Parallel Algebraic Multigrid Methods - High Performance Preconditioners

Ulrike Meier Yang

Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Box 808, L-560, Livermore, CA 94551, USA umyang@llnl.gov

Summary. The development of high performance, massively parallel computers and the increasing demands of computationally challenging applications have necessitated the development of scalable solvers and preconditioners. One of the most effective ways to achieve scalability is the use of multigrid or multilevel techniques. Algebraic multigrid (AMG) is a very efficient algorithm for solving large problems on unstructured grids. While much of it can be parallelized in a straightforward way, some components of the classical algorithm, particularly the coarsening process and some of the most efficient smoothers, are highly sequential, and require new parallel approaches. This chapter presents the basic principles of AMG and gives an overview of various parallel implementations of AMG, including descriptions of parallel coarsening schemes and smoothers, some numerical results as well as references to existing software packages.

1 Introduction

The development of multilevel methods was a very important step towards being able to solve partial differential equations fast and efficiently. One of the first multilevel methods was the multigrid method. Multigrid builds on a relaxation method, such as the Gauß-Seidel or the Jacobi method, and was prompted by the discovery, that while these relaxation schemes efficiently damp high frequency errors, they make little or no progress towards reducing low frequency errors. If however one moves the problem to a coarser grid previously low frequency errors turn now into high frequency errors and can be damped efficiently. If this procedure is recursively applied, one obtains a method with a computational cost that depends only linearly on the problem size.

There are two basic multigrid approaches: geometric and algebraic. In geometric multigrid, the geometry of the problem is used to define the various multigrid components. Algebraic multigrid (AMG) methods use only the information available in the linear system of equations and are therefore suitable to solve problems on more complicated domains and unstructured grids. AMG was first introduced in the 80s [7, 8, 5, 46]. Since then, a lot of research has been done and many new variants have been developed, e.g. smoothed aggregation [56, 55], AMGe [11], spectral AMGe [14], to just name a few. Whole books have been written on this topic [37, 52]. A good tutorial on multigrid, including AMG, is [12]. A very detailed introduction on AMG is [49].

Focus of this chapter is not an overview of all existing AMG methods (there would not be enough space), but a presentation of the basic idea of AMG, the challenges that come with a parallel implementation of AMG and how to overcome them, as well as the impact this challenge has had on AMG.

With the advent of parallel computers one has sought parallel algorithms. When vector computers where developed in the 70s, it was important to develop algorithms that operated on long vectors to make good use of pipelines or vector registers. For parallel computers with tens or hundreds of processors and slow intercommunication it was necessary to be able to partition an algorithm into big independent pieces in order to avoid large communication cost. Multigrid methods, due to their decreasing levels, did not appear to be good candidates for these machines. However with the development of high performance computer with tens or hundreds of thousands of processors it has become very important to develop scalable algorithms, and therefore multigrid methods as well as the application of multilevel techniques in other algorithms have become very popular in parallel computing.

In this chapter, the concept of AMG as well as various parallel variations will be described. While most of AMG can be parallelized in a straightforward way, the coarsening algorithm and the smoother are more difficult to parallelize. Therefore a large portion of this chapter is devoted to parallel approaches for these components. Additionally, interpolation as well as a few numerical results are presented. Finally, an overview of various parallel software packages is given that contain algebraic multigrid or multilevel codes.

2 Algebraic Multigrid - Concept and Description

We begin by outlining the basic principles and techniques that comprise AMG. Detailed explanations may be found in [46]. Consider a problem of the form

$$Au = f, \tag{1}$$

where A is an $n \times n$ matrix with entries a_{ij}. For convenience, the indices are identified with grid points, so that u_i denotes the value of u at point i, and the grid is denoted by $\Omega = \{1, 2, \ldots, n\}$. In any multigrid method, the central idea is that "smooth error," e, that is not eliminated by relaxation must be removed by coarse-grid correction. This is done by solving the residual equation $Ae = r$ on a coarser grid, then interpolating the error back to the fine grid and using it to correct the fine-grid approximation by $u \leftarrow u + e$.

Using superscripts to indicate level number, where 1 denotes the finest level so that $A^1 = A$ and $\Omega^1 = \Omega$, the components that AMG needs are as follows:

1. "Grids" $\Omega^1 \supset \Omega^2 \supset \ldots \supset \Omega^M$ with subsets:
   Set of coarse points or C-points $C^k, k = 1, 2, \ldots M-1$,
   Set of fine points or F-points $F^k, k = 1, 2, \ldots M-1$.
2. Grid operators $A^1, A^2, \ldots, A^M$.
3. Grid transfer operators:
   Interpolation $P^k, k = 1, 2, \ldots M-1$,
   Restriction $R^k, k = 1, 2, \ldots M-1$.
4. Smoothers $S^k, k = 1, 2, \ldots M-1$.

These components of AMG are constructed in the first step, known as the *setup phase*.

```
AMG Setup Phase:
    1. Set $k = 1$.
    2. Partition $\Omega^k$ into disjoint sets $C^k$ and $F^k$.
       a) Set $\Omega^{k+1} = C^k$ .
       b) Define interpolation $P^k$.
    3. Define $R^k$ (often $R^k = (P^k)^T$).
    4. Set $A^{k+1} = R^k A^k P^k$.
    5. Set up $S^k$, if necessary.
    6. If $\Omega^{k+1}$ is small enough, set $M = k+1$ and stop. Otherwise,
       set $k = k+1$ and go to step 2.
```

Once the setup phase is completed, the *solve phase*, a recursively defined cycle, can be performed as follows:

```
Algorithm: $MGV(A^k, R^k, P^k, S^k, u^k, f^k)$.
    If $k = M$, solve $A^M u^M = f^M$ with a direct solver.
    Otherwise:
        Apply smoother $S^k$ $\mu_1$ times to $A^k u^k = f^k$.
        Perform coarse grid correction:
           Set $r^k = f^k - A^k u^k$.
           Set $r^{k+1} = R^k r^k$.
           Apply $MGV(A^{k+1}, R^{k+1}, P^{k+1}, S^{k+1}, e^{k+1}, r^{k+1})$.
           Interpolate $e^k = P^k e^{k+1}$.
           Correct the solution by $u^k \leftarrow u^k + e^k$.
        Apply smoother $S^k$ $\mu_2$ times to $A^k u^k = f^k$.
```

The algorithm above describes a V(μ_1, μ_2)-cycle, other more complex cycles such as W-cycles can be found in [12].

Coarse grid selection and interpolation must go hand in hand and affect each other in many ways. There are basically two measures which give an indication about the quality of the AMG method, both need to be considered and are important, although depending on the user's priorities one might be more important than the other. The first one, the *convergence factor*, gives an indication on how fast the method converges, i.e. how many iterations are needed to achieve the desired accuracy, the second one, *complexity*, affects the number of operations per iteration and the memory usage.

There are two types of complexities that need to be considered: the *operator complexity* and the *average stencil size*. The operator complexity C_{op} is defined as the quotient of the sum of the numbers of nonzeroes of the matrices on all levels, $A^k, k = 1, ..., M$, divided by the number of nonzeroes of the original matrix $A^1 = A$. This measure indicates how much memory is needed. If memory usage is a concern, it is important to keep this number small. It also affects the number of operations per cycle in the solve phase. Small operator complexities lead to small cycle times. The average stencil size $s(A^k)$ is the average number of coefficients per row of A^k. While stencil sizes of the original matrix are often small, it is possible to get very large stencil sizes on coarser levels. Large stencil sizes can lead to large setup times, even if the operator complexity is small, since various components, particularly coarsening and to some degree interpolation, require that neighbors of neighbors are visited and so one might observe superlinear or even quadratic growth in the number of operations when evaluating the coarse grid or the interpolation matrix. Large stencil sizes can also increase parallel communication cost, since they might require the exchange of larger sets of data.

Both convergence factors and complexities need to be considered when defining the coarsening and interpolation procedures, as they often affect each other; increasing complexities can improve convergence, and small complexities lead to a degradation in convergence. The user needs therefore to decide his/her priority. Note that often a degradation in convergence due to low complexity can be overcome or diminished by using the AMG method as a preconditioner for a Krylov method like conjugate gradient, GMRES, BiCGSTAB, etc.

Many parts of AMG can be parallelized in a straightforward way, since they are matrix or vector operations. This is generally true for the evaluation of the interpolation or prolongation matrix as well as the generation of the triple matrix product $R^k A^k P^k$. Certainly all those operations require communication among processors and data exchange in some way, however all of this can be done in a straightforward way. There are however two components which present potentially a serious challenge, the coarsening routine as well as the relaxation routine. The original coarsening routine as described in [46] as well as the basic aggregation procedure are inheritantly sequential. Also, the relaxation routine used in general is the Gauß-Seidel algorithm, which is also sequential in nature. In the following sections, the various components, coarse grid selection, interpolation and smoothing are described.

3 Coarse Grid Selection

Before describing any parallel coarsening schemes, we will describe various sequential coarsening schemes, since most parallel schemes build on these.

