Analysis of an Aggregation-based Algebraic Multigrid Method and its Parallelization

Meng-Huo Chen

A dissertation[†]
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2014

Reading Committee:

Anne Greenbaum, Chair

Loyce Adams

Ulrich Hetmaniuk

Program Authorized to Offer Degree:
UW Applied Mathematics

---

†

University of Washington

Abstract

Analysis of an Aggregation-based Algebraic Multigrid Method and its Parallelization

Meng-Huo Chen

Chair of the Supervisory Committee:
Anne Greenbaum
Department of Applied Mathematics

The interests of this thesis are twofold. First, a two-grid convergence analysis based on the paper [*Algebraic analysis of aggregation-based multigrid* by A. Napov and Y. Notay, Numer. Lin. Alg. Appl. 18 (2011), pp. 539-564] is derived for various aggregation schemes applied to a finite element discretization of a rotated anisotropic diffusion equation. As expected, it is shown that the best aggregation scheme is one in which aggregates are aligned with the anisotropy. In practice, however, this is not what automatic aggregation procedures do. We suggest an approach for determining appropriate aggregates based on eigenvectors associated with small eigenvalues of a block splitting matrix.

In the second part of the thesis several issues regarding the parallel implementation of aggregation-based multigrid methods are discussed. The coarsest grid solving stage of multigrid cycles has been a bottleneck for parallel multigrid algorithms to attain a good speedup. A comparison between a parallel linear system direct solver (MUMPS) and a few steps of preconditioned conjugate gradient (PCG) methods for solving the coarsest grid system is carried out and tested on TACC Lonestar multi-processor machine. Regarding the preconditioner of conjugate gradient iterations, a parallel sparse approximate inverse (SAI) algorithm is used to construct an approximate inverse of the original matrix in order to replace the preconditioner solving step, which is inherently sequential, by matrix-vector multiplications. The linear systems tested arise from discretization of 2D or 3D partial differential equations, which are symmetric positive definite. The results exhibit that us-

ing PCG on the coarsest grid attains better speedup and overall better performance than MUMPS when the number of processors is greater than about 100.

The effects of different decompositions of the physical domain (rows/slab versus blocks/pencils) on the scaling and efficiency of aggregation-based algebraic multigrid are also studied and one sees that the blocks/pencils decomposition of the physical domain reduces the amount of communication and hence has better performance.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude to my advisor Professor Anne Greenbaum for the continuous support of my Ph.D study and research, for her patience, motivation, enthusiasm, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D study.

I would like to thank Professor Loyce Adams, who let me experience the research of multigrd methods in the field and practical issues beyond the textbooks, provides practical suggestion and patiently corrected my writing. To Professor Ulrich Hetmaniuk I was grateful for his help and suggestion on finite element methods.

I would also like to thank Professor Lawrence Snyder, who was willing to be the GSR of my Doctoral Supervisory Committee and provided valuable suggestion on parallel processing and computation.

DEDICATION

This thesis work is dedicated to my wife, Chiahui, who has been a constant source of support and encouragement during the challenges of graduate school and life. I am truly thankful for having you in my life.

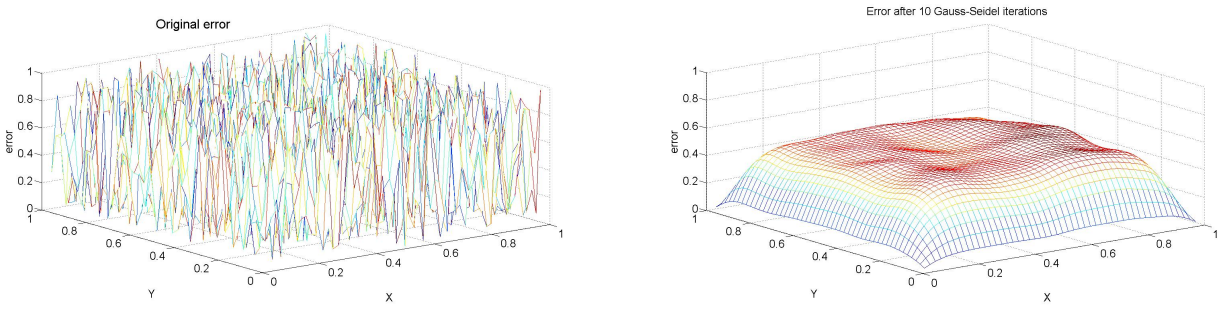
Chapter 1

INTRODUCTION

Multigrid methods have been widely used for solving linear systems arising from discretization of partial differential equations. Under certain circumstances the convergence rate of the method can be proved to be independent of the mesh size, and the total work to solve an N by N linear system is $O(N)$ if a fixed level of accuracy is needed, or $O(N \log N)$ to solve to within truncation error. This optimal convergence behavior makes multigrid methods superior to most other iterative methods. In this thesis, we study the behavior of a certain class of multigrid methods, called aggregation methods, when applied to a variety of second order self-adjoint elliptic partial differential equations. The discussion in this introductory chapter can be found in many books such as [6, 30, 34]. New material will be presented in Chapters 2 and 3.

Let the linear system be denoted as $Au = f$, where A denotes the matrix and f the right-hand side vector resulting from discretization of the PDE, and u is the solution vector that we seek. If v is an approximate solution, then it is convenient to view the error $e = u - v$ as being composed of components (waves) with different frequencies. Standard iterative methods, such as the weighted Jacobi and Gauss-Seidel schemes, reduce the high-frequency components and leave behind smooth, longer wave-length errors. Figure 1.1 shows the effect of 10 Gauss-Seidel iterations on a 2D error vector, and it can be seen that the high frequency error components are reduced significantly.

