a brief multigrid description (see [4], [5], or [6] for more information). A multigrid solver tries to approximate the original PDE problem of interest on a hierarchy of grids and use 'solutions' from coarse grids to accelerate the convergence on the finest grid. A simple multilevel iteration is illustrated below.

```
/* Solve A_k u = b (where k is the current grid level)  */
procedure multilevel(A_k, b, u, k)
        u = S_k(A_k, b, u);
        if ( k ≠ 1)
            P_k = determine_interpolant( A_k );
            r̂ = P_k^T (b − A_k u) ;
            Â_{k−1} = P_k^T A_k P_k;   v = 0;
            multilevel(Â_{k−1}, r̂, v, k − 1);
            u = u + P_k v;
        end if
end procedure
```

In the above method, the $S_k()$'s are approximate solvers (or more popularly called smoothers) and usually correspond to a basic iterative method such as Gauss-Seidel. The P_k's (interpolation operators that transfer solutions from coarse grids to finer grids) are the key ingredients that must be determined automatically within an algebraic multigrid method. [3]

[3]The P_k's are usually determined as a preprocessing step and not computed within the iteration.

We now outline the construction of the P_k's in the context of the smoothed aggregation multigrid method. In our description we use the notation 'grid points' in the aggregation process. While this term is more intuitive, it is generally more accurate to refer to graph vertices. A graph corresponding to a symmetric matrix is constructed by creating a vertex for each matrix row and adding a graph edge between vertices i and j if there is a matrix nonzero in the $(i,j)^{th}$ element. It is important to note that there are several important enhancements/modifications that should be made when constructing this graph. For example, if a nonzero value is relatively small, this edge should not be added to the graph. This is critical for anisotropic problems where it is best not to coarsen in directions of weak coupling. Furthermore, for PDE systems it is often advantageous to build a graph corresponding to the block matrix (grouping into blocks all unknowns at a grid point) as opposed to treating each degree of freedom as a separate vertex [12].

The smoothed aggregation P_k's are determined in two steps : coarsening and grid transfer construction. The first step is to take each grid point and assign it to an aggregate. This step is discussed in detail at the end of this section. For now, we state that on 3D Poisson problems discretized on a uniform grid with 27-point stencil, each aggregate consists of approximately 30 spatially close grid points. For simplicity of exposition the second step is described below for a simple Poisson equation (though our algorithms/code have been applied to more general PDE systems). A more detailed and general discussion can be found in [13]. This second step consists of first forming a tentative prolongator matrix $\tilde{P}_k$ and then applying a prolongator smoother $\tilde{S}_k$ to it giving rise to $P_k = \tilde{S}_k \tilde{P}_k$. The tentative prolongator matrix $\tilde{P}_k$ is constructed such that each row corresponds to a grid point and each column corresponds to an aggregate. The entries of $\tilde{P}_k$ are as follows (for specific applications such as elasticity problems, more complicated tentative prolongators can be derived based on rigid body motions) :

$$\tilde{P}_k(i,j) = \begin{cases} 1 & \text{if } i^{th} \text{ point is contained in } j^{th} \text{ aggregate} \\ 0 & \text{otherwise} \end{cases}$$

The tentative prolongator can be viewed as a simple grid transfer operator corresponding to piecewise constant interpolation. While $\tilde{P}_k$ can be used for P_k, a more robust method is realized by smoothing the piecewise constant basis functions. For example, applying a damped Jacobi smoother yields:

$$P_k = (I - \omega D_k^{-1} A_k)\tilde{P}_k \tag{1}$$

where choices for ω can be found in [13]. If ω and the aggregates are properly chosen, (1) leads to linear interpolation when applied to an one-dimensional Poisson problem. In general, however, (1) does not correspond to linear interpolation but yields better interpolation properties than piecewise constant interpolation. Proofs illustrating convergence mildly dependent on the number of mesh points can be found in [13]. This work is based on the convergence theory in [3].

We now return to the first step of determining the P_k's. The generation of a tentative prolongator requires that aggregates be defined. For isotropic problems using the above damped Jacobi smoother, the goal is to have fairly uniform shaped aggregate regions with a diameter of length three. Small aggregates (diameter less than three) lead to high iteration costs. This is because the number of unknowns on the next finest grid is equal to the number of aggregates and because the number of nonzeros per row in the coarse grid discrete operator depends on the distance between non-neighboring aggregates.[4] However, aggregates that are too large lead to grid transfer operators that look more like piecewise constant interpolation and give poorer multigrid convergence rates.

The basic aggregation procedure is given below.