3.1 Sequential Coarsening Strategies

There are basically two different ways of choosing a coarse grid. The first approach (which can be found e.g. in [46, 49]) strives to separate all points i into either coarse points or C-points, which will be taken to the next level, and fine points or F-points, which will be interpolated by the C-points. The second approach, coarsening by aggregation or agglomeration ([56]), accumulates aggregates which will be the coarse "points" for the next level.

"Classical" Coarsening

Since most likely not all matrix coefficients are equally important for the determination of the coarse grids, one should only consider those matrix entries that are sufficiently large. We introduce the concept of *strong influence* and *strong dependence*. A point i depends strongly on j or j strongly influences i if

$$-a_{ij} \geq \theta \max_{k \neq i}(-a_{ik}). \tag{2}$$

Note that this definition was originally motivated by the assumption that A is a symmetric M-matrix, i.e. a matrix that is positive definite and off-diagonally nonpositive, it can however formally be applied to more general matrices. The size of the *strength threshold* θ can have a significant influence on complexities, particularly stencil size, as well as convergence, as is demonstrated in Section 6. In the classical coarsening process (which we will denote Ruge-Stüben or RS coarsening) the attempt is made to fulfill the following two conditions:

(C1): For each point j that strongly influences an F-point i, j is either a C-point or it strongly depends on a C-point k that also strongly influences i.

(C2): The C-points should be a maximal independent subset of all points, i.e. no two C-points are connected to each other, and if another C-point is added the independence is lost.

(C1) is designed to insure the quality of interpolation, while (C2) is designed to restrict the size of the coarse grids. In general, it is not possible to fulfill both conditions, therefore (C1) is enforced, while (C2) is used as a guideline. We will show in Section 4 why (C1) is important.

RS coarsening consists of two passes and is illustrated in Figure 1 for a 4×4-grid. In the first pass, each point i is assigned a measure λ_i, which equals the number of points that are strongly influenced by i. Then a point with a maximal λ_i (there usually will be several) is selected as the first coarse point.

Now all points that strongly depend on i become F-points. For all points that strongly influence these new F-points, λ_j is incremented by the number of new F-points that j strongly influences in order to increase j's chances of becoming a C-point. This process is repeated until all points are either C- or F-points.

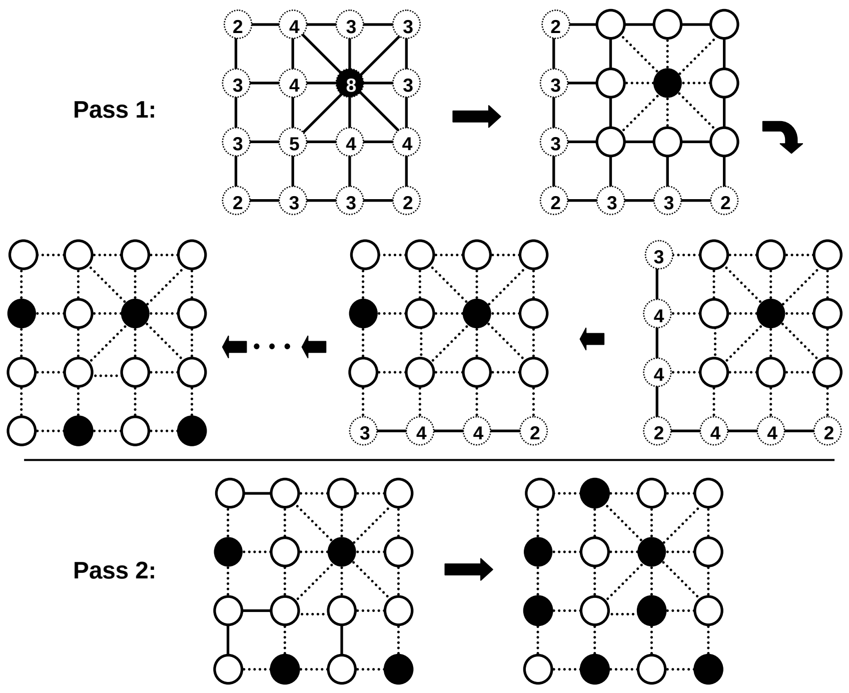


Fig. 1. RS coarsening. Black points denote C-points, white points with solid border denote F-points, white points with dotted border denote undetermined points.Pass 2: solid lines denote strong $F-F$ connections that do not satidfy condition (C1).

Since this first pass does not guarantee that condition (C1) is satisfied, it is followed by a second pass, which examines all strong $F-F$ connections for common coarse neighbors. If (C1) is not satisfied new C-points are added. This is illustrated in the second part of Figure 1, where solid lines denote strong $F-F$ connections that do not satisfy condition (C1).

Aggressive Coarsenings

Experience has shown [49] that often the second pass generates too many C-points, causing large complexities and inefficiency. Therefore condition (C1) has been modified to the following.

(C1$'$): Each F-point i needs to strongly depend on at least one C-point j.

Now just the first pass of the RS coarsening fulfills this requirement. This method leads to better complexities, but worse convergence.

Even though this approach often decreases complexities significantly, complexities can still be quite high and require more memory than desired. Allowing C-points to be even further apart leads to *aggressive coarsening.* This is achieved by the following new definition of strength: A variable i is *strongly n-connected along a path of length l* to a variable j, if there exists a sequence of variables $i_0, i_1, \ldots i_l$, with $i = i_0$ and $j = i_l$ and i_k strongly connected (as previously defined) to i_{k+1} for $k = 0, \ldots, l-1$. A variable i is *strongly n-connected w.r.t. (p, l)* to a variable j, if at least p paths of lengths $\leq l$ exist such that i is strongly n-connected to j along each of these paths. This can be most efficiently implemented by applying the first pass of RS coarsening twice, the first time as described in the previous section, the second time by defining strong n-connectivity w.r.t. (p, l) only between the resulting C-points (via neighboring F-points). For further details see [49]. The result of applying aggressive A2 coarsening, i.e. choosing $p = 2$ and $l = 2$, and aggressive A1 coarsening, i.e. $p = 1$ and $l = 2$, to the 5-point Laplacian on a 7×7-grid is illustrated in Figure 2.

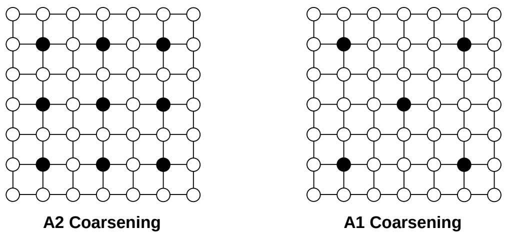


Fig. 2. Various sequential coarsenings for a 5-point Laplacian.

Coarsening by Aggregation

For the aggregation scheme, a different concept of strength is used. Here only matrix coefficients a_{ij} are considered, if they fulfill the following condition:

$$|a_{ij}| > \theta \sqrt{|a_{ii} a_{jj}|}. \tag{3}$$

An aggregate is defined by a root point i and its neighborhood, i.e. all points j, for which a_{ij} fulfills (3). Now the basic aggregation procedure consists of the following two phases. In the first pass, a root point is picked that is not adjacent to any existing aggregate. This procedure is repeated until all unaggregated points are adjacent to an aggregate. It is illustrated in Figure 3 for a 5-point Laplacian on a 6×5-grid. In the second pass, all remaining unaggregated points are either integrated into already existing aggregates or used to

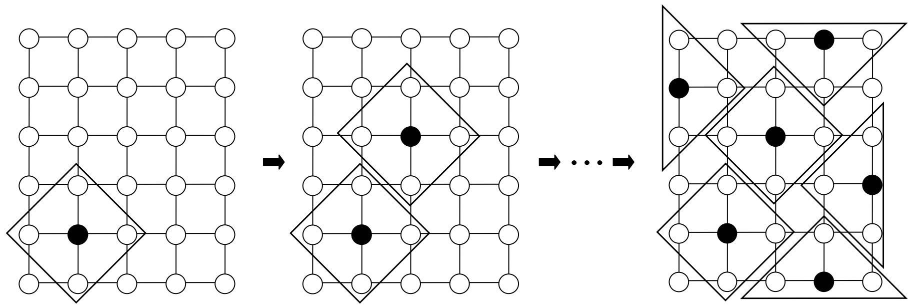


Fig. 3. Coarsening by aggregation. Black points denote root points, boxes and triangles denote aggregates.

form new aggregates. Since root points are connected by paths of length of at least 3, this approach leads to fast coarsening and small complexities. While aggregation is fundamentally different from classical coarsening, many of the same concerns arise. In particular, considerable care must be taken within the second pass when deciding to create new aggregates and what points should be placed into already existing aggregates. If too many aggregates are created in this phase, complexities grow. If aggregates are enlarged too much or have highly irregular shapes, convergence rates suffer.

3.2 Parallel Coarsening Strategies

There are various approaches of parallelizing the coarse grid selection schemes described in the previous section, which are described in the following subsections.

Decoupled Coarsening Schemes

The most obvious approach to parallelize any of the coarsening schemes described in the previous section is to partition all variables into subdomains, assign each processor a subdomain, coarsen the variables on each subdomain using any of the methods described above, and find a way of dealing with the variables that are located on the processor boundaries.