In order to reduce the low frequency components in the error, the main idea of multigrid is to move these low frequency components to a coarser grid. The cost for solving a linear system on a coarser grid is much less and, more importantly, the smoother components of the error on the finer grids will be composed of more oscillatory modes on the coarser grid. Figure 1.2 shows a 4-mode wave, composed of the lowest 4 of 16 wave numbers on the fine grid, but composed of half of the 8 wave numbers on the coarser grid.



(a) Before Iteration (b) After Iteration

Figure 1.1: Effect of 10 Gauss-Seidel smoothing steps

These low frequency components restricted from the finer level can be smoothed effectively now by performing a few (one or two) steps of a classical iterative method (damped Jacobi or Gauss-Seidel) on the coarser grid. For this reason these methods are called smoothers. This procedure can be carried out recursively until we reach the coarsest grid where one can solve the problem directly. The solution is then interpolated back to the finer levels where additional smoothing steps may be performed. The cycle will be repeated until we get convergence. In summary, the multigrid algorithm (known as a *V-cycle*) can be described as follows:

Algorithm 1.1

Given an initial guess u_0, compute the residual $r_0^{(0)} \equiv f - Au_0$.

For $k = 1, 2...$ (multigrid cycle)

For $j = 1, 2...J-1$ (grid level)

(1) Restrict $r_{k-1}^{(j-1)}$ onto grid level j:

$$f^{(j)} = R_j^{j-1} r_{k-1}^{(j-1)}$$

where R_j^{j-1} is the restriction matrix from grid level $j-1$ to grid level j.

(2) Perform a relaxation sweep (with zero initial guess) on grid level j:

$$G^{(j)} \delta_{k-1}^{(j)} = f^{(j)}$$

where $G^{(j)}$ represents an approximation to $A^{(j)}$.

(3) Then compute the residual:

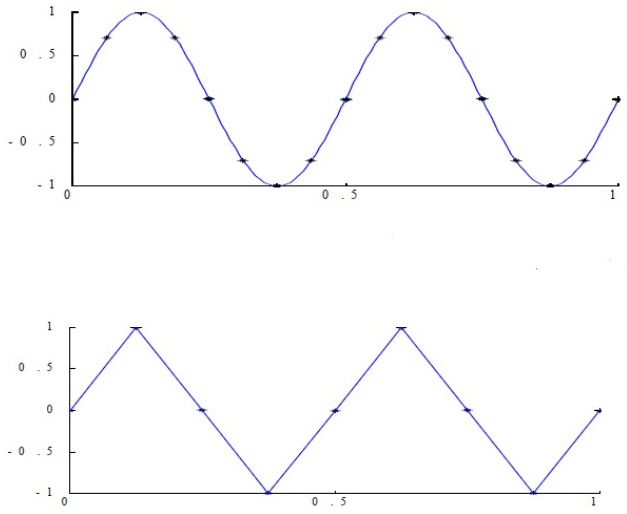


Figure 1.2: 4-mode wave on fine and coarse grids.

$r_{k-1}^{(j)} = f^{(j)} - A^{(j)}\delta_{k-1}^{(j)}$

endfor (grid level)

Restrict $r_{k-1}^{(J-1)}$ onto grid level J by setting $f^{(J)} = R_J^{J-1} r_{k-1}^{(J-1)}$ and solve the coarsest grid problem $A^{(J)} d_{k-1}^{(J)} = f^{(J)}$ directly.

For $j = J-1, ..., 1$ (grid level)

(1) Prolongate $d_{k-1}^{(j+1)}$ to grid level j and add to $\delta_{k-1}^{(j)}$:

$\delta_{k-1}^{(j)} = \delta_{k-1}^{(j)} + P_j^{j+1} d_{k-1}^{(j+1)}$

where P_j^{j+1} is the prolongation matrix from grid level $j+1$ to grid level j.

(2) Perform a relaxation sweep with the initial guess $\delta_{k-1}^{(j)}$ on grid level j:

$d_{k-1}^{(j)} = \delta_{k-1}^{(j)} + (G^{(j)})^{-1}(f^{(j)} - A^{(j)}\delta_{k-1}^{(j)})$.

endfor (grid level)

Prolongate $d_{k-1}^{(1)}$ to grid level 0 and replace $u_{k-1} \leftarrow u_{k-1} + P_0^1 d_{k-1}^{(1)}$.

Perform a relaxation sweep with initial guess u_{k-1} on grid level 0:

$u_k = u_{k-1} + (G^{(0)})^{-1}(f - Au_{k-1})$.

Compute the new residual

$r_k \equiv r_k^{(0)} = f - Au_k$.

endfor (multigrid cycle)

Here the super-index in parentheses of f, r, d, δ, G, and A represents the grid level and the sub-index represents the cycle number. $A^{(j)}$ is the difference operator on grid level j, and it can be defined in a number of different ways, some of which we will describe shortly. The total number of grid levels is $J + 1$. **The quality of a multigrid method depends on at least three ingredients: difference operators on coarser grids, prolongation/restriction operators, and relaxation schemes.** Additionally, there is the pattern of visiting the grids; here we have described a V-cycle but we will later discuss some alternatives. Also, we will describe how multigrid can be used as a stand-alone solver or as a preconditioner for a more sophisticated iterative method such as the conjugate gradient method for symmetric positive definite problems.