Basic Aggregation

phase 1: repeat until all unaggregated points are adjacent to an aggregate
- *a)* pick root point not adjacent to any existing aggregate
- *b)* define new aggregate as root point plus all its neighbors

phase 2: sweep unaggregated points into existing aggregates or use them to form new aggregates

On a uniform grid, a snapshot of the above algorithm might look like Figure 1 where three aggregates have already been formed and the fourth aggregate is about to be created. After the fourth aggregate is finished only the grid point in the upper left corner can be chosen in *phase 1a*. After this fifth aggregate is created, the remaining unaggregated points must be handled in *phase 2* as they are all adjacent to existing aggregates. It is important to understand that the more points handled by *phase 2*, the more likely that resulting aggregates will be less uniform with diameters larger or small

[4]This is a function of the matrix-matrix multiply ($P_k^T A_k P_k$) operation. That is, it is possible to generate nonzeros in the coarse grid discretization matrix corresponding to two non-neighboring aggregates that are separated by a thin aggregate.

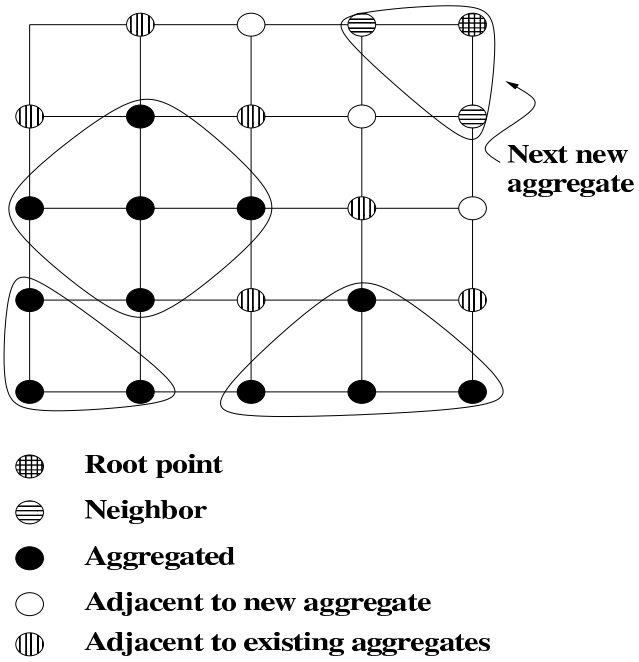

Figure 1: Snapshot of basic aggregation process.

than three. Thus, the key is to choose root points in *phase 1a* such that as many points as possible are aggregated. This means that we want to pack aggregates tightly. To pack efficiently, most schemes choose the next root point 'close' to existing aggregates (e.g. points with a distance three path from several already chosen root points) to avoid leaving many holes/points for *phase 2*. This can be seen in Figure 1 where a better choice for the next root point might have been the lowest point denoted by an empty circle. This would have allowed for an additional aggregate to be formed in *phase 1*. Unfortunately, it is this 'packing' feature which is difficult to attain in parallel.

3 Parallel Aggregation

Parallelization of most standard multigrid kernels needs to be done with care due to lower efficiencies associated with coarse grid processing, smoother complications due to the somewhat serial nature of the very popular/effective Gauss-Seidel method, and sparsity inefficiencies associated with the matrix-

matrix sparse multiplies needed to apply the damped Jacobi smoother (1) and to form the coarse grid discretization operator. These issues, however, are not the focus of this paper. We simply mention that we have implemented and incorporated smoothed aggregation within the ML package. This package requires users to furnish vectors and matrices. Matrices are supplied by providing size information, a matrix-vector product, and a *getrow* function (used to obtain nonzeros and column numbers within a single row). The ML package already contains V-cycle logic and many of the needed kernels : parallel matrix-matrix multiply, a variety of parallel smoothers (damped Jacobi, symmetric chaotic[5] Gauss-Seidel, block symmetric chaotic Gauss-Seidel, etc.) and a coarse direct solver. Additionally, the ML package is utilized by several different applications and facilitates the use of other software packages. A pseudo-code program fragment using smoothed aggregation is given in Figure 2.

The ML package has existing capabilities for handling general grid hierarchy issues to support various parallel geometric/algebraic multigrid methods. To incorporate smoothed aggregation into the existing ML framework, we needed to implement a new function that takes a discretization matrix and builds the tentative prolongator (using aggregation techniques). This new function is called by `ML_Gen_MGHierarchy_UsingAggregation` which takes advantage of the existing ML framework and ML's matrix algebra to create the smoothed prolongator as well as the complete multigrid hierarchy. More details concerning the ML package will appear in a paper [10].

