PARALLEL ALGEBRAIC MULTIGRID

CARLOS E. SAUER*

1. Introduction

Many different real-life phenomena are modeled by Partial Differential Equations in order to reproduce the real system in a mathematical framework. Often, numerical solutions for such models involve discretizations of the domain and the operators, reducing the problem to solve a system of linear equations $Ax = b$. As the size of the matrix A becomes larger, direct methods are no longer useful in practice. In such cases, iterative methods become very important and in this project I will focus on one of them: the *Algebraic Multigrid Method*.

In contrast to Geometric Multigrid, which needs the operator and domain to be discretized in coarser grids, the Algebraic Multigrid (AMG) approach is only based on the entries of A, like a black-box solver of $Ax = b$. While the Geometric Multigrid relies on a fixed hierarchy of grids, the Algebraic Multigrid adjusts the hierarchy to maintain fixed and simple smoothers on each step [1]. Additionally, the AMG algorithm has the advantage that it can be used for systems of linear equations that do not arise from differential equations [2].

In this project, I review the classical (sequential) AMG algorithm. An overview is shown in sections 2, 3 and 4. All components, smoothing, influences, interpolation weights and coarsening are analized in detail in sections 5 to 8. In section 9, the classical Ruge-Stueben sequential coarsening is described. Different strategies to implement the coarsening pre-processing step in parallel is then be described in section 10. The Coarse Grid Classification described in section 11, seems to avoid most of the disadvantages of the other parallel approaches and reveals itself as one of the best known methods for parallel coarsening.

2. Basics of Multigrid Methods

Given a PDE $\mathcal{L}u = g$ on a domain Ω, a suited discretization of size h reduces to problem to solve a linear system of equations $A_h x_h = b_h$. On one side, iterative methods work faster with better initial guesses. On the other side, coarser discretizations of the domain Ω generate smaller systems of equations and are thus much easier to handle. Multigrid methods take advantage of both: use information from coarser discretizations to improve the solution in finer discretizations.

Let $A_{2h}x_{2h} = b_{2h}$ be the discretized version of the PDEof size $2h$. There is no need to solve the problem completely in the lower level. Indeed, a rough approximation might be enough.

*Massachusetts Institute of Technology, 77 Mass Avenue, Cambridge, MA 02139 (csauer@mit.edu).

2.1. Geometric versus Algebraic Multigrid

In the previous section, the notion of geometric multigrid method was described. However, the Algebraic Multigrid (AMG) method is a similar one in which there is no PDE and no physical domain and the only given information is a system $A_h x_h = b_h$. Whenever the geometry of the problem is known, coarser discretizations are easy to generate.

In algebraic multigrid, coarser versions of $A_h x_h = b_h$ must be generated carefully. Note that it is not enough to delete a few rows and columns, because vital couplings between entries will be lost. This reveals the key importance of coarsening in the Algebraic Multigrid case and this project will give this step a high priority. From now on, we will only focus on AMG.

3. Key multigrid components

The basic theory of AMG assumes that A_h is a symmetric positive definite M-matrix, i.e., the diagonal is positive, the non-diagonal entries are non-positive and the sum of each row is also non-positive. Although this might be seen as a strong assumption, usually AMG also works reasonably well for non M-matrices.

We should define some key components for any multigrid algorithm. Given a system $A_h x_h = b_h$ of size N, we need to construct abstract *"grids"* $\Omega^1 \supset \Omega^2 \supset \cdots \supset \Omega^M$, where $\Omega^1 = \{1,2,\ldots,N\}$ represents all variables of x_h, and Ω^M is the coarsest version. For visualization purposes, the set Ω^1 can be thought as uniformly distributed points on an interval.

In order to interchange information between grids, we need *transfer operators*. The *interpolation* operator I_{k+1}^k from coarser grid Ω^{k+1} to finer grid Ω^k will be defined as a linear comination of the values at the coarse grid points. The coefficients of the linear combination will rely heavily on the coarsening process and thus we will define them later in sections 7 and 8. The *restriction* operator I_k^{k+1} from finer grid Ω^k to coarser grid Ω^{k+1} for each k is simply the transpose of the interpolation operator.

We also define *grid operators*, A^1, A^2,...A^M, which represent the action of the original matrix $A^1 = A_h$ on the different grid levels (notation is changed to avoid the size h of geometric multigrid). The classical way is to use a so called Galerkin approach by defining: $A^{k+1} = I_k^{k+1} A^k I_{k+1}^k$

Finally, we need to use a relaxation scheme at each step. This is an iterative method like of a Gauss-Seidel or relaxed Jacobi scheme.

4. Two-cycle structure

Assuming these components were all defined and available, a simple pseudocode of a Two-cycle is given by [3]:

- **Algorithm: TWOAMG(u,b,ν_1,ν_2)**
- Pre-smoothing: With initial guess u, perform ν_1 relaxation iterations on $A^1u=b$ obtaining a new u.

- Coarse grid correction: Restrict the residual $r \equiv I_1^2(b-A^1u)$, and relax ν_2 times on $A^2e=r$ with initial guess zero, obtaining some e.

- Prolongation of the error e using the interpolation operator and correct the solution by: $u \leftarrow u+I_2^1e$.

Note that instead of transfering the solution between grids, we restric the residual and interpolate the error. This algotithm should be iterated until convergence, improving the initial guess at each iteration.