Based on the information used to construct the prolongation/restriction operators and coarse grid operators, multigrid methods can be further classified into two categories: geometric multigrid and algebraic multigrid. Geometric multigrid methods use the PDE and a physical grid (often logically rectangular) to define these operators. In algebraic multigrid methods the construction of these operators depends solely on the matrix entries; hence these methods can be used to solve PDE's on unstructured grids or, possibly, even linear systems that do not arise from discretization of PDE's. Between these two extremes are methods that use some information about the problem structure (such as how many unknowns there are per node), but do not rely on a purely geometric method of grid coarsening.

1.1 Geometric Multigrid Methods

The construction of prolongation/restriction operators is closely related to how the coarse grids are formed. In order to describe simply how ingredients of geometric multigrid methods are obtained, we consider PDEs on a rectangular grid with uniform spacing. In geometric

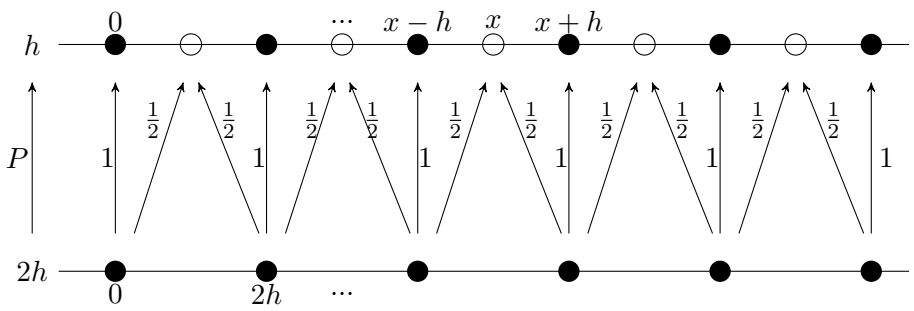


Figure 1.3: Prolongation for a 1D problem.

multigrid methods, the most common coarsening strategy is simply doubling the mesh size along each spatial direction, that is, $h \to 2h$, where h is the spacing on the finest grid. For the first example, we consider a 1D domain as shown in Figure 1.3. The set C of coarse grid points ($\bullet$) are the ones on the grid with mesh size $2h$. Suppose we have a function v_{2h} whose value is defined on coarse grid points. *Linear prolongation* scheme produces a function v_h whose value at a fine grid point x is

$$v_h(x) \equiv P v_{2h}(x) = \begin{cases} v_{2h}(x) & \text{for } \bullet \\ \frac{1}{2} v_{2h}(x-h) + \frac{1}{2} v_{2h}(x+h) & \text{for } \circ \end{cases} \tag{1.1}$$

According to this formula, the matrix form of the prolongation operator P is

$$P = \begin{pmatrix} 1 & 0 & \dots & & & 0 \\ \frac{1}{2} & \frac{1}{2} & & & & \\ & 1 & & & & \\ & \frac{1}{2} & \frac{1}{2} & & & \\ & & & \ddots & & \\ & & & & \frac{1}{2} & \frac{1}{2} \\ & & & & & 1 \end{pmatrix},$$

assuming that the first and last points on the fine grid are coarse grid points as well, as depicted in Fig. 1.3.

Linear prolongation can be combined with various types of restriction. For example, the

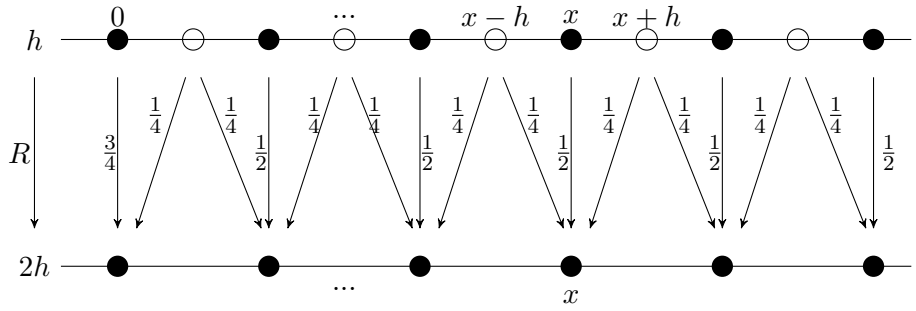


Figure 1.4: Restriction for a 1D problem.

full weighting scheme depicted in Figure 1.4 sets

$$v_{2h}(x) \equiv Rv_h(x) = \frac{1}{4}v_h(x-h) + \frac{1}{2}v_h(x) + \frac{1}{4}v_h(x+h) \tag{1.2}$$

Similar to the 1D case, on a 2D rectangular grid with uniform spacing the coarse grid points are those on a grid with $2h$ spacing in the x and y directions. A very frequently used prolongation method is *bilinear prolongation*, which is given by

$$v_h(x,y) \equiv Pv_{2h}(x,y) = \begin{cases} v_{2h}(x,y) & \text{for } \bullet \\ \frac{1}{2}v_{2h}(x,y+h) + \frac{1}{2}v_{2h}(x,y-h) & \text{for } \diamond \\ \frac{1}{2}v_{2h}(x+h,y) + \frac{1}{2}v_{2h}(x-h,y) & \text{for } \square \\ \frac{1}{4}[v_{2h}(x+h,y+h) + v_{2h}(x+h,y-h) + & \\ v_{2h}(x-h,y+h) + v_{2h}(x-h,y-h)] & \text{for } \circ \end{cases} \tag{1.3}$$

.

and the scheme is depicted in Figure 1.5. Here $\bullet$ represents the coarse grid points and the unfilled markers are the fine grid points that are prolongated from the four coarse points. Obviously, the selection of coarse grid points and prolongation schemes described above relies on the grid formation.