The remainder of this section focuses on the parallelization of the basic aggregation scheme. In the following subsections we describe 3 parallel aggregation schemes.

3.1 Decoupled Parallel Aggregation Scheme

A simple parallel method is to let each processor aggregate its piece of the grid ignoring connections between processors. That is, each processor is assigned a subgrid of the entire grid (as in standard parallel grid calculations). Each processor then aggregates its own subgrid using the serial aggregation algorithm given in the last section. This variant, which we refer to as decoupled aggregation is quite easy to program and actually works fairly well in

[5]Chaotic means that each processor performs a local Gauss-Seidel and uses off-processor information corresponding to the previous iteration.

```
ML_Create(...); /* create ML context (or object) */

/* Furnish matrix (and associate with grid level: fine_level)*/
/* by providing data pointer (Adata), getrow function        */
/* (Agetrow) and matrix-vector multiply function (Amatvec).  */

ML_Init_Amat(fine_level, Adata, Nrows, ...);
ML_Set_Amat(fine_level, Amatvec, ... );
ML_Set_Amat_Getrow(fine_level,Agetrow, ...);

/* Create coarse grids and grid transfer matrices */
/* and use Gauss-Seidel as smoother on all grids  */

ML_Gen_MGHierarchy_UsingAggregation(ml, fine_level, ...);
ML_Gen_Smoother_GaussSeidel(ML_ALL_LEVELS, ...);

/* Perform multigrid V-cycle until convergence    */

ML_Iterate(rhs, initial_guess, ...);
ML_Destroy(); /* destroy ML context (or object)   */
```

Figure 2: Code fragment that invokes aggregation within ML package.

practice (its performance, however, depends a lot on the domain partitioner; that is, how the grid is partitioned among different processors). Its main disadvantages are that it can produce many aggregates near inter-processor boundaries that are either smaller or larger than an ideal aggregate (see for example Figure 3). Furthermore, no aggregate can span two processors and thus the overall aggregation process is limited by the processor grid. That is, we cannot have fewer aggregates than processors. This might imply that the coarsest grid is not so coarse. For example, a two dimensional problem on 1024 processors might have a 32×32 coarsest grid. Solving this coarsest grid problem using only a couple of iterations of a simple smoother can slow the overall multigrid convergence while doing many iterations can slow the time per iteration. On the other hand, performing a direct solver on this coarsest grid can require a significant amount of time especially when there are several unknowns at a grid point and when the number of nonzeros per row is

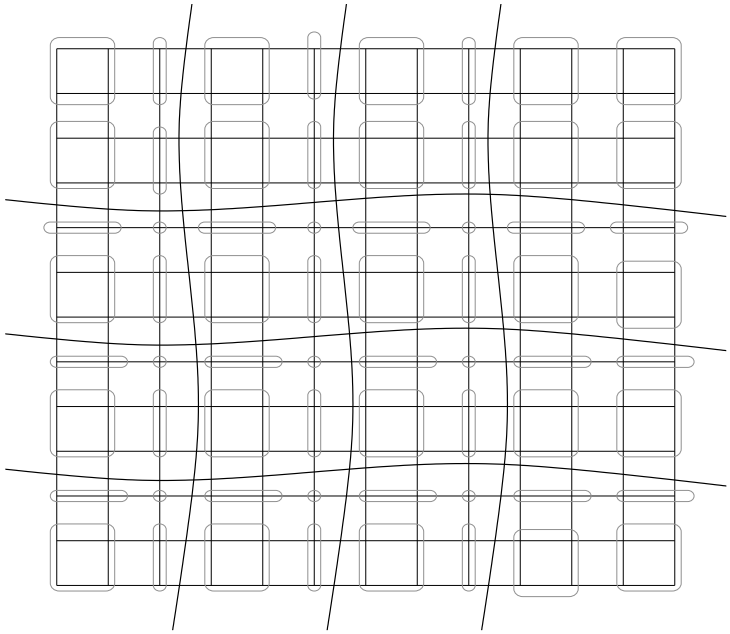

Figure 3: Illustration of decoupled aggregation where thick black lines delineate the processor grid and gray lines delineate the aggregates.

large (typically the case within most algebraic multigrid methods). For these reasons we explore two other aggregation variants : coupled aggregation and MIS aggregation.

3.2 Coupled Parallel Aggregation Scheme

Coupled aggregation proceeds in the following fashion. Each processor i splits all points assigned to it into two sets: interior (F_I^i) and border (F_B^i) points. Border points are those which share a grid edge with a point on another processor. Interior points are all points assigned to the processor that are not border points.