5. Relaxation: Gauss-Seidel method

In AMG the relaxation scheme is *fixed* and the preferred one is the classic Gauss-Seidel method. We consider the problem of solving $Ax=b$. Let A be written as $A=D-L-U$, where D is diagonal, L lower and U is upper triangular, with diagonal zero. The usual Gauss-Seidel iteration is written as[4]:

$$x^{r+1} = Gx^r + (D-L)^{-1}f \tag{5.1}$$

where $G=(I-(D-L)^{-1}A)$ is the Gauss-Seidel iteration matrix. Substracting from the exact solution x, we may write:

$$e^{r+1} = Ge^r \tag{5.2}$$

where $e^r = x^r - x$ is the iteration error. Since we do not know the initial error e^0 this equation is useless for computing the evolution of the error. However, it is useful deriving theoretical properties and results. One of this properties is related to the smoothness of the error. Methods like Gauss-Seidel and the under-relaxed Jacobi when applied to $Ax=b$ are called smoothers, in the sense that high frequency components of the error on a geometric grid decay much faster than lower ones.

5.1. Algebraic vs Geometric Smoothness

When the problem is purely algebraic and no grids are available, Fourier analysis is useless and we need to figure out a specific meaning for smooth errors. In fact, a given vector of values may appear to be geometrically oscillatory or not, depending upon the interval size of the chosen grid. In the algebraic context, an error is considered smooth if an iteration like 5.1 has almost no longer influence on the error. In other words, an error e is considered algebraically smooth if at some iteration step:

$$||Ge||_A \approx ||e||_A \tag{5.3}$$

where $||y||_A = \langle Ay, y\rangle^{1/2}$ is the norm induced by A. In [5], Stueben presents a simple example to show that algebraic smoothness do not need to be geometrically smooth: in fact, it might be quite oscillatory.

5.2. From smoothness to interpolation

Having defined a relaxation scheme, our goal is to construct coarse grids and interpolation operators. Thus, it might be interesting to use the algebraic notion of smoothness of the error, to design effective interpolation strategies. It can be shown [6], that (5.3) implies:

$$\langle D^{-1}r, r\rangle << \langle Ae, e\rangle$$

where $r = Ae$ is the residual. This can be written as:

$$\sum_i \frac{r_i^2}{a_{ii}} << \sum_i r_i e_i$$

Since this is valid for the sum, we might expect that such a relation is still valid comparing term to term. Therefore, at least in average for all i:

$$|r_i| << a_{ii}|e_i| \tag{5.4}$$

This is commonly interpreted by identifying smooth errors with small residuals, i.e. $Ae \approx 0$. This is a key assumption for the next step, in which the coarse grids and interpolation weights will be chosen. Furthermore, we may conclude this section by observing that $Ae \approx 0$ implies:

$$a_{ii}e_i \approx -\sum_{j\neq i} a_{ij}e_j \tag{5.5}$$

Notice that this means that smooth errors have the property that each component is aproximately a weighted average of the rest. This is not enough, but it gives a cue to begin the discussion about interpolation weights.

6. Influence: strongly and weakly

Although (5.5) relates the i-th component of the error to every other component, the weights depend on the entries of the i-th row of the matrix A, which in most applications is not dense. Therefore, we might expect that only a few components have a strong influence on e_i, depending on whether a_{ij} is relatively large or small with respect to other entries on his row.

We will say that the variable x_i *strongly* depends on the variable x_j if [6]:

$$-a_{ij} \geq \theta \max_{k\neq i}\{a_{ik}\} \tag{6.1}$$

where θ is a positive threshold value and is typically set to be 0.25 [3]. This value is reasonable since it corresponds to the threshold under which the 5-pt laplacian stencil

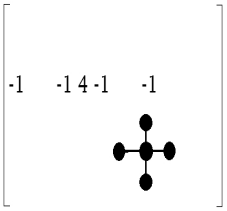

FIG. 6.1. **5-point Laplacian stencil and corresponding row in the matrix**

has its nodes strongly coupled (see Figure 6.1).

If (6.1) holds for some j, we say that $j \in S_i$. If (6.1) does not hold but $a_{ij} \neq 0$, we say that the variable x_i depends *weakly* on x_j and write, $j \in W_i$. For example, for non M-matrices, this definition implies that positive but off-diagonal entries only weakly influence the same row variable [7]. The sets S_i and W_i are called the set of strong and weak dependencies of variable x_i.

7. Coarsening and Interpolation

Assume we are at some level k of the typical multigrid process, where a grid Ω^k and a grid operator A^k are given and we want to choose a grid for the next lower level of the process. We are interested in splitting $\Omega^k = F \bigcup C$, where C will be called the coarse-grid points and F the fine grid points. The set C will be identified with our next grid Ω^{k+1}.

If we look to define interpolation operators from Ω^{k+1} to Ω^k, we only need to care about points in F, since values in C are directly transferred from Ω^{k+1}. Given any $i \in F$, we need to define not only which grid values in Ω^{k+1} influence the value at $i \in F$, but also how to weight these values, i.e:

$$(I_{k+1}^k e^{k+1})_i = \begin{cases} e_i^{k+1} & i \in C \\ \sum_{j \in C_i} \omega_{ij} e_j^{k+1} & i \in F \end{cases} \tag{7.1}$$

where $C_i = S_i \bigcap C$ is the set of points in Ω^{k+1} with strong influence in $i \in F$. Of course, there are also points in $F \bigcap S_i$ and $F \bigcap W_i$ which influence $i \in F$, at least weakly. Our goal is therefore to include all these dependencies in the weights ω_{ij}, so that (7.1) holds.

8. Interpolation Weights

We will assume first that the appropiate splitting $\Omega^k = F \bigcup C$ has been defined and we will try to obtain the interpolation operator as in (7.1). Ignoring the terms $a_{ij} = 0$, we can rewrite (5.5) as:

$$a_{ii} e_i \approx - \sum_{j \in C_i} a_{ij} e_j - \sum_{j \in F \bigcap S_i} a_{ij} e_j - \sum_{j \in F \bigcap W_i} a_{ij} e_j \tag{8.1}$$

The last two sums, which depend on fine-grid points, have to be rewritten to obtain an interpolation formula like (7.1). First, in $F \bigcap W_i$ we will approximate $e_i \approx e_j$ since

a_{ij} is relatively small [3], and the approximation (8.1) should still hold.