For some classes of PDE's, standard prolongation/restriction operators do not work well. Consider the steady-state diffusion equation

$$-\nabla \cdot (D\nabla U) = F, \tag{1.4}$$

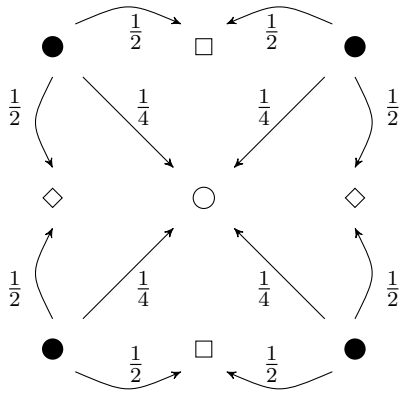


Figure 1.5: Prolongation for a 2D problem.

where ∇U is the gradient of U and D is the diffusion coefficient. When the coefficient D has large jumps across internal grid lines, bilinear prolongation is not appropriate and multigrid methods using bilinear prolongation converge extremely slowly. In the early 1980's, Alcouffe et al [1] proposed a more elaborate way of constructing prolongation operators to overcome this difficulty. Since, for such problems, it is not ∇U but $D\nabla U$ that is continuous, they proposed a prolongation scheme to reflect this. The formula is complicated [1] (p. 444) but is determined by entries in the difference operator. In this way, their geometric multigrid method has some algebraic characteristics. Additionally, the restriction operator is taken to be the transpose of prolongation, which is the most common choice in algebraic multigrid methods. With this modification of the restriction/prolongation operators, they are then able to achieve good performance with standard smoothers such as damped Jacobi and pointwise Gauss-Seidel, using the natural ordering of equations and unknowns.

There are situations in which pointwise Gauss-Seidel smoothing does not work well, with either standard or modified prolongation/restriction [6](p. 131). Consider the 2D anisotropic model problem

$$-\epsilon\frac{\partial^2 U}{\partial x^2} - \frac{\partial^2 U}{\partial y^2} = F$$

where $0 < \epsilon \ll 1$ and the anisotropy is aligned with a grid line. Pointwise Gauss-Seidel with standard coarsening performs badly because the smoothing effect is very poor [1]. A

remedy is to use line smoothers [6]. If we order the variables along lines of y, the direction of strong coupling, then the linear system can be written as

$$A = \begin{pmatrix} D & -cI & & & \\ -cI & D & -cI & & \\ & -cI & D & -cI & \\ & \cdot & \cdot & \cdot & -cI \\ & & & -cI & D \end{pmatrix} \tag{1.5}$$

where $c = \epsilon/h^2$ and I is the identity matrix. The diagonal blocks D are tridiagonal and identical. Each block is associated with an individual vertical line and has stencil $\frac{1}{h^2}(-1 \;\; 2+ 2\epsilon \;\; -1)$. In the block *Jacobi* smoothing for each y-line we solve the system of the form

$$Dv_i = g_i \tag{1.6}$$

where v_i contains the variables in the ith y-line that need to be updated and g_i is the corresponding right-hand side vector and its jth component g_{ij} is

$$g_{ij} = f_{ij} + \frac{\epsilon}{h^2}(v_{i-1,j} + v_{i+1,j}). \tag{1.7}$$

The block *Jacobi* scheme updates all y-lines simultaneously using the values from the previous iteration while the line *Gauss-Seidel* smoothing updates the variables in one y-line and then proceeds to update the second y-line using the updated values in the first y-line and so on, until the last y-line is updated. Similarly, if ϵ is much larger than 1 in some parts of the domain and much smaller than 1 in other parts, then alternating line smoothing is an ideal smoothing scheme [1].

In summary, the approaches for coarsening, constructing prolongation/restriction operators and line smoothing described so far rely on geometric information, such as grid structure, mesh sizes and PDEs (e.g., diffusion coefficients, anisotropy). Although in [1] the difference operators at level $j+1$ are determined algebraically by the Galerkin formula

$$A^{(j+1)} = R_{j+1}^{j} A^{(j)} P_j^{j+1} \tag{1.8}$$

where the restriction operator R_{j+1}^{j} is the transpose of the prolongation operator P_j^{j+1}, this is still considered as a geometric multigrid method because it relies on grid information

to form coarser grids and on the PDE to determine P_j^{j+1}. In some geometric multigrid methods the difference operators at coarser levels are constructed by direct discretization of the PDE on that grid [2, 11, 30]. Due to its algebraic nature, however, the Galerkin formula (1.8) is an important tool for forming "coarse grid" difference operators in most algebraic multigrid methods, where no geometric information is used at all.

1.2 Classical Algebraic Multigrid Methods

Instead of using geometric information (grids, PDEs), algebraic multigrid methods (abbreviated as AMG) use only the entries of the system matrix in the coarsening process. This feature allows algebraic multigrid methods to solve a broader range of linear systems, such as the ones arising from discretization of PDEs on an unstructured grid, or even ones that do not come from PDEs. For general discussions of algebraic vs geometric multigrid methods, see, for example, [6, 30, 13]. AMG methods can be further classified into two categories: classical AMG and aggregation-based AMG. In this section, we discuss the former one.