The coupled aggregation procedure is given below.

Coupled Parallel Aggregation

phase 1: repeat on each processor i until all unaggregated points in F_B^i are adjacent to some aggregate

a) pick root point in F_B^i not adjacent to any existing aggregate

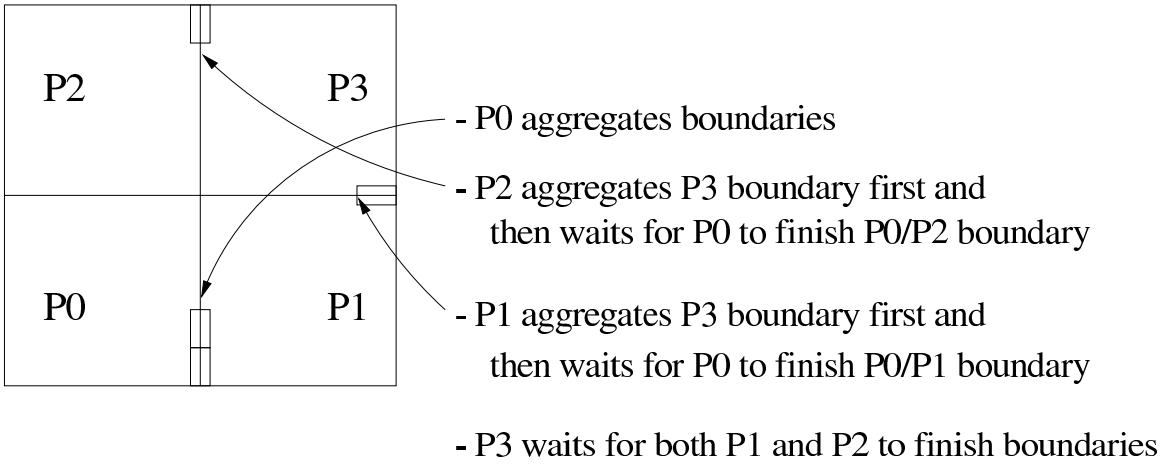

Figure 4: Illustration of coupled aggregation.

(local or remote) and not adjacent to any unconsidered
points in processor j ($\forall j < i$)
b) define new aggregate as root point plus all its neighbors

phase 2: aggregate points in F_I^i based on the serial aggregation procedure
phase 3: sweep unaggregated points (in F_B^i and F_I^i) into local aggregates
phase 4: form new local aggregates for the remaining points in F_B^i and F_I^i

In brief, this algorithm aggregates points near inter-processor boundaries before aggregating the interior (see Figure 4). It should be observed that some processors (especially higher numbered processors) may need to wait before it is possible to aggregate some border points. That is, a processor may need to wait until lower numbered processors have aggregated their adjacent border points. While this algorithm is somewhat more sequential, there is still a great deal of work that can be done in parallel and that the aggregation step is usually not the dominant part of the overall computation. Typically for a three dimensional problem, the worst case scenario would require the highest numbered processor to wait $\sqrt[3]{P}$ stages where P is the number of processors. However, while this scheme requires more time to compute, it can yield significantly better aggregates and does not limit the coarseness of grids to the number of processors.

3.3 MIS-based Parallel Aggregation Scheme

The third aggregation scheme is based on maximally independent sets (MIS) [9]. In brief, a variant of the coupled aggregation scheme can be realized by applying the widely studied parallel MIS algorithms on the square of the matrix representing the grid connectivities. In the following we explain the relationship between the requirements of the aggregation procedure and the MIS algorithm.

Again, the goal is to form an initial set of aggregates with as many points as possible. If the degree of each point (number of adjacent edges) is approximately the same, this is equivalent to finding as many aggregates as possible. The principal restriction is that a root point cannot be adjacent to an existing aggregate (i.e. two root points cannot have a path of length one or length two between them). Therefore, maximizing the number of initial aggregates is equivalent to finding the largest number of root points such that the distance between any two root points is at least three. We can take the original matrix graph and replace it by another graph where a new edge is added for each distance two path in the original graph. This new graph actually corresponds to squaring the matrix. Then, maximizing the number of initial aggregates is equivalent to finding an MIS of the graph corresponding to the square of the matrix. The MIS problem has been well studied and there are several different parallel variants. It is interesting to note that coarsening within classical multigrid has often been formulated in terms of MIS algorithms. In classical multigrid, coarsening is accomplished by labeling a subset of grid points as coarse points (those that will remain on the next coarser grid). These coarse grid points can be determined with the help of an MIS algorithm applied to the original matrix graph. In our MIS aggregation, the MIS scheme is actually applied to the square of the matrix and is used to generate root points from which aggregates are derived. Clearly the goal of coarsening within the two multigrid schemes is roughly the same. However, given the differences (coarsening at different rates and aggregating versus choosing a point subset), this direct connection via the MIS algorithm on either the original graph or the matrix square graph was quite surprising.