The easiest option is to just coarsen independently on each subdomain while ignoring the processor boundaries. Such an approach is the most efficient one, since it requires no communication, but will most likely not produce a very good coarse grid. The decoupled RS coarsening, which will be denoted by RS0 coarsening, generally violates condition (C1) by generating strong $F-F$ connections without common coarse neighbors (see Figure 5a, which shows the coarse grid generated by RS0 coarsening on 4 processors; black points denote C-points, while white and gray points denote F-points.) and often leads to poor convergence, see Section 6 and [25]. While in practice this approach might lead to fairly good results for coarsening by aggregation [53], it can produce many aggregates near processor boundaries that are either smaller or larger than an ideal aggregate and so lead to larger complexities or have a negative effect on convergence. Another disadvantage of this approach is that it cannot have fewer coarse points or aggregates than processors. In the case of thousands of processors this leads to a large grid and a large system on the coarsest level and might be inefficient to solve using a direct solver. Ways to overcome this problem are described at the end of this section.

Coupled Coarsening Strategies

If we want to improve RS0 coarsening, we need to find ways to deal with the variables that are located on the processor boundaries. This starts with a decision about whether one wants to compute the measures locally or globally, i.e. whether one wants to include off processor influences. The use of global measures can improve the coarsening and convergence, but is in general not enough to fulfill condition (C1). One possible way of treating this problem is — after one has performed a first and a second pass on each processor independently — to perform a third pass only on the processor boundary points which will add further C-points and thus ensure that condition (C1) is fulfilled. This approach is called RS3 coarsening and can be found in [25]. It is illustrated in Figure 5a for a 5-point Laplacian on a 10×10-grid on 4 processors. The black points denote the C-points that have been generated in the first (and second) pass, the gray points denote the C-points generated in the third pass. One of the disadvantages of this approach is that this can generate C-point clusters on the boundaries, thus increasing stencil sizes at the boundaries where one would like to avoid those, in order to keep communication cost low.

In the coupled aggregation method, aggregates are first built on the boundary. This step is not completely parallel. When there are no more unaggregated points adjacent to an aggregate on the processor boundaries, one can proceed to choose aggregates in the processor interiors, which can be done in parallel. In the third phase unaggregated points on the boundaries and in the interior are swept into local aggregates. Finally, if there are any remaining points, new local aggregates are formed. This process yields significantly better aggregates and does not limit the coarseness of grids to the number of processors, see [53].

Parallel Independent Set Coarsenings

A completely parallel approach is suggested in [17, 25]. It is based on parallel independent set algorithms as described by Luby and Jones and Plassman in [36, 30]. This algorithm, the CLJP (Cleary-Luby-Jones-Plassman) coarsening, begins by generating global measures as in RS coarsening, and then adding a random number between 0 and 1 to each measure, thus making them distinctive. It is now possible to find unique local maxima. The algorithm, which is illustrated for a small example on a 4×4-grid in Figure 4, proceeds as follows: If i is a local maximum, make i a C-point, eliminate the connections to all points j that influence i and decrement j's measure. (Thus instead of immediately making C-point neighbors F-points, we increase their likelihood of becoming F-points. This models the two passes of the RS coarsening into one pass.) Further for all points j that depend on i, remove its connection to i and examine all points k that depend on j on whether they also depend on i. If i is a common neighbor for both k and j decrement the measure of j and remove the edge connecting k and j from the graph. If a measure gets smaller than 1, the point associated with it becomes an F-point. The advantage of

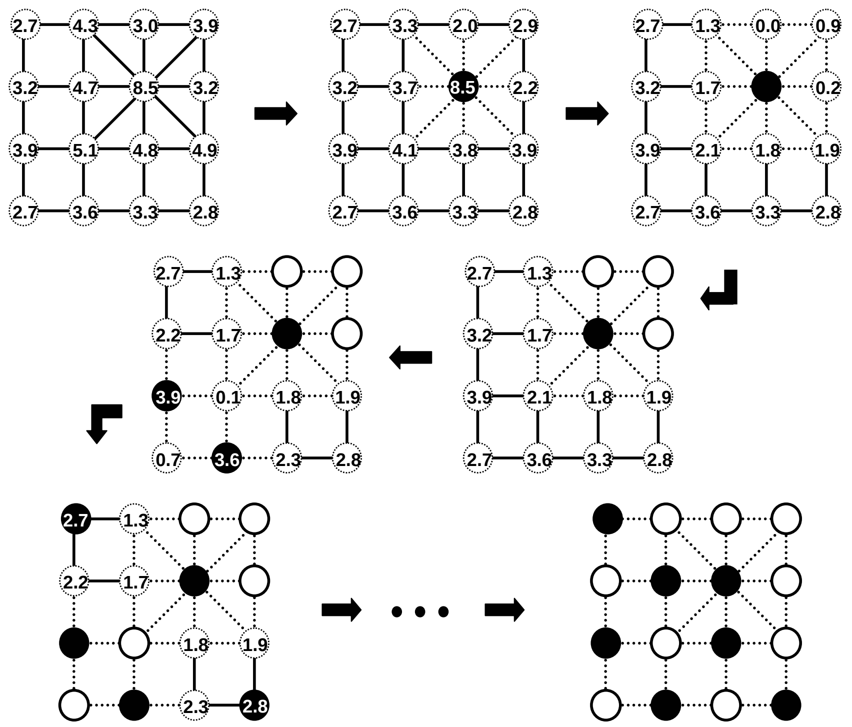


Fig. 4. CLJP coarsening. Black points are C-points, white points with solid border F-points, white points with dotted border undetermined points.

this procedure is, assuming one uses the same global set of random numbers, that it is completely independent of the number of processors, a feature that is desired by some users. It also facilitates debugging. The additional advantage of this procedure is that it does not require the existence of a coarse point in each processor as the coarsening schemes above and thus coarsening does not slow down on the coarser levels. While this approach works fairly well on truly unstructured grids, it often leads to C-point clusters and fairly high complexities, see Figure 5b. These appear to be caused by the enforcement of condition (C1).

To reduce operator complexities, while keeping the property of being independent of the number of processors, a new algorithm, the PMIS coarsening [19], has been developed that is more comparable to using one pass of the RS coarsening. While it does not fulfill condition (C1), it fulfills condition (C1′). PMIS coarsening begins just as the CLJP algorithm with distinctive global measures, and sets local maxima to be C-points. Then points that are influenced by C-points are made F-points, and are eliminated from the graph. This procedure will continue until all points are either C- or F-points. For an illustration of the PMIS coarsening applied to a 5-point Laplacian see Figure 5c.

Another aggregation scheme suggested in [53] is also based on a parallel maximally independent set algorithm. Since the goal is to find an initial set of aggregates with as many points as possible with the restriction that no root point can be adjacent to an existing aggregate. Therefore maximizing the number of aggregates is equivalent to finding the largest number of root points such that the distance between any two root points is at least three. This can be accomplished by applying a parallel maximally independent set (MIS) algorithm, e.g. the asynchronous distributed memory algorithm ADMMA [1], to the square of the matrix in the first phase of the coupled aggregation scheme.

Subdomain Blocking

Another parallel approach is *subdomain blocking* [33]. Here, coarsening starts with the processor boundaries, and one then proceeds to coarsen the inside of the domains. *Full subdomain blocking* is performed by making all boundary points coarse and then coarsening into the interior of the subdomain using any coarsening scheme one wishes to use, such as one pass of RS coarsening or any of the aggressive coarsening schemes. The disadvantage of this scheme is that it generates far too many C-points on the boundary, which can cause problems on the coarser grids. For an illustration see Figure 5d. A method, which avoids this problem, is *minimum subdomain blocking*. This approach uses standard coarsening on the boundaries and then coarsens the interior of the subdomains.

Combination Approaches and Miscellaneous

Another option which has shown to work quite well for structured problems is the following combination of the RS and the CLJP coarsening which is based on an idea by Falgout [25]. This coarsening starts out as RS0 coarsening. It then uses the C-points that have been generated in the first step and are located in the interior of each processor as the first independent set (i.e. they will all remain C-points) and feeds them into the CLJP-algorithm. The resulting coarsening, which satisfies condition (C1), fills the boundaries with further C-points and possibly adds a few in the interior of the subdomains, see Figure 5e. A more aggressive scheme, which satisfies condition (C1′), and uses the same idea, is the HMIS coarsening [19]. It performs only the first pass of RS0 coarsening to generate the first independent set, which then is used by the PMIS algorithm. Figure 5f shows its application to a 5-point Laplacian on 4 processors. The black C-points are the first independent set, which have been generated by the first pass of the RS0 coarsening, whereas the gray C-points have been determined by the application of the PMIS coarsening.

Another approach is to color the processors so that subdomains of the same color are not connected to each other. Then all these subdomains can be coarsened independently. This approach can be very inefficient since it might lead to many idle processors. An efficient implementation that builds on this approach can be found in [31]. Here the number of colors is restricted to n_c, i.e. processors with color numbers higher than n_c are assigned the color n_c. Good results were achieved using only two colors on the finest level, but allowing more colors on the coarser levels.