Next, we need to redistribute the influence of points in $F \bigcap S_i$, among the points in C_i. This requires an heuristic criterion for the step where we obtain C and F [3]:

- **Criterion 1:** for each $i \in F$, each $j \in S_i$ should be either in C_i or it should depend itself on at least one point of C_i, i.e. $S_j \bigcap C_i \neq \varnothing$

Therefore, we are allowed to rewrite each e_j for $j \in F \bigcap S_i$ as a weighted average of the corresponding elements $k \in C_i$ obtaining:

$$e_j \approx \frac{\sum_{k \in C_i} a_{jk} e_k}{\sum_{k \in C_i} a_{jk}} = \sum_{k \in C_i} \hat{a}_{jk} e_k \tag{8.2}$$

where $\hat{a}_{jk} = \frac{a_{jk}}{\sum_{s \in C_i} a_{js}}$. This sounds reasonable, since the dependence is directly related to the size of the a_{jk}'s. Thus, (8.1) becomes:

$$(a_{ii} + \sum_{k \in F \bigcap W_i} a_{ik}) e_i \approx - \sum_{j \in C_i} (a_{ij} + \sum_{k \in F \bigcap S_i} a_{ik} \hat{a}_{kj}) e_j$$

(for simplicity, there is a slightly interchange of index between j and k with respect to what they were in (8.2)). Therefore, we have obtained the appropiate interpolation weights which satisfy (7.1):

$$\omega_i j = - \frac{a_{ij} + \sum_{k \in F \bigcap S_i} a_{ik} \hat{a}_{kj}}{a_{ii} + \sum_{k \in F \bigcap W_i} a_{ik}} \tag{8.3}$$

9. Sequential Coarsening

As stated in the previous section, we will use Criterion 1 when choosing which points we will drop from the fine-grid in order to construct the coarse-grid points. It is important to point out that the coarse-grid points in C, are also fine-grid points. The notation F, is reserved for those points which are exclusively fine and not coarse-grid points. In general, I follow the classical guidelines described in [5].

Note that a coarse grid with too many points will need more work in this and lower levels. Although in such cases the interpolation of the error to the finer level will be done more accurately, we will prefer to control the size of C with an additional criterion. Indeed, we will consider the following heuristic criteria:

- **Criterion 2:** No C-point should depend strongly on another C-point.

The Criterion 1 has to be satisfied strictly, because our weights (8.3) depend strongly on this assumption. Having said this, the Criterion 2 will be applied whenever this does not affect the fulfillment of Criterion 1. This coarsening strategy is usually implemented in two steps.

1. **First Step:** In the first step, we try to satisfy the goal of Criterion 2: keep C as small as possible. Thus, points which have a strong influence on a large number of points are preferred as C-points. Let us define S_i^T to be the set of points which depend on i strongly as in (6.1). Considering the cardinality λ_i of those sets, we select the point with the largest λ_i as our first point of C.

   Next, by Criterion 2, all points in S_i^T are set to be in F. It is interesting to notice that those unassigned points which also strongly influence the new points in F may be important for accurate interpolations and they should more likely be in C than in F. This preference is implemented by incrementing $\lambda_j = \lambda_j + 1$ for each point j which strongly influences any F-point.

   This procedure is repeated with the unassigned points, by taking the point with the largest λ_i, and so on until all points are either in C or in F.

2. **Second Step:**[1] In the second step, we look for violations of the Criterion 1 and try to fix them by moving points from F to C, even if Criterion 2 is no longer valid.

   With this in mind we should test thoroughly the set F to make sure that each point $j \in F \bigcap S_i$ depends strongly on at least one point in C_i. In other words, if there are any strong connection between two F-points, these should share the at least one C-point which influences both strongly.

   Since we would like to move as few points as possible to keep our set C small, once we find a $j \in F \bigcap S_i$ for some $i \in F$, that has no strong dependence on C_i, we do not move j immediately to C, just tentatively. If this change also solves the problem for every other element in $F \bigcap S_i$, the change is made permanent. If another point also has no strong dependence on C_i, even with $j \in C_i$, we assume that the problem is with i and not with j. Therefore, j is returned back to F and i is moved to C, solving both dependence problems at once while keeping C as small as possible.

   In practice, there are just a few cases where points will be moved to C in the second step. This means that the first step is not only a fast algorithm, but also performs reasonable well for Criterion 1. The sequential algorithm is illustrated in Figure 9.1 for the 5-point laplacian stencil, where C-points are blue, F-points are red and the numbers indicate the cardinalities defined in the First Step.

   The question that arises when looking for the point with the largest influence set S^T is that many points may have the same amount of influences. The answer is that for the sequential algorithm, one can pick any of them. It is important to say that the coarsening strategy does not need to reach to an unique splitting C and F and all posible splittings are fair enough for AMG. We only need a splitting which guarantees an accurate representation of errors and accurate interpolation, with a relatively small set C so we can

[1] I should say that [5] and [3] have a contradictory assertion in the second step of the coarse-grid selection algorithm. I followed [5] on this issue, since it maked more sense to me.