Throughout the section, we assume that in addition to being SPD, the matrix A is an M-matrix; that is, it has positive diagonal entries and nonpositive off-diagonal entries. This assumption is used in determining which points are strongly coupled and hence which points should be defined as coarse grid points and used in the interpolation formula for a particular fine grid point. In practice, if the original matrix A is not an M-matrix, or if coarse grid matrices generated by this process are not M-matrices, then the positive off-diagonal connections are considered to be weak connections and are ignored [13].

1.2.1 Algebraically Smooth Error, Strong Influence and Dependence

As mentioned in the previous section, in geometric multigrid methods, the slow varying (smooth) components of error left over after smoothing are transfered to a coarser grid and it is hoped that these components are eliminated efficiently there. In algebraic multigrid methods, smooth error is defined as any error components that are *not* reduced efficiently by the fine grid relaxation scheme, whether or not they correspond to physically smooth

error. If the relaxation scheme has the form

$$u_{k+1} = u_k + G^{-1}(f - Au_k),$$

then the error $e_{k+1} = u - u_{k+1}$ satisfies

$$e_{k+1} = (I - G^{-1}A)e_k.$$

If e_k has components only in the direction of eigenvectors corresponding to small eigenvalues of $G^{-1}A$, then $G^{-1}Ae_k \approx 0$ and e_{k+1} will not change much from e_k. In this case, we say that the error is smooth with respect to the splitting matrix G. For example, if a weighted Jacobi relaxation method is used, then $G = \omega^{-1}D$, where D is the diagonal of A, and algebraically smooth error satisfies componentwise

$$e_i \approx -\frac{1}{a_{ii}} \sum_{j \neq i} a_{ij} e_j. \tag{1.9}$$

A similar equation can be derived when $G = D - L$ is the Gauss-Seidel smoothing, and relation (1.9) is the basis for standard AMG prolongation schemes [6, 30].

Put another way, algebraically smooth error is error for which the 2-norm of the preconditioned residual is much less than the A-norm of the error. The A-norm of a vector v is $\|v\|_A \equiv < Av, v >^{1/2} = \|A^{1/2}v\|$, where $\|\cdot\|$ without a subscript denotes the 2-norm. The preconditioned residual is $G^{-1}(f - Au_k)$. If, in addition, the row sums of the matrix are close to 0, then it can be shown with some algebraic manipulation [6] that algebraically smooth error satisfies

$$\sum_i \sum_j (-a_{ij})(e_i - e_j)^2 << \sum_i a_{ii} e_i^2. \tag{1.10}$$

For most values of i, then, if some $-a_{ij}$ is comparable in size to a_{ii}, it must be the case that $(e_i - e_j)^2 << e_i^2$. That is, algebraically smooth error varies slowly in the direction of strong couplings.

Based on this observation, we define the notions of strong influence and dependence, which will be used to determine coarse grid points and the prolongation operator. This discussion follows that in [6].

Definition. Given a threshold value $0 < \beta \leq 1$, the variable u_i *strongly depends* on u_j if

$$-a_{ij} \geq \beta \max_{k \neq i} \{-a_{ik}\}. \tag{1.11}$$

In this case the variable u_j is said to *strongly influence* u_i.

1.2.2 Selecting Coarse Grid Points

Let C denote the set of coarse grid points and F the set of fine grid points that are not coarse grid points; the total set of fine grid points is $C \cup F$. We wish to choose the coarse grid variables so that smooth error can be represented accurately on the coarse grid and so that interpolation or prolongation from the coarse to fine grid results in accurate values of the fine grid variables. If an F-point i strongly depends on a point j, then this suggests that the point j should be in C. The third requirement is that the set C should have substantially fewer points than the fine grid [13].

For each point i, let S_i denote the set of points that strongly influence i, and let C_i, the coarse interpolatory set, denote the set of coarse grid points that strongly influence point i. Another useful set is the set of points that strongly depend on i, and this will be denoted as S_i^T [13]. In order to satisfy the above conditions as much as possible, the coarse grid selection process is guided by the following two heuristic criteria:

1. For each F-point i, every point j in S_i either should be in the coarse interpolatory set C_i or should strongly depend on at least one point in C_i.

2. No C-point should strongly depend on another C-point, and of all sets with this property, C should be a maximal set; that is, if any other point is added to C, then the condition would be violated.

Heuristic (1) is designed to ensure accurate prolongation from the coarse to fine grid. It would be best if C_i contained all points in S_i, but this might result in too many C-points; a moderate level of accuracy should be attainable if C_i contains at least one point in S_i. Condition (2) is designed to strike a balance between the size of the coarse grid (and hence the amount of work required by coarse grid iterations) and the accuracy with which smooth

errors can be represented on the coarse grid (which influences the number of multigrid cycles needed to solve the linear system). It is not always possible to enforce both (1) and (2). Usually, condition (1) is enforced rigorously, while (2) is used as a guide.

Based on these two criteria, coarse grid selection in **classical algebraic multigrid** consists of two phases. In the first phase the algorithm does the initial partition of variables into the C- and F- points according to strong dependencies. In the second phase, some F-points may be changed to C-points to enforce (1).