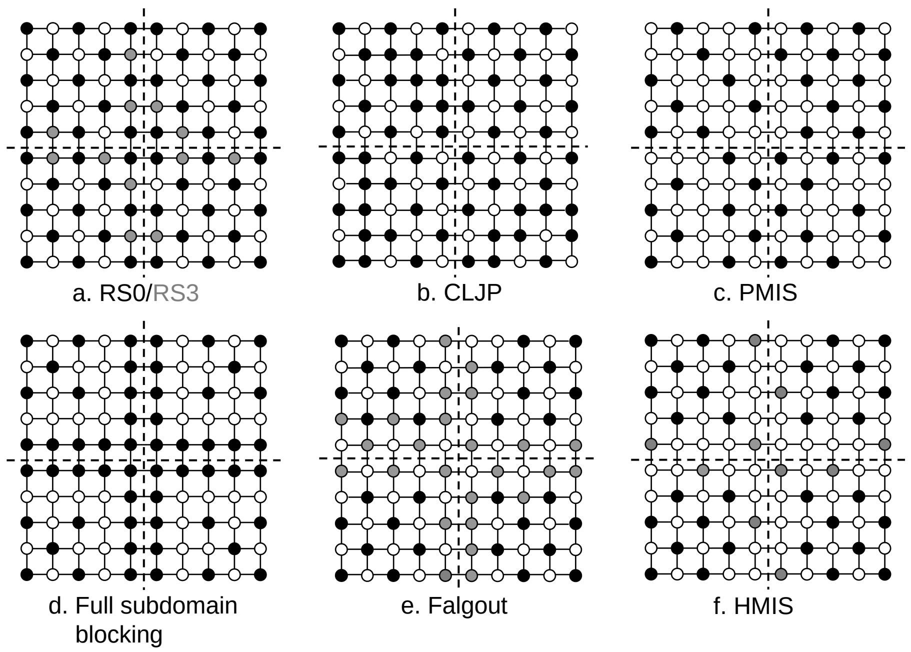


Fig. 5. Various parallel coarsenings of a 5-point Laplacian on a 10×10-grid using 4 processors. White points are F-points, black points are C-points, gray points are C-points generated during special boundary treatments.

Dealing with the Coarser Levels

One of the difficulties that arises when parallelizing multilevel schemes is the treatment of the coarser levels. Since many AMG schemes use the sequential approach on each processor and this often requires at least one coarse point or aggregate on each processor, coarsening will slow down on the coarser grids leading to a coarsest grid of the size of at least the number of processors. This could be thousands of unknowns and lead to a very inefficient coarse grid solve, and potentially prevent scalability. There are various possibilities to deal with this situation. Via aggregation one can combine the contents of various processors, when a certain size is achieved. This approach also coarsens previous processor boundaries and thus deal with cluster of coarse points in these areas. One disadvantage of this approach is that it requires a lot of communication and data transfer across processors.

Another possibility to deal with a slowdown in coarsening is to switch to a coarsening scheme that does not require C-points on each processor, such as CLJP or PMIS, when coarsening slows.

4 Interpolation

In this section, we will consider the construction of the interpolation operator. The interpolation of the error at the F-point i takes the form

$$e_i = \sum_{j \in C_i} w_{ij} e_j \tag{4}$$

where w_{ij} is an interpolation weight determining the contribution of the value e_j in the final value e_i, and C_i is the subset of C-points whose values will be used to interpolate a value at i.

In classical AMG the underlying assumption is that algebraically smooth error corresponds to having very small residuals; that is the error is smooth when the residual $r = f - Au \approx 0$. Since the error, e, and the residual are related by $Ae = r$, smooth error has the property $Ae \approx 0$. Let i be an F-point to which we wish to interpolate, N_i the neighborhood of i, i.e. the set of all points, which influence i. Then the ith equation becomes

$$a_{ii} e_i + \sum_{j \in N_i} a_{ij} e_j = 0. \tag{5}$$

Now, "classical" interpolation as described in [46] proceeds by dividing N_i into the set of coarse neighbors, C_i, the set of strongly influencing neighbors, F_i^s, and the set of weakly influencing neighbors, F_i^w. Using those distinctions as well as condition (C1), which guarantees that a neighbor in F_i^s is also strongly influenced by at least one point in C_i yields the following interpolation formula

$$w_{ij} = -\frac{1}{a_{ii} + \sum_{k \in F_i^w} a_{ik}} \left(a_{ij} + \sum_{k \in F_i^s} \frac{a_{ik} a_{kj}}{\sum_{m \in C_i} a_{km}} \right). \tag{6}$$

Obviously, this interpolation formula fails whenever (C1) is violated, since there would be no m and one would divide by zero. One can somewhat remedy this by including elements of F_i^s that violate (C1) in F_i^w, but this will affect the quality of the interpolation and lead to worse convergence. Nevertheless often good results can be achieved with this interpolation, if the resulting AMG method is used as a preconditioner for a Krylov method, see [19]. One advantage of this interpolation formula is that it only involves immediate neighbors and thus is easier to implement in parallel, since it requires only one layer of ghost points located on a neighbor processor.

Another interpolation formula which also requires only immediate neighbors, and can be used when (C1) is violated, but leads in general to worse convergence rates is *direct interpolation*:

$$w_{ij} = -\left(\frac{\sum_{k \in N_i} a_{ik}}{\sum_{l \in C_i} a_{il}} \right) \frac{a_{ij}}{a_{ii}}. \tag{7}$$

It is easy to implement sequentially as well as in parallel.

However, if one needs an interpolation that yields lower convergence rates, a better approach is *standard interpolation*, which uses an extended neighborhood. For all points $j \in F_i^s$, one substitutes e_j in (5) by $-\sum_{k \in N_j} a_{jk} e_k / a_{jj}$. This leads to a new formula

$$\hat{a}_{ii} e_i + \sum_{j \in \hat{N}_i} \hat{a}_{ij} e_j = 0, \qquad \hat{N}_i = \{ j \neq i : \hat{a}_{ij} \neq 0 \}. \tag{8}$$

The interpolation weights are then defined as in the direct interpolation by replacing a by $\hat{a}$ and N by $\hat{N}$.

If one uses any of the aggressive coarsening schemes, it is necessary to use long range interpolation, such as *multipass interpolation*, in order to achieve reasonable convergence. Multipass interpolation starts by deriving interpolation weights using direct interpolation for all fine points that are influenced by coarse points (which are connected by a path of length 1). In the following pass it evaluates weights using the same approach as in standard interpolation for points that are influenced by those points for which interpolation weights have already been determined and which are connected by a path of length 1. This process is repeated until weights have been obtained for all remaining points. One disadvantage of multipass interpolation is that since it is based on direct interpolation it often converges fairly slowly. Therefore it has been suggested to improve this interpolation formula by using an a posteriori Jacobi relaxation and applying it to the interpolation operator as follows

$$P_{FC}^{(n)} = (I - D_{FF}^{-1} A_{FF}) P_{FC}^{(n-1)} - D_{FF}^{-1} A_{FC}, \tag{9}$$

where P_{FC} and $(A_{FF} \quad A_{FC})$ consist of those rows of the matrices P and A that refer to the F-points only, and D_{FF} is the diagonal matrix with the diagonal of A_{FF}. Of course using one or more sweeps of Jacobi interpolation will increase the number of nonzeroes of P and also the complexity of the AMG method. Therefore often *interpolation truncation* is used, which truncates those elements in the interpolation operator that are absolutely smaller than a chosen *truncation factor* τ, leading to smaller stencil sizes.

Parallel implementation of long range interpolation can be tedious, since due to the long ranges it involves several layers of off processor points. Some of these points might even be located in processors that are not neighbor processors according to the chosen data structure. Therefore this approach might require expensive communication. One way to avoid this problem has been suggested in [49]. There, interpolation proceeds only into the interior of the subdomain and not across boundaries. While this approach can be implemented very efficiently on a parallel computer, it can cause convergence problems.

Aggregation methods use a different type of interpolation. At first a tentative interpolation operator $(P^k)^{(0)}$ is created using one or more seed vectors. These seed vectors should correspond to components that are difficult to smooth. The tentative interpolation interpolates the seed vectors perfectly by ensuring that all of them are in the range of the interpolation operator. For Poisson problems the entries are defined as follows:

$$(P^k)^{(0)}_{ij} = \begin{cases} 1 & \text{if point } i \text{ is contained in aggregate } j \\ 0 & \text{otherwise,} \end{cases} \tag{10}$$

For specific applications such as elasticity problems, more complicated tentative prolongators can be derived based on rigid body motions. Once the tentative prolongator is created it is smoothed via a damped Jacobi iteration

$$(P^k)^{(n)} = (I - \omega (D^k)^{-1} A^k)(P^k)^{(n-1)}, n = 0, \ldots, \tag{11}$$

where D^k is the diagonal matrix with the diagonal of A^k. For a more detailed description of this method see [56, 55].