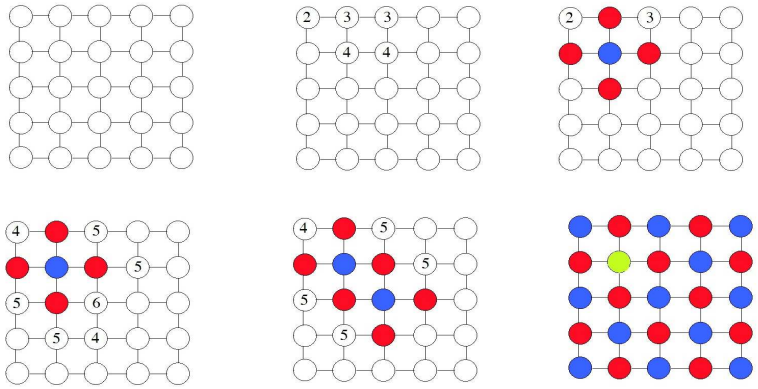

FIG. 9.1. **Sequential Coarsening applied to 5-point laplacian 5x5 grid**

save memory and speed up the solution phase.

10. Parallel coarsening strategies

Note that the described Ruge-Stueben algorithm is sequential in nature: after selecting every coarse point, one needs to update the set of influences or cardinalities for the next step.

There are different strategies to do this in parallel. A naive approach consists in decomposing the abstract domain in various processors and use the sequential algorithm in each of them. Once we have one coarse grid for each processor, the problem arises when the nodes at the sub-domain boundaries does not satisfy Criterion 1, i.e. we have lots of red $F-F$ strong couplings without a common blue C point. This can be seen in Figure 10.1

10.1. Subdomain blocking

One of the primitive parallel strategies to deal with problems at the boundary is the *Subdomain blocking* method. This method proposes in each processor to first coarsen the boundary and then move towards the interior of each subdomain as shown in Figure 10.2.

There are two types: *Full subdomain blocking* and *Minimum subdomain blocking*. The first one simply consists in assigning all boundary nodes to be coarse points. This satisfies Criterion 1, but increases the number of coarse grid points thus increasing the complexity and computing time of the solution phase of AMG.

The *minimum* version is less agressive and proposes to coarsen the boundary of each processor with the same sequential algorithm. This guarantees at least that each F point at the boundary depends strongly on at least one C-point at the same

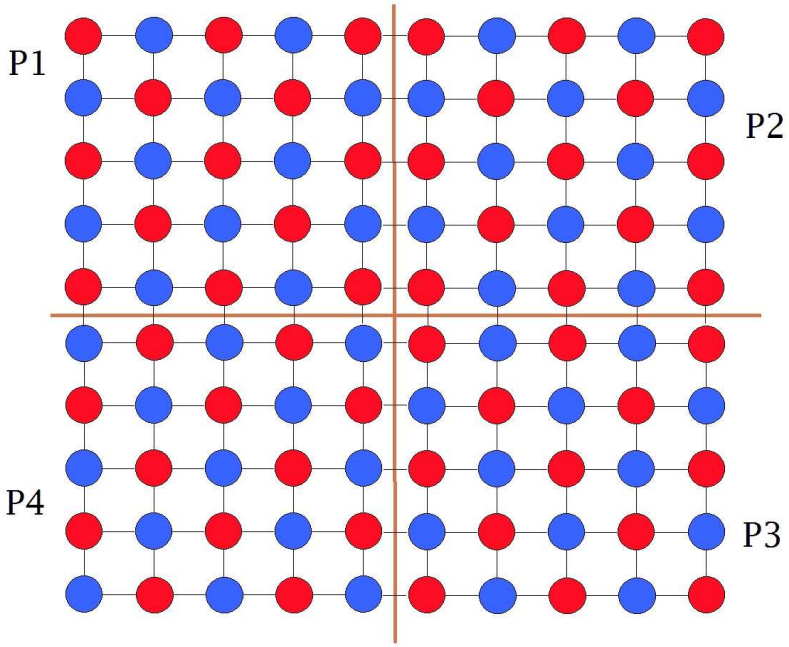

FIG. 10.1. **Naive decomposition in 4 processors without any boundary treatment**

boundary. There is no communication between processors but on the contrary it does not provide a solution for $F-F$ couplings across processors shown in Figure 10.1.

10.2. Other approaches

An iterative approach was developed by Luby, Jones and Plassman called the CLJP coarsening. At each step, an independent set of nodes appropiately chosen are assigned as C-points and all incident edges of the directed graph are removed. If there is a point whose incident edges have all been removed and it is not in C, it will be an F point. The iterations keep going until all nodes have been assigned. This coarsening requires communication between processors at each step and may also lead to C-point clusters, thus requiring more memory storage and computing time during the solution phase.

There is a way to accelerate the CLJP by reducing the number of iterations and thus of communication between processors. This method is called Falgout coarsening and consists in choosing the first independent set for it as the coarse grid points C of the interior of each processor domain after a simple sequential coarsening. During the CLJP however, more C-points are added to the coarse grid increasing complexity.