The first pass begins by assigning to each point i a measure λ_i, which is usually taken to be the number of points that it strongly influences; that is, the cardinality of S_i^T. A point i with maximum λ_i is then selected as the first coarse grid point. The points that it strongly influences (that is, those in S_i^T) are then taken to be F-points, since no C-point should strongly influence another C-point. For each of these new F-points, we should look at other points that strongly influence it as potential candidates for C-points. Therefore, for each new F-point j, we increment by 1 the measure λ_k of each unassigned point k that strongly influences point j; that is, each unassigned point in S_j. This process is then repeated. A new unassigned point i with maximal λ_i is chosen as the next coarse grid point, the unassigned points in S_i^T are taken to be F-points, and the measures of unassigned points that strongly influence the new F-points are incremented by 1. The process is repeated until all points have been assigned to either C or F. It should be noted that there often will be many points with the same maximal value λ_i. Thus the result of the coarsening algorithm depends very much on the order in which points are scanned for maximal λ's. The hope is that the results will come close to satisfying (1) and (2) independent of which maximal value λ_i is chosen at each step.

As an example, consider the 9-point Laplacian, with operator stencil

$$\begin{pmatrix} -1 & -4 & -1 \\ -4 & 20 & -4 \\ -1 & -4 & -1 \end{pmatrix} \tag{1.12}$$

The selection process is depicted in the figures (a) - (k) on the next page. For this example,

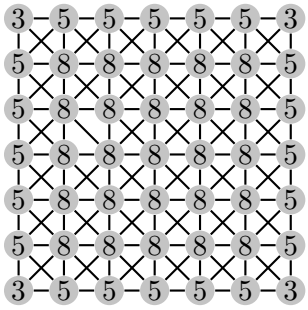
we assume the threshold $\beta = 0.25$. Then every connection is one of strong dependence and initially all interior points have a measure of $\lambda = 8$. The points along the side have $\lambda = 5$ and the four corner points of the grid have a measure of $\lambda = 3$, as shown in Figure (a). In Figure (b), following lexicographical ordering, the first coarse point with the largest λ is chosen at the $(2, 2)$ position. Then the neighbors of the first coarse grid point are assigned as F-variables as seen in Figure (c) and the parameter λ of the remaining variables that strongly influence the new F-variables are updated, adding 1 for each F-variable that the point strongly influences . The rest of the figures show the repeating coarsening process as it evolves. Finally a C-F partition of the grid is obtained in Figure (k). Note that for this simple example all F-points in the resulting partition after the first pass strongly depend on at least one of the C-points and there is no need to apply the second pass of selection.

If the result of the initial pass does not satisfy (1), then a second phase is implemented. In this phase, F-points are examined to see if there are strong F-F dependencies between points *not* depending strongly on a common C-point. If so, then one of the two F-points is tentatively turned into a C-point. As more F-points are examined and further F-F dependencies are found, we first see if (1) can be satisfied using points on the list of tentative C-points. If so, then no additional points are added to C, but if not then one of the F-points must be tentatively added to C. The idea is to satisfy (1) while changing as few F-points to C-points as possible.

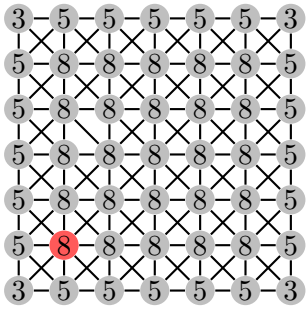
1.2.3 Constructing the prolongation operator

Once the coarse grid set C is determined, one can construct the prolongation operator P. In classical algebraic multigrid, for each fine-grid variable i, the neighborhood N_i consisting of all points that couple to i (strongly or weakly) is divided into three categories [13]:

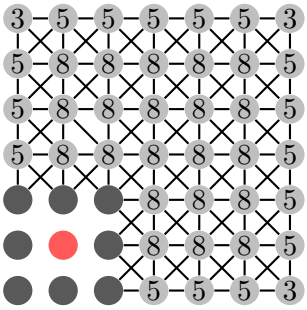
- the neighboring coarse-grid variables (in C) that strongly influence i; this is the coarse interpolatory set for i, denoted by C_i;
- the neighboring fine-grid variables that strongly influence i, denoted by D_i^s ;
- the variables that do not strongly influence i, denoted by D_i^w; it is called the set of



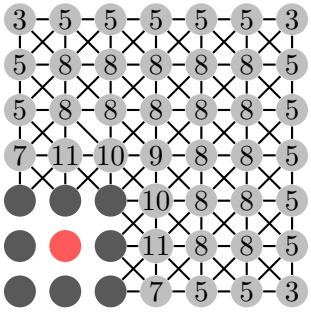
(a) Each nodes is assigned with a measure λ.



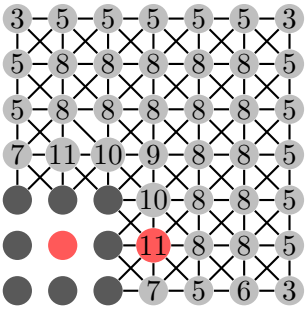
(b) Select C-pt with maximal measure.



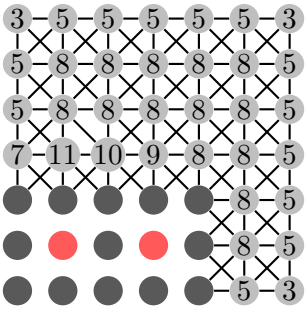
(c) Select neighbors as F-points.