5 Smoothing

One important component of algebraic multigrid is the smoother. A good smoother will reduce the oscillatory error components, whereas the 'smooth' error is transferred to the coarser grids. Although the classical approach of AMG focused mainly on the Gauß-Seidel method, the use of other iterative solvers has been considered. Gauß-Seidel has proven to be an effective smoother for many problems, however its main disadvantage is its sequential nature. For elasticity problems, Schwarz smoothers have shown to be extremely efficient, here again the most efficient ones are multiplicative Schwarz smoothers, which are highly sequential.

The general definition of a smoother S applied to a system $Au = f$ is

$$e_{n+1} = Se_n \textbf{ or } u_{n+1} = Su_n + (I - S)A^{-1}f, \tag{12}$$

where $e_n = u_n - u$ denotes the error. Often S is the iteration matrix of an iterative solver $S = I - Q^{-1}A$, where Q is a matrix that is part of a splitting $Q + (Q - A)$ of A, e.g. Q is the lower triangular part of A for the Gauß-Seidel method. Other approaches such as polynomial smoothers or approximate inverse set Q^{-1} to be an approximation of A^{-1} that can easily be evaluated. Often iterative schemes that are used as smoothers are also presented as

$$u_{n+1} = u_n + Q^{-1}(f - Au_n). \tag{13}$$

5.1 Parallel Relaxation Schemes

There are various conventional relaxation schemes that are already parallel, such as the Jacobi or the block Jacobi algorithm. Here Q is the diagonal matrix (or block diagonal matrix) with the diagonal (or block diagonal) elements of A. These relaxation schemes require in general a smoothing or relaxation parameter for good convergence, as discussed in the next subsection. Another equally parallel related algorithm that in general leads to better convergence than Jacobi relaxation, is C-F Jacobi relaxation, where first the variables associated with C-points are relaxed, then the F-variables. In algebraic multigrid C-F Jacobi is used on the downward cycle, and F-C Jacobi, i.e. relax the F-variables before the C-variables, on the upward cycle.

5.2 Hybrid Smoothers and the Use of Relaxation Parameters

The easiest way to implement any smoother in parallel is to just use it independently on each processor, exchanging boundary information after each iteration. We will call such a smoother a *hybrid smoother*. Using the terminology of (13), for a computer with p processors Q would be a block diagonal matrix with p diagonal blocks $Q_k, k = 1, ..., p$. For example, if one applies this approach to Gauß-Seidel, Q_k are lower triangular matrices (we call this particular smoother hybrid Gauß-Seidel; it has also been referred to as Processor Block Gauß-Seidel [3]). While this approach is easy to implement, it has the disadvantage of being more similar to a block Jacobi method, albeit worse, since the block systems are not solved exactly. Block Jacobi methods can converge poorly or even diverge unless used with a suitable damping parameter. Additionally, this approach is not scalable, since the number of blocks increases with the number of processors and with it the number of iterations increases. In spite of this, good results can be achieved by setting $Q = (1/\omega)\tilde{Q}$ and choosing a suitable relaxation parameter ω. Finding good parameters is not easy and made even harder by the fact that in a multilevel scheme one deals with a new system on each level, which requires new parameters. It is therefore important to find an automatic procedure to evaluate these parameters. Such a procedure has been developed for symmetric positive problems and smoothers in [57] using convergence theory for regular splittings. A good smoothing parameter for a positive symmetric matrix A is $\omega = 1/\lambda_{max}(\tilde{Q}^{-1}A)$, where $\lambda_{max}(M)$ denotes the maximal eigenvalue of M. A good estimate for this value can be obtained by using a few relaxation steps of Lanczos or conjugate gradient preconditioned with $\tilde{Q}$. This procedure can be applied to any symmetric positive definite hybrid smoother, such as hybrid symmetric Gauß-Seidel, Jacobi, Schwarz smoothers or symmetric positive definite variants of sparse approximate inverse or incomplete Cholesky smoothers.

5.3 Multicoloring Approaches

Another approach to parallelize Gauß-Seidel or similar smoothers such as block Gauß-Seidel or multiplicative Schwarz smoothers is to color subsets of points in such a way that subsets of the same color are independent of each other and can be processed in parallel. There are various parallel coloring algorithms available [30], however use of those as smoothers has shown to be inefficient, since often they generate too many colors, particularly on the coarser levels. Another approach is to color the processors, which will give some parallelism, but is overall not very efficient, since it leads to a large number of processors being idle most of the time.

One truly efficient implementation of a multicolor Gauß-Seidel method is described in [2, 3]. Here the nodes are ordered in such a way that interior nodes are processed while waiting for communication necessary to process boundary nodes. The algorithm first colors the processors and uses an ordering of the colors to receive an ordering of the processors. Each processor partitions its nodes into interior and boundary nodes, which are further divided into boundary nodes that require only communication with higher processors, those that communicate only with lower processors and the remaining boundary nodes. Interior nodes are also divided into smaller sets which are determined by taking into account computational cost to ensure that updates of boundary points occur at roughly the same time and thus idle time is minimized. For further details see [2].

5.4 Polynomial Smoothers

Another very parallel approach which promises to be also scalable is the use of polynomial smoothers. Here Q^{-1} in (13) is chosen to be a polynomial $p(A)$ with

$$p(A) = \sum_{0 \leq j \leq m} \alpha_j A^j. \tag{14}$$

To use this iteration as a multigrid smoother, the error reduction properties of $q(A) = I - p(A)A$ must be complementary to those of the coarse grid correction. One way to achieve this is by splitting the eigenvalues of A into low energy and high energy groups. The ideal smoother is then given by a Chebyshev polynomial that minimizes over the range that contains the high energy eigenvalues subject to the constraint $q(0) = 1$. If one knows the two eigenvalues that define the range, it is easy to compute the coefficients of the polynomials via a simple recursion formula.

Another polynomial smoother is the MLS (multilevel smoother) polynomial smoother. It is based on a combined effect of two different smoothing procedures, which are constructed to complement each other on the range of coarse grid correction. The two procedures are constructed so that their error propagation operators have certain optimum properties. The precise details concerning this polynomial can be found in [10] where this smoother was first developed in conjunction with the smoothed aggregation method.

The advantage of these methods is that they are completely parallel and their parallelism is independent of the number of processors. The disadvantage is that they require the evaluation of eigenvalues. Both require the maximal eigenvalue of a matrix and the Chebyshev polynomial smoother also the lower range of the high frequency eigenvalues. However, if these smoothers are used in the context of smoothed aggregation, which is usually the case, the maximal eigenvalues are already available since they are needed for the smoothing of the interpolation. Further details on this topic can be found in [3].

5.5 Approximate Inverse, Parallel ILU and More

Obviously, one can use any other parallel solver or preconditioner as a smoother. Q^{-1} can be chosen as any approximation to A^{-1}, e.g. a sparse approximate inverse. The use of approximate inverses in the context of multilevel methods is described in [51, 13, 38]. ParaSails, a very efficient parallel implementation of an approximate inverse preconditioner, approximates Q by minimizing the Frobenius norm of $I - QA$ and uses graph theory to predict good sparsity patterns for Q [15, 16]. Other options for smoothers are incomplete LU factorizations with $Q = \tilde{L}\tilde{U}$, where $\tilde{L}$ and $\tilde{U}$ are sparse approximations of the actual lower and upper triangular factors of A. There are various good parallel implementations available such as Euclid [28, 29], which obtains scalable parallelism via local and global reorderings, or PILUT [32], a parallel ILUT factorization. It is also possible to use conjugate gradient as a smoother.

6 Numerical Results

This section gives a few numerical results to illustrate some of the effects described in the previous sections. We apply various preconditioned AMG methods to 2-dimensional (2D) and 3-dimensional discretizations (3D) of the Laplace equation

$$-\Delta u = f, \tag{15}$$

with homogeneous Dirichlet boundary conditions on a unit square or unit cube. We use the codes BoomerAMG and MLI from the *hypre* library. BoomerAMG is mostly built on the "classical" AMG method and provides the coarsening algorithms: RS0, RS3, CLJP, Falgout, PMIS and HMIS. It deals with a slowdown in coarsening by switching to CLJP, and uses Gaussian elimination on the coarsest level, which is at most of size 9. It uses a slightly modified form of the "classical" interpolation described in Section 4 [25, 19]. MLI is an aggregation code. It uses the fast parallel direct solver SuperLU [50] on the coarsest level, which in our experiments is chosen to be at least 1024, the number of processors in our largest test run, and smaller than 4100. MLI's coarsening strategy is decoupled aggregation. We consider two variants: aggregation (AG), which uses the unsmoothed interpolation operator $(P^k)^{(0)}$ in (10), and smoothed aggregation (SA), which applies one step of smoothed Jacobi to $(P^k)^{(0)}$. For all methods, hybrid symmetric Gauß-Seidel smoothing was used. While RS3, Falgout and CLJP work almost as well as stand-alone solvers for the first two test problems, all other methods are significantly improved when accelerated by a Krylov method. Therefore in all of our experiments, they were used as preconditioners for GMRES(10).