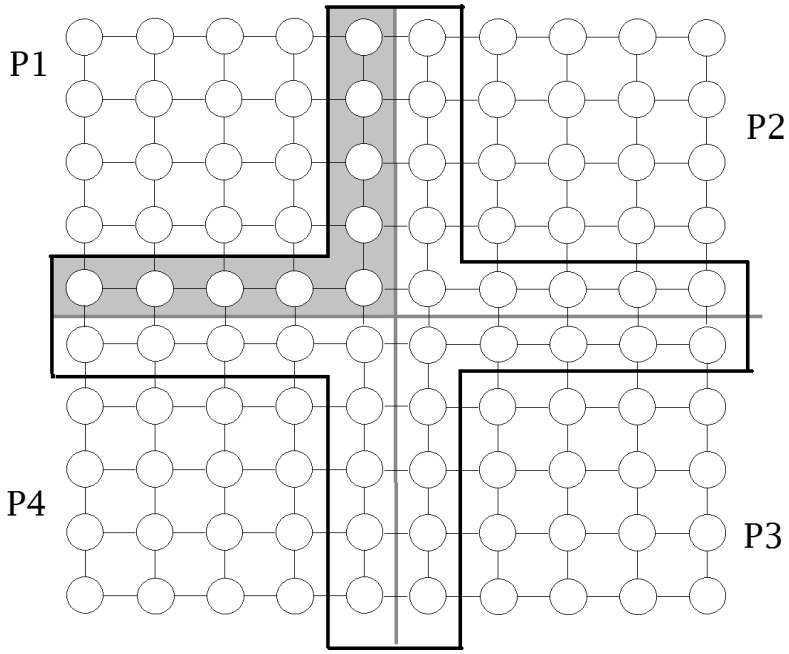

FIG. 10.2. **Subdomain Blocking: first coarsen gray area**

11. Coarse Grid Classification

We shall describe a new algorithm called Coarse Grid Classification [8], which tries to avoid some disadvantages observed in the previous approaches. The key idea will still rely upon a sequential coarsening in each processor, however the boundary problem will be addresed in a different way.

Recall that due to potential symmetry in the sequential coarsening algorithm there is usually more than one choice for the largest λ (see First Step). Thus, depending on our choice for the first coarse grid point we may end up with a different coarse grid as illustrated in Figure 11.1. This is an advantage for our domain decomposition approach since we will have a degree of freedom to choose a specific coarse grid in every processor that better matches our needs across processors.

The Coarse Grid Classification may be implemented in two steps following **??**: First and independently on each processor P, we run the sequential algorithm multiple times with a different starting coarse grid point to generate a set of n_P possible coarse grids, say $V_p = \cup_{i=1}^{n_P} C_{P,i}$. In the second step we need to select one coarse grid from every processor to build an acceptable coarse grid for the whole domain.

The second step is done by transferring all $P \times n_P$ generated coarse grids to a single processor. Since we have a relatively small number of possible coarse grid (it is bounded by the maximal stencil size), the communication should not be an issue.

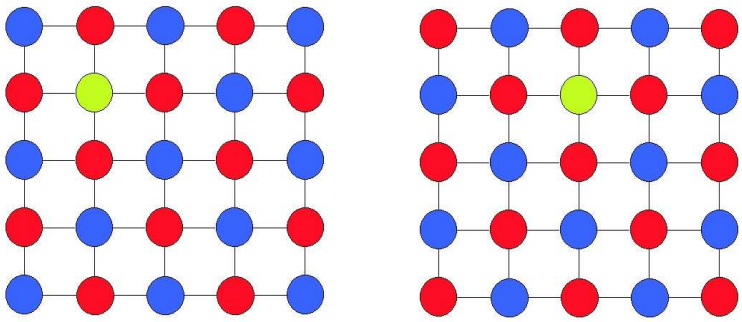

FIG. 11.1. **Ambiguity: starting coarse grid point in green**

Now we may define a weighted graph $G=(V,E)$ in which the vertices represent the possible coarse grids, namely $V=\cup_{p=1}^{P} V_p$ and the edges consists of all pairs (u,v), $u\in V_p$, $v\in V_q$ where p,q are neighboring processors. By neighboring processors we mean that they there is at least one node in one processor which strongly influences a node on the other processor.

There are three types of possible couplings between two nodes at the boundary of two processors: $C-C$, $F-F$ and $C-F$. Since we prefer $C-F$ couplings, we penalize both $C-C$ and $F-F$ with factors -1 and -8 respectively. The problem with strong F-F couplings is that they might not share a common C-point which strongly influences both as required by Criterion 1. Additionally, we slightly penalize C-C couplings since they are acceptable, but increase the grid complexity. Now we may assign a weight for each edge $e=(u,v)\in E$ by writing:

$$\eta(e)=-n_{CC}-8n_{FF}$$

where n_{CC}, and n_{FF} denote the number of strong C-C and F-F couplings between the neighboring coarse grids $u\in V_p$ and $v\in V_q$.

The graph G is transferred to a single processor, which do not require large communication costs since the number of vertices and edges are small compared to the number of unknowns N. For each vertex $v\in V_p$ for some p, we define a set of best matches vertices $B_{v,q}$ in the neighboring processor q as:

$$B_{v,q}=\{w : w=arg\max_{u\in V_q}\eta(v,u)\}$$

We also define the transpose $B^T_{v,q}=\{w : v\in B_{w,p}\}$ and a weight for each node v as $\mu_v=\sum_q |B_{v,q}|+|B^T_{v,q}|$. The value μ_v gives an idea of how many coarse grids on other processors match the coarse grid given by v reasonably well.

We want to pick on each processor the coarse grid with the highest weight μ. If there are processors with no strong couplings at boundaries, they weights μ will be zero and we may delete these processors by picking any coarse grid from their set. Now, we start by choosing the node v with the maximal weight

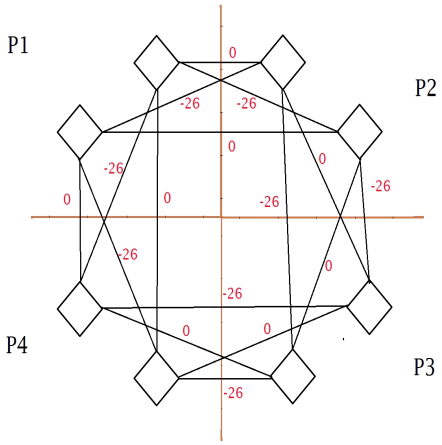

FIG. 11.2. **Construction of directed graph in a single processor.**

μ_v and separate its processor from the graph G as well. We now would prefer to choose a node in $B_{v,q} \cup B_{v,q}^T$ for some q. This is enforced by increasing the weight of each node in $B_{v,q} \cup B_{v,q}^T$ by $\mu_v + 1$ and choosing the node with the maximal weight again. This is repeated until we have chosen one node for every processor.