We implement our third aggregation scheme by replacing *phase 1* of the decoupled aggregation by an MIS algorithm on the square of the matrix (our current implementation actually squares the matrices using ML's sparse matrix-matrix multiply). The MIS algorithm computes all the root points

which are used to form aggregates. This step is followed by *phase 2* to aggregate any points that are missed in *phase 1*. For the parallel MIS method, we use the asynchronous distributed memory algorithm (ADMMA) by Adams. This method has some similarities to the coupled method. In particular, each processor chooses root points from among points that it owns. These root points can be chosen from anywhere (not restricted to being near inter-processor boundaries first). However, points near inter-processor boundaries cannot be chosen before certain nearby off-processor points are handled on lower numbered processors. Additionally, there is some logic to encourage root points being chosen close to existing root points. We omit the details of this method and refer the reader to [2]. It is important to note that actually computing the matrix square is not an efficient implementation of the above scheme. Clearly, this requires more storage and a fair amount of computation time to multiply two matrices. Another possibility (which we have not implemented) is to implicitly build a new matrix with a *getrow* function that uses the original matrix's *getrow* to implicitly replace distance two paths by direct edge connections.

4 Numerical Results

We first give results on both Poisson and anisotropic Poisson operators. Specifically, we consider

$$u_{xx} + u_{yy} + u_{zz} = f$$

and

$$u_{xx} + \epsilon u_{yy} = f$$

defined on the unit cube and unit square, respectively (with Dirichlet boundary conditions around the cube or square) and ϵ chosen as 10^{-5} for the anisotropic problem.

The two-dimensional unit square problems are partitioned over a two dimensional square processor array such that each x-axis grid line is distributed over $\sqrt{P}$ processors, and each y-axis grid line is distributed over $\sqrt{P}$ processors. The three-dimensional cube is partitioned over a two dimensional square processor array such that each x-axis grid line is distributed over $\sqrt{P}$ processors, each y-axis grid line is distributed over $\sqrt{P}$ processors, and each z-axis grid line is entirely owned by a processor. It is important to note that

N	procs	CG/MG			CG/symGS
		decoupled	coupled	MIS	
		15/4.5	29/ 8.8	15/5.4	106/17.0
113^3	64	1.5	1.4	1.7	
		21/9.1	47/18.7	10/5.1	138/33.2
155^3	121	1.5	1.4	1.9	
		21/13.0	62/36.7	27/20.1	192/59.4
225^3	256	1.5	1.4	1.7	
		20/21.5	71/56.8	33/33.8	247/105.1
295^3	441	1.5	1.4	1.7	
		24/13.6	61/32.6	15/15.9	246/98.3
287^3	676	1.5	1.4	1.6	

Table 1: MG aggregation comparison on 3D Poisson problem. Total run time (in seconds)/iterations on top and algebraic complexity below.

while the fine grid is a structured grid (done to facilitate problem setup), the matrices for all coarser levels correspond to unstructured graphs. In these examples we apply automatic coarsening schemes until coarsening can no longer occur or until we reach a coarse grid with 100 points. On each grid level (including the coarsest grid) two steps of a chaotic symmetric Gauss-Seidel method are used as a smoother. The algebraic multigrid method is used as a preconditioner inside a conjugate gradient scheme. Results are also given for a conjugate-gradient method preconditioned by 2 steps of chaotic symmetric Gauss-Seidel. The solution times reported do not include the multigrid setup time but only the time spent in the iterations. The individual kernel times will be discussed later in this section. Operator complexity numbers are obtained by summing the number of matrix nonzeros on all grid levels. This operator complexity can be used as a rough guide to the amount of storage required for each scheme. Finally, all of the results presented in this paper were produced on the Intel TFLOP computer located at Sandia National Laboratories.

Table 1 illustrates our results for the three dimensional Poisson problem. In particular, all the multigrid preconditioners significantly outperform the

N	procs	CG/MG			CG/symGS
		decoupled	coupled	MIS	
		23.1/12	.6/12	.7/11	3.8/100
137^2	64	2.1	1.7	1.4	
		88.6/12	.8/12	.9/12	5.3/132
188^2	121	2.2	1.7	1.4	
		350.4/12	1.7/12	1.8/12	9.7/165
273^2	256	2.2	1.7	1.4	
		949.0/12	1.4/12	1.7/12	5.9/204
358^2	441	2.2	1.7	1.4	
		-	1.0/12	1.2/12	8.7/237
443^2	676		1.7	1.4	

Table 2: MG aggregation comparison on 2D anisotropic problem (using direct solver for coarsest grid). Total run time (in seconds)/iterations on top and operator complexity below.