All test problems were run on the Linux cluster MCR at Lawrence Livermore National Laboratory. We use the following notations in the tables:

- p: number of processors,
- θ: strength threshold as defined in (2) for RS0, RS3, CLJP, Falgout, PMIS and HMIS, as defined in (3) for AG and SA, (Section 3),
- τ: interpolation truncation factor (Section 4),
- C_{op}: operator complexity (Section 2),
- s_{avg}: maximal average stencil size, i.e. $\max_{1\leq k\leq M} s_(A^k)$, (Section 2),
- $\#its$: number of iterations,
- t_{setup}: setup time in seconds,
- t_{solve}: time of solve phase in seconds,
- t_{total}: total time in seconds.

The first test problem is a 2D Laplace problem with a 9-point discretization on a unit square. We kept the number of grid points fixed at 122,500 (350×350) on each processor, while increasing the number of processors and the overall problem size. The setup and total times for runs using 1, 4, 16, 64, 256 and 1024 processors are presented in Figure 6. For the aggregation based solvers AG and SA, $\theta = 0$ was chosen. For almost all other solvers, $\theta = 0.25$ was used, since this choice led to the best performance. RS0, however, performed significantly better with $\theta = 0$. Table 1 contains complexities and numbers of iterations for the 4 processor and 1024 processor runs. It shows that operator complexities are constant across an increasing number of processors, while stencil sizes increase for RS0, RS3, Falgout and SA. The aggregation methods obtain the best operator complexities, while CLJP's operator complexities

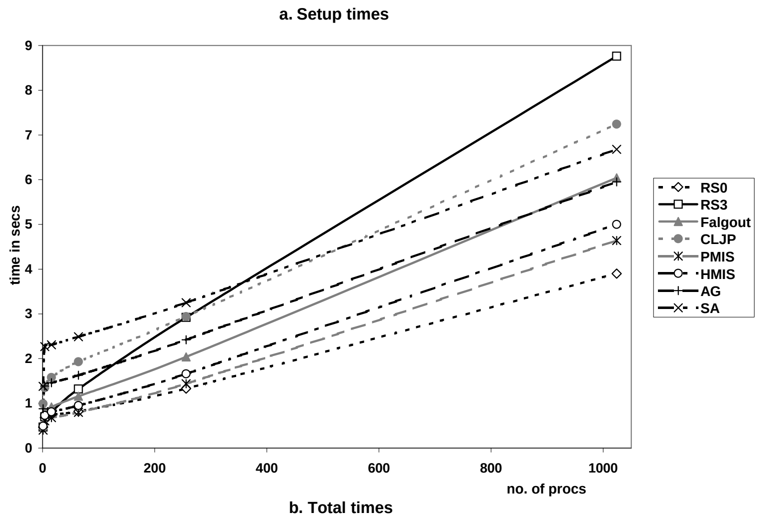
	$p = 4$			$p = 1024$		
Method	C_{op}	s_{avg}	$\#its$	C_{op}	s_{avg}	$\#its$
RS0	1.33	11	14	1.33	18	172
RS3	1.34	20	6	1.36	69	7
Falgout	1.34	16	6	1.35	32	8
CLJP	1.92	32	8	1.96	36	9
PMIS	1.25	11	24	1.23	11	42
HMIS	1.33	9	10	1.33	12	19
AG	1.13	9	37	1.13	9	93
SA	1.13	9	10	1.13	27	21

Table 1. Complexities and numbers of iterations for GMRES(10) preconditioned with various AMG methods.

are the highest. The overall best stencil sizes are obtained by AG, HMIS and PMIS. The most significant growth in stencil size occurs for RS3, apparently caused by the addition of the third pass on the boundaries. The effect of this stencil growth can clearly be seen in Figure 6a. AG and RS0 have the largest number of iterations, showing the influence of the decoupled coarsening. It also turns out that increasing the strength threshold for RS0 leads quickly to disastrous convergence results: for $\theta = 0.25$, $p = 1024$, RS0 needs more than 700 iterations to converge. While SA also uses decoupled coarsening, its interpolation operator is significantly improved by the use of one weighted Jacobi iteration, leading to faster convergence.

The results in Figure 6 show that the overall fastest method for this problem is Falgout, closely followed by HMIS and CLJP. While RS3 has slightly better convergence, the fact that its average stencil sizes increase faster with increasing number of processors than those of the other coarsenings, leads to faster increasing setup times and worse total times than Falgout, HMIS and CLJP. The smallest setup time is obtained by RS0, which requires no communication during the coarsening phase, followed by PMIS and HMIS, which benefit from their small stencil sizes. SA's setup times are larger than those of AG due to the smoothing of the interpolation operator. Note that while the results for the aggregation methods demonstrate good complexities, and thus low memory usage, they are not necessarily representative of smoothed aggregation in general or other implementations of this method. The use of more sophisticated parallel aggregation schemes, a different treatment of the coarsest level and the use of other smoothers might yield better results.

The second test problem is a 7-point finite difference discretization of the 3-dimensional Laplace equation on a unit cube using $40 \times 40 \times 40$ points per processor. For three-dimensional test problems, it becomes more important to consider complexity issues. While the choice of $\theta = 0.25$ often still leads to best convergence, complexities can become so large for this choice that setup times and memory requirements might be unreasonable. The effect of using different strength thresholds on complexities and convergence is illus-



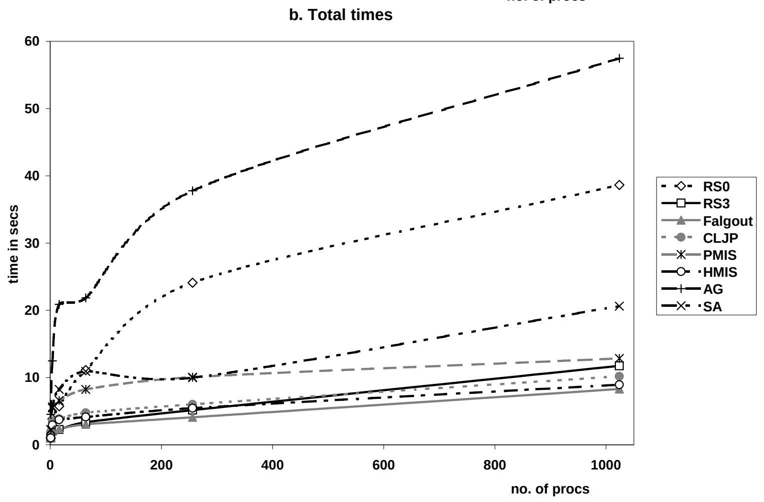


Fig. 6. Setup and total times for a 2-dimensional Laplacian problem with a 9-point stencil for an increasing number of processors and increasing problem size with a fixed number of grid points (350×350) per processor.

trated in Table 2 for Falgout-GMRES(10) for 288 processors. Based on the results of this experiment we are choosing $\theta = 0.75$ for our further experiments. Another useful tool to decrease stencil size is interpolation operator truncation. Increasing the truncation factor τ also decreases s_{avg}, while only slightly decreasing C_{op}. Best timings when choosing $\theta = 0$ and increasing τ were achieved for $\tau = 0.3$. In general, this choice leads to larger stencil sizes than choosing $\theta = 0.75$ and $\tau = 0$, but better convergence, as can be seen in Table 3. When increasing the number of processors and with it the overall problem size, operator complexities and stencil sizes initially increase notice-

θ	C_{op}	s_{avg}	$\#its$	t_{setup}	t_{solve}	t_{total}
0.00	3.26	458	6	21.82	2.67	24.49
0.25	6.32	3199	5	96.77	7.93	104.70
0.50	5.32	538	7	18.60	5.97	24.57
0.75	6.08	232	10	11.05	6.73	17.78

Table 2. Effect of strength threshold on Falgout-GMRES(10) for a 7-point finite difference discretization of the 3-dimensional Laplacian on 216 processors with 40^3 degrees of freedom per processor.

Method	θ	τ	C_{op}	s_{avg}	$\#its$	t_{setup}	t_{solve}	t_{total}
RS0	0.50	0.0	3.70	170	14	6.69	4.67	11.36
RS3	0.75	0.0	6.18	457	9	19.35	5.83	25.18
	0.00	0.3	3.52	756	6	42.83	5.69	48.52
CLJP	0.75	0.0	13.38	119	16	14.94	13.23	28.17
	0.00	0.3	4.46	232	9	17.94	2.80	20.74
Falgout	0.75	0.0	6.12	237	10	13.79	5.88	19.67
	0.00	0.3	3.22	275	7	19.14	2.56	21.70
PMIS	0.00	0.0	2.09	49	20	4.69	4.29	8.98
HMIS	0.00	0.0	2.75	61	13	5.15	3.44	8.59
AG	0.08	0.0	1.25	16	36	3.22	11.74	14.96
SA	0.08	0.0	1.75	172	13	4.96	5.90	10.86

Table 3. AMG-GMRES(10) with different coarsening strategies applied to a 7-point finite difference discretization of the 3-dimensional Laplacian on 512 processors with 40^3 degrees of freedom per processor.

ably, particularly for RS3, Falgout and CLJP. They, however, cease growing for large problem sizes. The complexities presented in Table 3 are therefore close to those that can be obtained for larger problem sizes. Operator complexities are overall larger than in the previous test problem, particularly for the CLJP coarsening, when no interpolation truncation is applied. Best operator complexities are obtained for the aggregation based schemes, followed by PMIS and HMIS. The best overall timings are achieved by HMIS. RS0 performs very well for this problem for this particular parameter choice, however if applied to the 7-point 3D problem using 39^3 instead of 40^3 unknowns per processor, $p = 1024$, C_{op} is twice, $\#its$ is four times and t_{total} is three times as large as the values reported in Table 3, while the other coarsenings are affected to a much lesser degree by the change in system size.