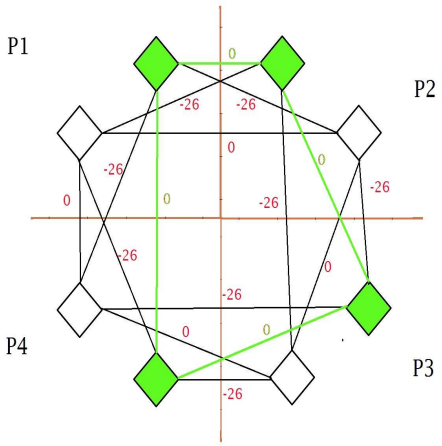

FIG. 11.3. **Decision about which coarse grid to choose at each processor**

Note that the union of the individual coarse grids now represents a consistent coarse grid for the whole domain and in general we do not need to apply further corrections.

12. Numerical Examples

AMG itself has been tested on a broad range of problems as can be seen in [6],

[3],[7]. The AMG method appears to be robust, even for problems that are not related to PDEs, for example, applied to Markov Chains [10]. A optimized implementation, and scalable if possible, is needed in order to perform reliable comparison against other methods.

I have an implemention for the AMG algorithm, the setup phase (coarsening and operators) and the two-cycle algorithm, using the Gauss-Seidel method as relaxation for each step. The implementation is sequential and not optimized, so for now it has no meaning to compare time against other methods.

I decided then to test the implementation in two cases: one corresponding to a 2D-Laplace equation discretized on an uniform grid; the second one, corresponding to a matrix obtained from MatrixMarket: PLATZ1919, from Platzman's oceanographic models full three ocean model. In both cases, I will compare the performance of the AMG algorithm against the Gauss-Seidel method.

12.1. 2D Laplace Problem

Consider a rectangular domain and an $M \times N$ uniform discretizacion. For simplicity, assume Dirichlet conditions are given on the boundary points so that we only care about interior points. Using a 5-stencil discretization, the resulting system of equations has a matrix which satisfies all properties of an M-matrix. Indeed, AMG was originally designed to deal this kind or problems. The structure of the matrix is shown in Figure 12.1.

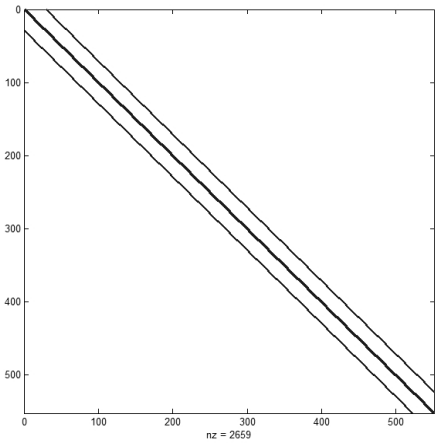

FIG. 12.1. **Typical 5-band structure of 5-point laplacian discretization**

The sequential coarsening algorithm generates a coarser version of this matrix with half of the original size. A closer look into the first 5 components of the fine and coarse version of the matrix reveals that the overall structures is the same. However, the entries are quite different as can be seen in Figure 12.2.

4	-1	0	0	0
-1	4	-1	0	0
0	-1	4	-1	0
0	0	-1	4	-1
0	0	0	-1	4

3.5000	-0.2500	0	0	0
-0.2500	3.2500	-0.2500	0	0
0	-0.2500	3.2500	-0.2500	0
0	0	-0.2500	3.2500	-0.2500
0	0	0	-0.2500	3.2500

FIG. 12.2. **Fine version on the left; Coarse version on the right.**

The Figure (12.3) shows the comparison of number of iterations needed for the AMG to solve the laplacian problem in 5 different cases (increasing matrix sizes).

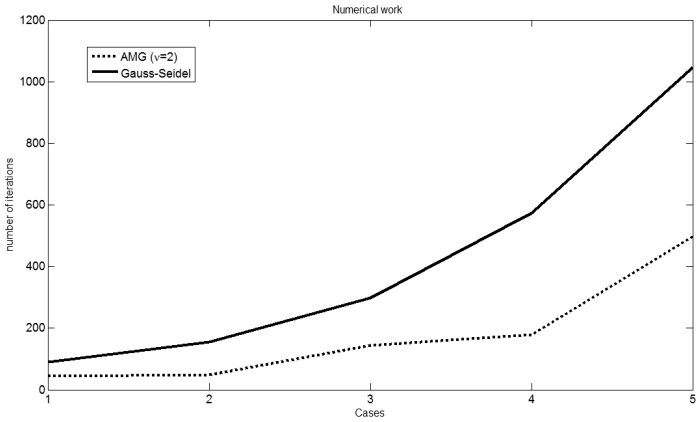

FIG. 12.3. **Performance of AMG cycles agains plain Gauss-Seidel method.**

As can be seen, the AMG cycles are half of the Gauss-Seidel iterations needed to reach the same tolerance error. Recall that one AMG cycle, corresponds to $\nu_1 + \nu_2$ relaxation steps and additional work such as to compute residual transfers and interpolation. In this case, $\nu_1 = \nu_2 = 1$, i.e, one AMG cycle, uses at least two relaxation steps.

12.2. Platzman's Oceanographic Model

The matrix was originally of size 1919×1919, and for faster work I decided to take a block of 800×800 as the input matrix. This matrix has many properties of an M-matrix: it is symmetric, positive-definite, sparse and the diagonal entries are all positive. But there are positive off-diagonal elements and the row sums are not equal to zero. This makes this matrix interesting to test the coarsening step of the AMG method, to see how it deals with a non-M-matrix. In particular, this step had no problem with this matrix, but it had some problems with matrices which had no M-matrices properties. In Figure (12.4) one can see the spartity and structure of the matrix.