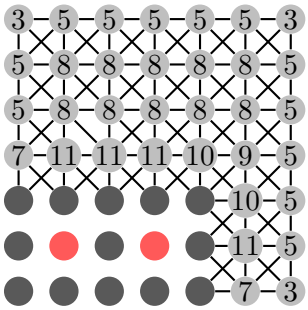
(d) Update measures of F-pt neighbors.



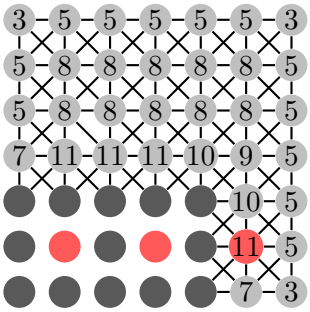
(e) Select C-pt with maximal measure.



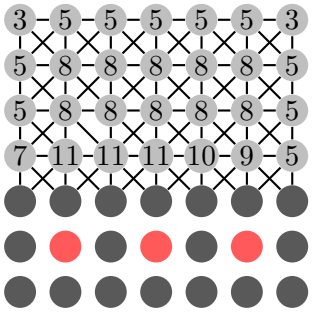
(f) Select neighbors as F-points.



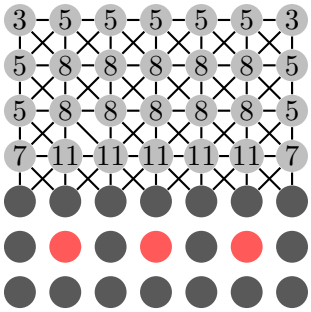
(g) Update measures of F-pt neighbors.



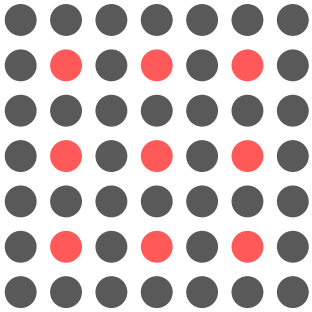
(h) Select C-pt with maximal measure.



(i) Select neighbors as F-points.



(j) Update measures of F-pt neighbors.



(k) Finally the red points are coarse grid points and the black ones are fine grid points that are not in coarse grid

weakly connected neighbors; these variables can be either fine- or coarse-grid variables.

We require that the ith component of Pe be given by

$$(Pe)_i = \begin{cases} e_i & \text{if } i \in C \\ \sum_{j \in C_i} w_{ij} e_j & \text{if } i \in F \end{cases} \tag{1.13}$$

where w_{ij} is the prolongation weight.

Rewrite equation (1.9) using the three categories as

$$a_{ii} e_i \approx - \sum_{j \in C_i} a_{ij} e_j - \sum_{j \in D_i^s} a_{ij} e_j - \sum_{j \in D_i^w} a_{ij} e_j \tag{1.14}$$

For the terms involving weakly connected neighbors D_i^w, we distribute them to the diagonal coefficients by simply replacing e_j in the rightmost sum of (1.14) by e_i,

$$\left(a_{ii} + \sum_{j \in D_i^w} a_{ij} \right) e_i \approx - \sum_{j \in C_i} a_{ij} e_j - \sum_{j \in D_i^s} a_{ij} e_j \tag{1.15}$$

Such substitution is sensible since a_{ij} are small for $j \in D_i^w$ and any error committed in making this assignment will be relatively insignificant. Secondly, approximate the e_j in the terms involving the neighborhood of second category by weighted sums of the e_k for $k \in C_i$, that is,

$$e_j \approx \frac{\sum_{k \in C_i} a_{jk} e_k}{\sum_{k \in C_i} a_{jk}} \tag{1.16}$$

Substituting (1.16) into (1.15), the prolongation weight w_{ij} is given by

$$w_{ij} = - \frac{a_{ij} + \sum_{m \in D_i^s} \left(\frac{a_{im} a_{mj}}{\sum_{k \in C_i} a_{mk}} \right)}{a_{ii} + \sum_{l \in D_i^w} a_{il}} \tag{1.17}$$

1.2.4 Forming the Coarse Grid Matrix and Continuing

As mentioned at the end of the Section 1.1 (formula 1.8), the coarse grid matrix A_C is determined via the Galerkin formula,

$$A_C = P^T A P, \tag{1.18}$$

where the restirction operator is $R = P^T$ when A is symmetric, as we are assuming throughout this thesis. The coarse grid selection and coarse grid matrix formation are repeatedly applied until the coarsest level is reached and the grid hierarchy $A^{(j)}$, $j = 0, 1, \ldots, J$ is constructed.

The smoothers in classical algebraic multigrid are simple ones that are often used in geometric multigrid, for example, damped Jacobi and (pointwise) Gauss-Seidel (which is usually preferred). As the discussion of this section indicates, all constructions are based on the matrix entries, with no geometric information involved.

1.3 Aggregation-Based Algebraic Multigrid Methods

The weighted prolongation formulas (1.13) and (1.17) in **classical algebraic multigrid** show that values at F-points are linear combinations of (usually) more than one C-variable. In this section we discuss **aggregation-based algebraic multigrid**, which uses the simplest possible approach, piecewise constant prolongation/restriction, on transferring vectors between grid levels. This discussion is based on the work of Notay, et al [24, 25, 22]. Earlier work on aggregation methods can be found, for example, in [32, 33]

1.3.1 Prolongation and forming of coarse grid

In aggregation-based algebraic multigrid the variables of the linear system are subdivided into disjoint small groups I_k, $k = 1, ..., N_C$ called "aggregates", where each I_k represents a variable on the coarse level and the total number of aggregates is N_C. See Figure 1.6.