Finally, we present results for the 3D Laplace problem on the unit cube using an unstructured finite element discretization. Note that operator complexities are overall lower here. RS0 is performing the worst due to a large number of iterations. The best timings are achieved by PMIS, followed by HMIS and SA. Interestingly enough, while CLJP's operator complexities were much larger than Falgout's and convergence was slightly worse for the struc-

Method	θ	C_{op}	s_{avg}	$\#its$	t_{setup}	t_{solve}	t_{total}
RS0	0.75	2.51	47	86	3.55	15.53	19.08
RS3	0.75	2.65	56	41	4.87	11.69	16.56
CLJP	0.75	2.71	76	18	5.79	6.33	12.12
Falgout	0.75	2.84	77	21	6.42	7.20	13.62
PMIS	0.25	1.46	53	22	2.41	3.60	6.01
HMIS	0.25	1.60	61	21	2.91	3.64	6.55
AG	0.00	1.06	18	54	2.13	12.22	14.35
SA	0.00	1.24	102	18	2.73	5.68	8.61

Table 4. AMG-GMRES(10) with different coarsening strategies applied to an unstructured finite element discretization of the 3-dimensional Laplacian on 288 processors with approx. 20,000 degrees of freedom per processor.

tured test problems, it performs slightly better than Falgout for this truly unstructured problem. A similar effect can be observed for HMIS and PMIS. Results for a variety of test problems applied to various coarsening schemes in BoomerAMG can be found in [25, 19]. Numerical test results for various elasticity problems comparing BoomerAMG and MLI can be found in [9].

7 Software Packages

There are various software packages for parallel computers which contain algebraic multilevel methods. This section contains very brief descriptions of these codes. Unless specifically mentioned otherwise, these packages are open source codes, and information on how to obtain them is provided below.

7.1 hypre

The software library *hypre* [23] is being developed at Lawrence Livermore National Laboratory and can be downloaded from [27]. One of its interesting features are its conceptual interfaces [23, 21], which are described in further detail in Chapter **??**. It contains various multilevel preconditioners, including the geometric multigrid codes SMG and PFMG [20],the AMG code BoomerAMG and the smoothed aggregation code MLI, as well as sparse approximate inverse and parallel ILU preconditioners.

7.2 LAMG

LAMG is a parallel algebraic multigrid code, which has been developed at Los Alamos National Laboratory. It makes extensive use of aggressive coarsening, such as described in [49]. It has shown to give extremely scalable results on upto 3500 processors. Further information on the techniques used in LAMG can be found in [31]. LAMG is not an open source code.

7.3 ML

ML [26] is a massively parallel algebraic multigrid solver library for sparse linear systems. It contains various parallel multigrid methods, including smoothed aggregation, a version of classic AMG and a special algebraic multigrid solver for Maxwell's equations. The smoothed aggregation code offers all the parallel coarsening options for aggregation based AMG described above and among other features polynomial and multi-colored Gauß-Seidel smoothers. It is being developed at Sandia National Laboratories and can be downloaded from [41].

7.4 pARMS

The library pARMS contains parallel algebraic recursive multilevel solvers. These solvers are not algebraic multigrid methods in the sense described above, but are algebraic multilevel solvers which rely on a recursive multi-level ILU factorization. Further details on these methods can be found in [47, 34] The pARMS library has been developed at the University of Minnesota and is available at [42].

7.5 PEBBLES

PEBBLES (Parallel and Element Based grey Box Linear Equation solver) is an algebraic multigrid package for solving large sparse, symmetric positive definite linear equations which arise from finite element discretizations of elliptic PDEs of second order [24]. It is a research code with element preconditioning, different interpolation and coarsening schemes and more that is being developed at the University in Linz. More information on the package and how to obtain the code is available at [43].

7.6 PHAML

PHAML (Parallel Hierarchical Adaptive Multilevel solvers) is a parallel code for the solution of general second order linear self-adjoint elliptic PDEs. It uses a finite element method with linear, quadratic or cubic elements over triangles. The adaptive refinement and multigrid iteration are based on a hierarchical basis formulation [40]. For further information and to download the software package which is being developed at the National Institute for Science and Technology (NIST) see [44].

7.7 Prometheus

Prometheus is a parallel multigrid library that has been developed originally at the University of California at Berkeley. It contains a multigrid solver for PDE's on finite element generated unstructured grids, but also a smoothed aggregation component named ATLAS. Prometheus is available at [45].

7.8 SAMGp

SAMGp (Algebraic Multigrid Methods for Systems) is a parallel software library which has been developed at Fraunhofer SCAI. It uses subdomain blocking as a parallel coarsening scheme [33] and allows for various different coarsening schemes, such as aggressive coarsening, making it very memory efficient. SAMGp is a commercial code. Information on how to purchase it as well as a user's manual can be found at [48].

7.9 SLOOP

A parallel object oriented library for solving sparse linear systems named SLOOP [18, 39] is being developed at CEA (Commisariat a l'Energie Atomique) in France. SLOOP's primary goal is to provide a friendly user interface to the parallel solvers and ease the integration for new preconditioners and matrix structures. It contains various parallel preconditioners: algebraic multigrid, approximate inverse and incomplete Cholesky. This library is currently not an open source code, but there are plans to change this in the near future.

7.10 UG

UG (Unstructured Grids) is a parallel software package that is being developed at the University of Heidelberg particularly for the solution of PDEs on unstructured grids [4]. It has various features, including a parallel AMG code, adaptive local grid refinement and more. For further information and a copy of the code see [54].

8 Conclusions and Future Work

Overall, there are many efficient parallel implementations of algebraic multigrid and multilevel methods. Various parallel coarsening schemes, interpolation procedures, parallel smoothers as well as several parallel software packages have been briefly described. There has truly been an explosion of research and development in the area of algebraic multilevel techniques for parallel computers with distributed memories. Even though we have tried to cover as much information as possible, there are still various interesting approaches that have not been mentioned. One of those approaches that shows a lot of promise is the concept of compatible relaxation. This was originally suggested by Achi Brandt [6]. Much research has been done in this area. Although many theoretical results have been obtained [22, 35], we are not aware of an efficient implementation of this algorithm to this date. However, once this has been formulated, compatible relaxation holds much promise for parallel computation. Since the smoother is used to build the coarse grid, use of a completely parallel smoother (e.g. C-F Jacobi relaxation) will lead to a parallel coarsening algorithm.

References

1. Adams, M.: A parallel maximal independent set algorithm. In *Proceedings of the 5th Copper Mountain Conference on Iterative Methods*, (1998)
2. Adams, M.: A distributed memory unstructured Gauss-Seidel algorithm for multigrid smoothers. In *ACM/IEEE Proceedings of SC2001: High Performance Networking and Computing*, (2001)
3. Adams, M., Brezina, M., Hu, J., and Tuminaro, R.: Parallel multigrid smoothing: polynomial versus Gauss-Seidel. *Journal of Computational Physics*, 188:593–610, (2003)
4. Bastian, P., Birken, K., Johannsen, K., Lang, S., n. Neuß, Rentz-Reichert, H., and Wieners, C.: UG: a flexible software toolbox for solving partial differential euations. *Computing and Visualization in Science*, 1:27–40, (1997)
5. Brandt, A.: Algebraic multigrid theory: The symmetric case. *Appl. Math. Comp.*, 19:23–56, (1986)
6. Brandt, A.: General highly accurate algebraic coarsening schemes. *Electronic Transactions on Numerical Analysis*, 10:1–20, (2000)
7. Brandt, A., McCormick, S., and Ruge, J.: Algebraic multigrid (AMG) for automatic multigrid solutions with application to geodatic computations. Technical report, Institute for Computational Studies, Fort Coolins, CO, (1982)
8. Brandt, A., McCormick, S., and Ruge, J.: Algenbraic multigrid (AMG) for sparse matrix equations. In Evans, D., editor, *Sparsity and Its Applications.* Cambridge University Press, (1984)
9. Brezina, M., , Tong, C., and Becker, R.: Parallel algebraic multigrids for structural mechanics. *SIAM Journal of Scientific Computing*, submitted, (2004). Also available as LLNL technical report UCRL-JRNL-204167
10. Brezina, M.: Robust iterative solvers on unstructured meshes. Technical report, University of Colorado at Denver, (1997). Ph.D.thesis
11. Brezina, M., Cleary, A. J., Falgout, R. D., Henson, V. E., Jones, J. E., Manteuffel, T. A., McCormick, S. F., and Ruge, J. W.: Algebraic multigrid based on element interpolation (AMGe). *SIAM J. Sci. Comput.*, 22(5):1570–1592, (2000). Also available as LLNL technical report UCRL-JC-131752
12. Briggs, W., Henson, V., and McCormick, S.: *A multigrid tutorial.* SIAM, Philadelphia, PA, (2000)
13. Bröker, O. and Grote, M.: Sparse approximate inverse smoothers for geometric and algebraic multigrid. *Applied Numerical Mathematics*, 41:61–80, (2002)
14. Chartier, T., Falgout, R. D., Henson, V. E., Jones, J., Manteuffel, T., McCormick, S., Ruge, J., and Vassilevski, P.: Spectral AMGe (ρAMGe). *SIAM Journal on Scientific Computing*, 25:1–26, (2003)
15. Chow, E.: A priori sparsity patterns for parallel sparse approximate inverse preconditioners. *SIAM J. Sci. Comput.*, 21(5):1804–1822, (2000). Also available as LLNL Technical Report UCRL-JC-130719 Rev.1
16. Chow, E.: Parallel implementation and practical use of sparse approximate inverses with a priori sparsity patterns. *Int'l J. High Perf. Comput. Appl.*, 15:56–74, (2001). Also available as LLNL Technical Report UCRL-JC-138883 Rev.1
17. Cleary, A. J., Falgout, R. D., Henson, V. E., and Jones, J. E.: Coarse-grid selection for parallel algebraic multigrid. In *Proc. of the Fifth International Symposium on: Solving Irregularly Structured Problems in Parallel*, volume 1457 of