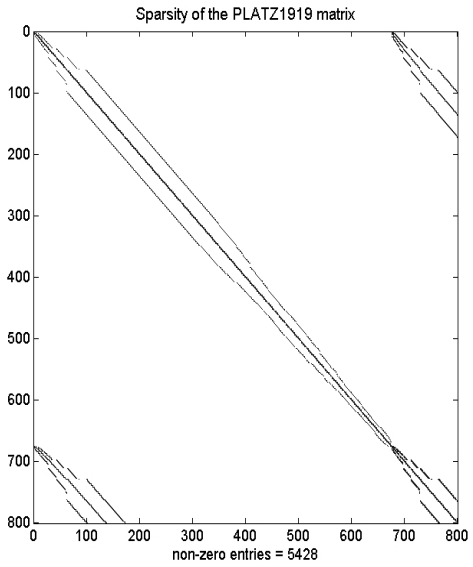

FIG. 12.4. **Properties of platz: symmetric, sparse, almost M-matrix**

In order to make the problem more challenging, a random perturbation matrix was added to the almost M-matrix: $C^1 = A^1 + 0.1 * rand(400)$. This seems to be just a little perturbation, but the new matrix is no longer symmetric and is a dense matrix. Moreover, many off-diagonal entries have now positive or small negative values, which will affect our sets of weak connections, and therefore the computation of interpolation weights and operators. The coarsening step still works fine with this perturbed matrix.

The behavior of AMG for these matrices are similar to what we had before. This shows again that the AMG algorithm is somehow robust: It was designed only for M-matrices, but works well for some non-M-matrices.

13. Conclusions and Future Work

Algebraic Multigrid is an active field of research since the mid-nineties, and many different variations and applications are being developed beyond the classical AMG. Although the classical AMG works well for a large number of problems, its derivation rely upon strong assumptions which may restrict its effectiveness. New versions of AMG thus try to solve specific problems which are not adressed by the classical AMG.

In this prject, no implementation was expected; however, since the Coarse Grid Classification relies heavily on the sequential algorithm, the sequential coarsening of Ruge-Stuebn has been implemented with success. This implementation together with the strategy earlier described for parallelization should provide a straightforward

implementation of the parallel coarsening.

Hybrid smoothers can be used in order to parallelize the relaxations processes. It consists in using the Gauss-Seidel method independently on each processor and sharing information across processor boundaries after each iteration. Other approaches like polynomial smoothers are also available [8].

REFERENCES

[1] Stüben, K., A review of Algebraic Multigrid, *J. Comp. Appl. Math.* 128 (1-2) 2001.
[2] Barth, T.,et al.Multiscale and multiresolution methods: theory and applications *Lectures Notes in Comp.Sci.& Eng.* Springer, Berlin 2002.
[3] Clearly, A.J.,et al. Robustness and Scalability of Algebraic Multigrid, *J. Sci. Comp.* SIAM 2000.
[4] Burden, R., Faires, J. D. Numerical Analysis 5th Ed., PWS 1993
[5] Stueben87, An introduction to numerical analysis, Cambridge University Press, 433 pages, 2003.
[6] Briggs, W. L. et al. A Multigrid Tutorial, Second Edition SIAM 2000
[7] Falgout, R.D., An Introduction to Algebraic Multigrid, *Computing in Science and Eng.* Lawrence L. Nat. Lab. Report 2006.
[8] Yang, U., Parallel Algebraic Multigrid Methods -High Performance Preconditioners, *Numerical Solution of Partial Differential Equations on Parallel Computers* 2006.
[9] Griebel, M. et al. Coarse Grid Classification: A Parallel Coarsening Scheme For Algebraic Multigrid Methods, *Numerical Linear Algebra with Applications* 13 (2-3) 2006.
[10] De Sterck, H.,et al. Algebraic Multigrid for Markov Chains, *J. Sci. Comp. 32* pp. 544-562, SIAM 2010