Gauss-Seidel preconditioners. It is important to note, however, that run times improvements do not grow as significantly as the difference in iterations. This is due to the lower multigrid parallel efficiencies. Comparing the different aggregation schemes, we see that the total number of iterations, run times, and operator complexity do not vary enormously for the decoupled and MIS schemes. At present the coupled scheme is taking more iterations and we suspect that this may be due to some large aggregates. We need to study more closely the coupled scheme as we expect its performance to be comparable to that of the decoupled scheme on this test problem.

Table 2 illustrates our results for a two dimensional anisotropic problem using a direct solver (instead of chaotic Gauss-Seidel) on the coarsest grid. The anisotropic situation is quite different from the Poisson case. In particular, the aggregation methods perform differently. It is well known that for multigrid methods on structured anisotropic problems, it is best to either use semi-coarsening or line-relaxation. This is due to the poor smoothing properties corresponding to directions of weak coupling. Semi-coarsening refers to coarsening the grid only in certain coordinate directions (e.g. not

in the direction of weak coupling). The weak connection dropping which occurs when building the matrix graph will essentially mimic this effect. In particular, aggregates are formed by grouping strongly connected neighbors. While necessary for convergence, this significantly limits the total number of aggregates that can be created. It is this aggregation limitation which is at the root of the poor performance for the decoupled algorithm. Specifically, the decoupled algorithm restricts aggregates to lie on no more than one processor. Thus on our 2D anisotropic problem, aggregation can only occur along y-lines due to weak coupling in the y-direction. However, the decoupled algorithm implies that each y-line must contain at least $\sqrt{P}$ aggregates and therefore the coarsest grid must have at least $\sqrt{NP}$ points. When $N = 443^2$ and $P = 676$, this corresponds to 11518 grid points. In this example we chose a direct solver for the coarsest grid (to eliminate errors in the x direction). This direct solver is replicated on each processor. Normally, this inefficiency is not a concern as the coarse grid contains very little work. We note that we have also experimented with Gauss-Seidel on the coarsest grid. Unfortunately, the slow Gauss-Seidel convergence on the relatively large coarse grid obtained via decoupled aggregation deteriorates the overall multigrid convergence. We expect, however, that the decoupled scheme could be more competitive when efficient parallel direct solvers are available for the solution of the coarsest grid problem (though considerable parallel direct solver improvements are needed for this anisotropic problem).

In Figure 5, we show the results of a quasi-static electric potential calculation inside the code ALEGRA, a primary Sandia platform for multi-physics modeling. The Figure corresponds to a problem consisting of several materials within an assembly containing a bar of poled ceramic. A fused silica flyer plate impacts the assembly and when the wave passes through the bar the ceramic dipoles releasing bound charge and generating a voltage which drives a current through a load. The entire simulation requires many linear solves. Timing results for the first linear solve (which is typical of the solves throughout the several hour long simulation) are given in Table 3. We are currently working to remove the iteration growth that occurs with the multigrid solves. However, even with suboptimal performance the multigrid preconditioning far outperforms symmetric Gauss-Seidel. Here again we see some variance between the different aggregation schemes though it is not completely obvious which one is superior.

As our last example, we consider a thermal convection (or buoyancy driven) flow of a fluid in a differentially heated square box in the presence of

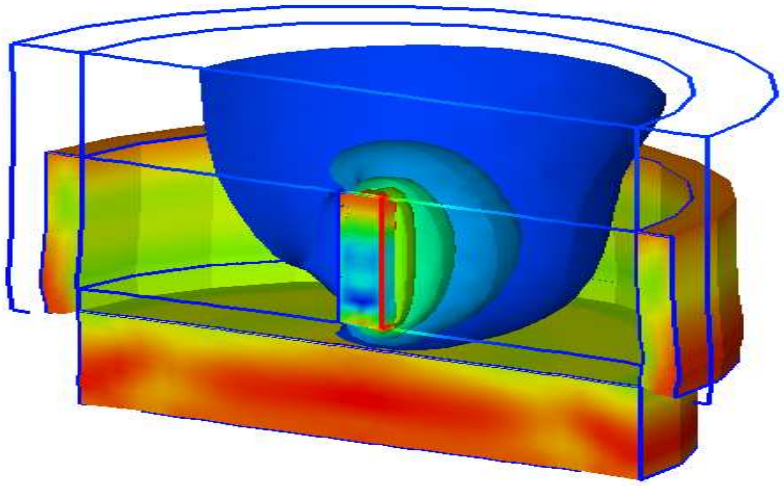

Figure 5: A fused silica flyer plate impacts assembly containing bar of poled ceramic. Stress contours along with 4 electric potential isosurfaces are shown.