Lecture Notes in Computer Science, pages 104–115, New York, (1998). Springer–Verlag. Held at Lawrence Berkeley National Laboratory, Berkeley, CA, August 9–11, 1998. Also available as LLNL Technical Report UCRL-JC-130893
18. Colombet, L., Meurant, G., et al.: Manuel utilisateur de la bibliotheque (SLOOP) 3.2 SLOOP 3.2 users manual. Technical report, CEA/DIF/DSSI/SNEC, (2004)
19. De Sterck, H., Yang, U. M., and Heys, J.: Reducing complexity in parallel algebraic multigrid preconditioners. *SIAM Journal on Matrix Analysis and Applications*, submitted, (2004). Also available as LLNL technical report UCRL-JRNL-206780
20. Falgout, R. D. and Jones, J. E.: Multigrid on massively parallel architectures. In Dick, E., Riemslagh, K., and Vierendeels, J., editors, *Multigrid Methods VI*, volume 14 of *Lecture Notes in Computational Science and Engineering*, pages 101–107, Berlin, (2000). Springer. Proc. of the Sixth European Multigrid Conference held in Gent, Belgium, September 27-30, 1999. Also available as LLNL technical report UCRL-JC-133948
21. Falgout, R. D., Jones, J. E., and Yang, U. M.: Conceptual interfaces in hypre. *Future Generation Computer Systems*, to appear, (2003). Also available as LLNL technical report UCRL-JC-148957
22. Falgout, R. and Vassilevski, P.: On generalizing the AMG framework. *SIAM Journal on Numerical Analysis*, to appear, (2003). Also available as LLNL technical report UCRL-JC-150807
23. Falgout, R. D. and Yang, U. M.: *hypre*: a library of high performance preconditioners. In Sloot, P., Tan., C., Dongarra, J., and Hoekstra, A., editors, *Computational Science - ICCS 2002 Part III*, volume 2331 of *Lecture Notes in Computer Science*, pages 632–641. Springer–Verlag, (2002). Also available as LLNL Technical Report UCRL-JC-146175
24. Haase, G., Kuhn, M., and Reitzinger, S.: Parallel algebraic multigrid methods on distributed memory computers. *SIAM Journal on Scientific Computing*, 24:410–427, (2002)
25. Henson, V. E. and Yang, U. M.: BoomerAMG: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41:155–177, (2002). Also available as LLNL technical report UCRL-JC-141495
26. Hu, J., Tong, C., and Tuminaro, R.: ML 2.0 smoothed aggregation user's guide. Technical Report SAND2001-8028, Sandia National Laboratories, (2002)
27. hypre: High performance preconditioners. http://www.llnl.gov/CASC/hypre/
28. Hysom, D. and Pothen, A.: Efficient parallel computation of ILU(k) preconditioners. In *Proceedings of SuperComputing 99*. ACM, (1999). published on CDROM, ISBN #1-58113-091-0, ACM Order #415990, IEEE Computer Society Press Order # RS00197
29. Hysom, D. and Pothen, A.: A scalable parallel algorithm for incomplete factor preconditioning. *SIAM J. Sci. Comput.*, 22(6):2194–2215, (2001)
30. Jones, M. and Plassman, P.: A parallel graph coloring heuristic. *SIAM J. Sci. Comput.*, 14:654–669, (1993)
31. Joubert, W. and Cullum, J.: Scalable algebraic multigrid on 3500 processors. Technical Report Technical Report No. LAUR03-568, Los Alamos National Laboratory, (2003)
32. Karpis, G. and Kumar, V.: Parallel threshold-based ILU factorization. Technical Report 061, University of Minnesota, Department of Computer Science/Army HPC Research Center, Minneapolis, MN 5455, (1998)

33. Krechel, A. and Stüben, K.: Parallel algebraic multigrid based on subdomain blocking. *Parallel Computing*, 27:1009–1031, (2001)
34. Li, Z., Saad, Y., and Sosonkina, M.: pARMS: a parallel version of the algebraic recursive multilevel solver. *Numerical Linear Algebra with Applications*, 10:485–509, (2003)
35. Livne, O.: Coarsening by compatible relaxation. *Numerical Linear Algebra with Applications*, 11:205–228, (2004)
36. Luby, M.: A simple parallel algorithm for the maximal independent set problem. *SIAM J. on Computing*, 15:1036–1053, (1986)
37. McCormick, S. F.: *Multigrid Methods*, volume 3 of *Frontiers in Applied Mathematics*. SIAM Books, Philadelphia, (1987)
38. Meurant, G.: A multilevel AINV preconditioner. *Numerical Algorithms*, 29:107–129, (2002)
39. Meurant, G.: Numerical experiments with parallel multilevel preconditioners on a large number of processors. *SIAM Journal on Matrix Analysis and Applications*, submitted, (2004)
40. Mitchell, W.: Unified multilevel adaptive finite element methods for elliptic problems. Technical Report UIUCDCS-R-88-1436, Department of Computer Science, University of Illinois, Urbana, IL, (1988). Ph.D. thesis
41. ML: A massively parallel algebraic multigrid solver library for solving sparse linear systems. http://www.cs.sandia.gov/ tuminaro/ML_Description.html
42. pARMS: Parallel algebraic recursive multilevel solvers. http://www-users.cs.umn.edu/ saad/software/pARMS/
43. PEBBLES: Parallel and elment based grey box linear equation solver. http://www.numa.uni-linz.ac.at/Research/Projects/pebbles.html
44. PHAML: The parallel hierarchical adaptive multilevel project. http://math.nist.gov/phaml/
45. Prometheus. http://www.cs.berkeley.edu/ madams/prometheus/
46. Ruge, J. W. and Stüben, K.: Algebraic multigrid (AMG). In McCormick, S. F., editor, *Multigrid Methods*, volume 3 of *Frontiers in Applied Mathematics*, pages 73–130. SIAM, Philadelphia, PA, (1987)
47. Saad, Y. and Suchomel, B.: ARMS: an algebraic recursive multilevel solver for general sparse linear systems. *Numerical Linear Algebra with Applications*, 9:359–378, (2002)
48. SAMGp: Algebraic multigrid methods for systems. http://www.scai.fraunhofer.de/samg.htm
49. Stüben, K.: Algebraic multigrid (AMG): an introduction with applications. In Trottenberg, U., Oosterlee, C., and Schüller, A., editors, *Multigrid*. Academic Press, (2001)
50. SuperLU. http://acts.nersc.gov/superlu/
51. Tang, W.-P. and Wan, W. L.: Sparse approximate inverse smoother for multigrid. *SIAM J. Matrix Anal. Appl.*, 21:1236–1252, (2000)
52. Trottenberg, U., Oosterlee, C., and Schüller, A.: *Multigrid*. Academic Press, (2001)
53. Tuminaro, R. and Tong, C.: Parallel smoothed aggregation multigrid: aggregation strategies on massively parallel machines. In Donnelley, J., editor, *Supercomputing 2000 Proceedings*, (2000)
54. UG: A flexible software toolbox for solving partial differential equations. http://cox.iwr.uni-heidelberg.de/ ug/index.html

55. Vanek, P., Brezina, M., and Mandel, J.: Convergence of algebraic multigrid based on smoothed aggregation. *Numerische Mathematik*, 88:559–579, (2001)
56. Vaněk, P., Mandel, J., and Brezina, M.: Algebraic multigrid based on smoothed aggregation for second and fourth order problems. *Computing*, 56:179–196, (1996)
57. Yang, U. M.: On the use of relaxation parameters in hybrid smoothers. *Numerical Linear Algebra with Applications*, 11:155–172, (2004)