14. Code for general sequential case

```
function [Ah,I,A2h,D,F,C]=Gexample(AA,Nh)
%General example
%Matrix cut
Ah=AA(1:Nh,1:Nh);

%DEPENDENCE MATRIX
D=sparse(Nh,Nh);
theta=0.25;
for n=1:Nh
%amax=max([-Ah(n,n+1:Nh) -Ah(n,1:n-1)]);
amax=max(setdiff(-Ah(n,:),-Ah(n,n)));
    for m=1:Nh
        if -Ah(n,m)>=theta*amax
            D(m,n)=1;
        else if Ah(n,m)==0
                  D(m,n)=0;
            else D(m,n)=-1;
            end
        end
     end
    D(n,n)=0;
end

%Detect how many strong influencing points has i
lambda=zeros(Nh,1);
```

```
weak=zeros(Nh,1);

%COLORING SCHEME
for i=1:Nh
lambda(i)=Nh-size(find(D(i,:)-1),2);
weak(i)=Nh-size(find(D(i,:)+1),2);
end

F=[];
C=[];

[q,w]=max(lambda);
while sum(lambda(:))>=1
C=unique([C w]);
STw=setdiff(1:Nh,find(D(w,:)-1));
STw=setdiff(STw,F);
F=[F STw];

for j=1:size(STw,2)
    Sw=setdiff(1:Nh,find(D(:,(STw(j)))-1));
    lambda(Sw)=lambda(Sw)+1;
end
lambda(union(F,C))=0;
[q,w]=max(lambda);
%if sum(lambda(:))<100
%    F
%    C
%size(F);
%size(C);
%pause
%end
sum(lambda(:))
end

C=setdiff(1:Nh,F);
F=sort(F);
c=size(C,2)
f=size(F,2)
pause

%SECOND STEP+??? VERIFY H1 violations
i=1;
while(i<f+1)
    %Strong Influencing fine-grid points to F(i)
    i
     S=setdiff(1:Nh,find(D(:,F(i))-1))
     FS=intersect(F,S)
     sum(FS)
     CS=intersect(C,S)
```

```
%     pause
     if sum(FS)~=0
             for m=1:size(FS,2)
            ss=sum(Ah(FS(m),CS(:)))
            end
             if ss==0
                    pause
                    i
              end
     end
     %detects the first j in FS that does not depend strongly on CS
     if size(CS,2)==0
      F=setdiff(F,F(i));
      C=[C F(i)];
     else if size(FS,2)~=0
%influencing matrix from CS to FS
           SEC=full(D(CS,FS))
%if all elements in max(SEC+1)=2, all FS depend on some CS, and it's ok
           [xx,m]=min(max(SEC+1))
 %detects the first j in FS that does not depend strongly on CS
           if xx~=2
             FSprov=setdiff(FS,FS(m))
             CSprov=[CS FS(m)]
%influencing matrix from CSprov to FSprov
             SECprov=full(D(CSprov,FSprov)
%if all elements in max(SECprov+1)=2, all FSprov depend on some CSprov, and it's ok)
              xxx=min(max(SECprov+1))
%detects the first j in FS that does not depend strongly on CS
               if (xxx~=2)&(size(FSprov,2)~=0)
                F=setdiff(F,F(i));
                C=[C F(i)]
                else
                F=setdiff(F,F(m));
                C=[C F(m)]
                end
           end
      end
    c=size(C,2)
    f=size(F,2)
    i=i+1;
%    pause
    end
end

%WEIGHTS
Weights=sparse(f,Nh);
for i=1:f
%    i
    %Weak Influencing fine-grid points to F(i)
```

```
     W=setdiff(1:Nh,find(D(:,F(i))+1));
     FW=intersect(F,W);
    %Strong Influencing fine-grid points to F(i)
     S=setdiff(1:Nh,find(D(:,F(i))-1));
     FS=intersect(F,S);
    %Strong Influencing coarse-grid points to F(i)
     CS=intersect(C,S);
     ed=Ah(F(i),F(i))+sum(Ah(F(i),FW(:)));
     en=zeros(1,Nh);
    if sum(FS)~=0
       i
      pause
         SEC=full(D(CS,FS));
        for m=1:size(FS,2)
            en=en+Ah(F(i),FS(m))*Ah(F(i),:)/sum(Ah(FS(m),CS(:)));
            sum(Ah(FS(m),CS(:)))
        m
        pause
        end
    end
    Weights(i,:)=-(Ah(F(i),:)+en)/ed;
    max(Weights(i,:))
    i
end
%Weights
%pause


%INTERPOLATION MATRIX
I=zeros(Nh,c);
for n=1:c
    I(C(n),n)=1;
end
for m=1:f
    for n=1:c
    I(F(m),n)=Weights(m,C(n));
    end
end

A2h=I'*Ah*I;



function [phijh,error,sfh]=twoAMG(w,v1,vc,Ah,I,A2h,xxxh,xxx2h)
% TWOGRIDMETHOD: Implements the Two-Grid method
% Solves Ahx=sfh
% v1, vc, v2: iteration on the corresponding smoothing steps

% %
```

```
% % %PREPROCESSING - RUN SEPARATELY
% % [Af,I,Ac]=example(31,21); RUNS LAPLACIAN

% % RUNS PLATZ Model
% %[A,rows,cols,entries,rep,field,symm] = mmread('plat1919.mtx');
% % [Ah%,I,A2h,Da,Fa,Ca]=Gexample(A,1000)

%fine mesh
Nh=size(Ah,1);
[sfh]=ones(Nh,1); %Activate in the caase of 4 blocks and given sources
%xxxh=Ah\sfh;

%coarse mesh
N2h=size(A2h,1);
[sf2h]=I'*(sfh);
%[A2h]=mountA(N2h);
%Ac is the coarse matrix, see function example() above.
%xxx2h=A2h\sf2h;


phijh=sparse(Nh,1); %Initial guess
phi_o=ones(Nh,1); %Initialize while loop
e=0;


while ((norm(phijh-phi_o,2)/Nh>1e-6))      % Stopping Criteria
    phi_o=phijh;


%Pre smoothing - Relaxation on fine mesh
%w=0.8; %or 0.5
%v1=1; %or 1 : Relax iterations v1
[phijh,n,err]=gauss(Ah,sfh,Nh,w,v1,xxxh,phijh);


%Restriction of Residual
rh=sfh-Ah*phijh;

[r2h]=I'*(rh);

%Coarse Grid Correction
%w=0.8; %or 0.5
%vc=4; %or 2 : Relax iterations v1
phi2h=sparse(N2h,1); %Initial guess
[phij2h,n,err]=gauss(A2h,r2h,N2h,w,vc,xxx2h,phi2h);
%phij2h=A2h\r2h;  %for EXACT


%Prolongation of error
```

```
[eh] = I*(phij2h);
phijh=phijh+eh;



%Post smoothing - Relaxation on fine mesh
%w=0.8; %or 0.5
%v2=1; %or 1 : Relax iterations v1
%Initial guess: phijh
%[phijh,n,err]=gauss(Ah,sfh,Nh,w,v2,xxxh,phijh);

e=e+1
error(e)=norm(xxxh-phijh,2)/Nh;
end
```