N	procs	CG/MG			CG/symGS
		decoupled	coupled	MIS	
		2.3/32	2.3/34	2.5/29	3.3/91
76,157	32	1.03	1.07	1.08	
		5.0/37	5.0/42	4.3/34	8.0/127
251,065	64	1.04	1.06	1.08	
		6.8/43	7.4/49	6.3/38	15.4/187
588,137	128	1.04	1.07	1.10	
		16.4/49	14.3/57	13.0/44	39.9/27
1,961,605	256	1.04	1.06	1.09	
		21.6/54	14.1/67	18.0/48	62.7/364
4,6222,225	512	1.04	1.06	1.09	

Table 3: MG aggregation comparison on 3D QSE problem. Total run time (in seconds)/iterations on top and operator complexity below.

gravity. It requires the solution of the momentum transport, energy transport, and total mass conservation equations on the unit square in the plane with a combination of Dirichlet and Neumann boundary conditions. When the corresponding Navier-Stokes equations are suitably nondimensionalized, two parameters appear, the Prandtl number Pr and the Rayleigh number Ra. In our study we took $Pr = 1$ and $Ra = 100$. A typical computed solution, is given in Figure 6. The overall nonlinear problem is linearized via

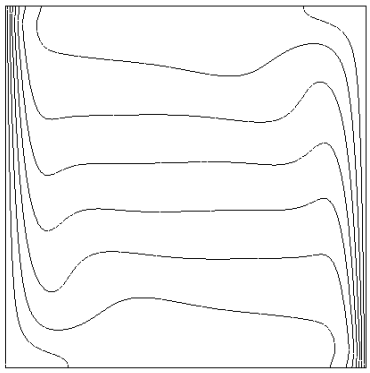

Figure 6: Thermal convection in a square cavity at $Ra = 1,000,000$: Contour plot of temperature shows a thermal boundary layer along hot and cold walls.

Newton's method. In this paper, we give the iterations and solution times corresponding to solving the second Newton iteration (i.e. the Jacobian matrix equation that arises in the second nonlinear step). The results are given in Table 4 correspond to using a GMRES(100) method, a direct solver on the coarsest grid, and a domain decomposition ILU smoother on the finer levels. This smoother corresponds to performing an ILU algorithm on the submatrix owned by each processor. In this case we see a big difference between the methods. In particular, the single level method does not converge at all (except for the smallest problem). The decoupled aggregation method takes a very long time though the number of iterations is reasonable. In this case, almost all of the execution time is spent factorizing and performing the backsolve on the coarsest grid system (which has a large number of nonzeros per row and at least 1024 unknowns as there are 256 processors and 4 unknowns at each grid point). Both the MIS and the coupled schemes perform reasonably with the MIS method performing better on larger problems.

N	GMRES/DD-ILU	GMRES/MG		
		decoupled	coupled	MIS
	7.8/201	63.0/29	1.8/34	3.2/27
4,356		7.0	1.7	1.9
	> 1000	61.4/41	3.0/48	6.3/38
16,900		2.05	1.3	1.6
	> 1000	136.8/60	5.5/68	6.3/52
66,564		1.8	1.2	1.2
	> 1000	158.5/93	27.0/124	16.7/70
264,196		1.3	1.1	1.2
	> 1000	570.1/195	197.9/389	80.4/147
1,052,676		1.2	1.1	1.2

Table 4: MG aggregation comparison on 2D Thermal convection problem. Total run time (in seconds)/iterations on top and operator complexity below.

We now explore the different kernels within the aggregation schemes. Table 5 illustrates the run time breakdown on the thermal convection problem. The CG time includes all the iteration time excluding time spent in the multigrid preconditioner. The V-cycle time includes all the multigrid time for grid transfers, computing residuals and smoothers. The 'construct P' time includes all the time for aggregation, and the time to multiply the tentative prolongator with the damped-Jacobi smoother (for all levels). Since our focus is on the kernel breakdown (as opposed to convergence), all the schemes are run 150 iterations (approximately the number of iterations for the MIS scheme in Table 4). The main difference between the schemes is in the time to construct the prolongation operator. In particular, the MIS and coupled aggregation schemes take significantly more time than the decoupled scheme. If aggregation time is a concern, it might be best to use the decoupled scheme on the finest levels and then switch to either the MIS or coupled scheme on coarser levels so that aggregation is not limited by the processors. Overall, we can see that the time to aggregate and to perform matrix multiplies is not too large for this problem. However, many iterations

kernels	time (seconds)		
	decoupled	coupled	MIS
CG time	35.4	34.7	33.1
coarse LU	348.4	2.7	3.8
V cycle	54.3	38.2	45.8
$P_k^T A_k P_k$	4.8	3.0	3.5
construct P	2.2	17.0	8.0
total	445.1	95.6	94.2

Table 5: Kernel times for 150 iterations of thermal convection problem containing 1,052,676 degrees of freedom on 256 processors.

are taken and when fewer iterations are required, startup time is more significant. Often in practice, some startup phases can be amortized over many solves where either the actual linear system is the same or the linear system sparsity pattern remains fixed.

5 Conclusions

Parallel variants of one of the most promising algebraic multigrid methods, smoothed aggregation, have been presented. The main obstacle is the proper parallelization of the aggregation phase. For some problems a simple decoupled parallel aggregation schemes performs adequately. However, in some situations taking into account the inter-processor coupling is important to maintain good convergence properties and keep the cost per iteration minimal. We have proposed a parallel coupled aggregation scheme which first aggregates grid points near inter-processor boundaries and then aggregates interior grid points. In addition, another parallel coupled aggregation scheme has been proposed by casting the first phase of the aggregation process as a search for the MIS. Results demonstrating the effectiveness of these procedures have been given. Performance results for some unstructured applications on the Intel TFLOP computer using our parallel smoothed aggregation scheme have also been given.

Acknowledgement : The authors would like to thank Professor Stephen Vavasis for his advice on formulating the aggregation process as an MIS problem.

References

[1] M. Adams, *Evaluation of Geometric and an Algebraic Multigrid Method on 3D Finite Element Problems in Solid Mechanics*. Talk at the Copper Mountain Conference on Iterative Methods, Copper Mtn, Co, 4/2000.

[2] M. F. Adams, *A parallel maximal independent set algorithm*, in Proceedings 5th copper mountain conference on iterative methods, 1998. Best student paper award winner.

[3] J. Bramble, J. Pasciak, J. Wang, and J. Xu, *Convergence Estimates for Multigrid Algorithms without Regularity Assumptions*, Math Comp., 57 (1991), pp. 23–45.

[4] A. Brandt, *Multi-level Adaptive Solutions to Boundary-Value Problems*, Math. Comp., 31 (1977), pp. 333–390.

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

[6] ——, *Iterative Solution of Large Sparse Linear Systems of Equations*, Springer-Verlag, Berlin, 1994.

[7] V. Henson and U. Yang, *Coarse Grid Selection of Algebraic Multigrid*, Tech. Rep. UCRL-MI-13089, Lawrence Livermore National Laboratory, Livermore, CA, 1999.

[8] E. Jenkins, R. Berger, J. Hallberg, S. Howington, C. T. Kelley, J. Schmidt, A. Stagg, and M. Tocci, *A Two-Level Aggregation-Based Newton-Krylov-Schwarz Method for Hydrology*, in Parallel Computational Fluid Dynamics 1999, D. Keyes, A. Ecer, J. Periaux, and N. Satofuka, eds., North Holland, 2000, pp. 257–264.

[9] M. T. Jones and P. E. Plassman, *A Parallel Graph Coloring Heuristic*, SIAM J. Sci. Comput., 14 (1993), pp. 654–669.

[10] C. Tong, R. Tuminaro, K. Devine, and J. Shadid, *Design of a Multilevel Preconditioning Module for Unstructured Calculations*, Tech. Rep. in preparaion, Sandia National Laboratories, Albuquerque NM, 87185, 2000.

[11] P. Vanek, *Acceleration of Convergence of a Two Level Algorithm by Smooth Transfer Operators*, Appl. Math., 37 (1992), pp. 265–274.

[12] P. Vanek, M. Brezina, and J. Mandel, *Convergence of Algebraic Multigrid Based on Smoothed Aggregation*, Tech. Rep. report 126, UCD/CCM, Denver, CO, 1998.

[13] P. Vanek, J. Mandel, and M. Brezina, *Algebraic Multigrid Based on Smoothed Aggregation for Second and Fourth Order Problems*, Computing, 56 (1996), pp. 179–196.