In each aggregate, the function's values are assumed to be uniform, that is, functions are piecewise constant. Thus the prolongation operator is of the form

$$P_{ik} = \begin{cases} 1 & \text{if } i \in I_k, \\ 0 & \text{otherwise} \end{cases} \qquad (1 \le i \le N, 1 \le k \le N_C). \tag{1.19}$$

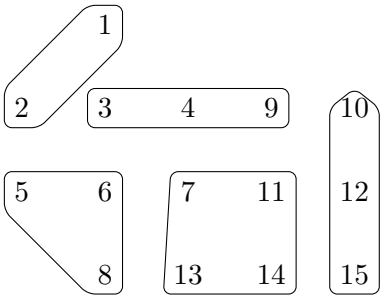


Figure 1.6: Subdivision of fine level variables into aggregates

P is an $N \times N_C$ matrix with exactly one nonzero entry per row [24]. With proper ordering of the variables, P can be written in the form

$$P = \begin{pmatrix} \mathbf{1}_{m_1} & & & \\ & \mathbf{1}_{m_2} & & \\ & & \ddots & \\ & & & \mathbf{1}_{m_{N_C}} \end{pmatrix}$$

where each vector $\mathbf{1}_{m_i}$ corresponds to an aggregate with m_i elements. The restriction operator, like in classical algebraic multigrid, is $R = P^T$

The construction of the coarse grid matrix uses the Galerkin formula (1.18). In the practical implementation, there is no need to explicitly form P and the entries of coarse grid matrix A_C are calculated by

$$(A_C)_{ij} = \sum_{k \in I_i} \sum_{l \in I_j} a_{kl}. \tag{1.20}$$

1.3.2 Pairwise aggregation

The process of aggregation starts by forming pairs of variables. Here we describe the automatic aggregation procedure in [20]. Using a similar principle to that in classical algebraic multigrid, the first step of the process is to define the set of variables S_i that are strongly negatively connected to i, using threshold β:

$$S_i = \{j \neq i : a_{ij} < -\beta \max_{\substack{k \neq i \\ a_{ik} < 0}} |a_{ik}|\}, \tag{1.21}$$

and the number m_i of points that i strongly influences:

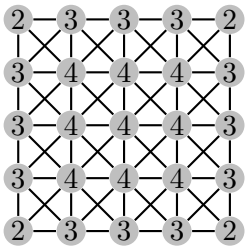
$$m_i = |\{j : i \in S_j\}|. \tag{1.22}$$

At the beginning an unmarked node $\tilde{i}$ with minimal measure $m_{\tilde{i}}$ is chosen. Then an aggregate is formed by grouping the chosen node $\tilde{i}$ with the unmarked node $\tilde{j} \in S_{\tilde{i}}$ to which it is most strongly negatively coupled. If $S_{\tilde{i}}$ contains no unmarked nodes then the coarsening algorithm leaves $\tilde{i}$ as a singleton. The indices in the pair are then marked "used" and m_i is decremented by 1 for each unmarked node i in $S_{\tilde{i}}$ or $S_{\tilde{j}}$. This process is then repeated until all nodes are marked. Then one forms the initial prolongation operator P_1 and a tentative coarse grid matrix $A_1 = P_1^T A P_1$. For a better coarsening ratio, the second pass of the same pairing process is applied to the newly formed aggregates, using matrix A_1 in place of A. In this way, one obtains pairs of pairs and a new prolongation matrix P_2. The final coarse grid matrix is $A_C = P_2^T A_1 P_2 = P_2^T P_1^T A P_1 P_2$, and the prolongation matrix is $P = P_1 P_2$.

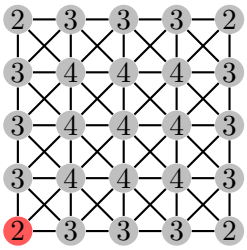
For demonstration we consider again the 9-point stencil in (1.12) and the whole process is shown in Figures (a)-(g) on the next page. The first coarse variable is taken to be one with minimum m_i (lower left corner node). Once an aggregate is formed, 1 is subtracted from m_i of unmarked nodes that are strongly negatively connected to the nodes in the formed aggregate (comparing with the addition in classical algebraic multigrid). An initial coarse grid matrix A_1 is formed based on the aggregates in figure (i). Then the second pass of pairing process is applied on A_1 to form the final aggregates in figure (j) and coarse grid matrix A_C.

1.3.3 Improvement

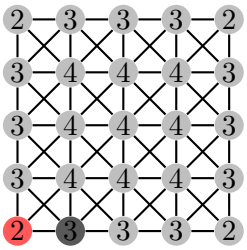
Piecewise constant prolongation has some serious disadvantages: Information from a piecewise linear or higher order approximation is lost. High frequency error is introduced through prolongation. In general the grid size or level independent convergence is not easy to obtain using V-cycle or W-cycle on model problems [28]. Thus aggregation-based AMG needs certain improvements in order to become practical. However, aggregation-based AMG methods have their advantages. The setup phase is simpler and hence setup time is always shorter than that in classical algebraic multigrid. Moreover, in general, the average number of



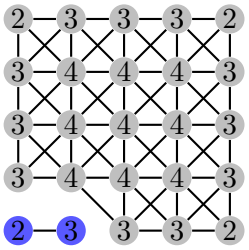
(a) Each nodes is assigned with a measure m_i.



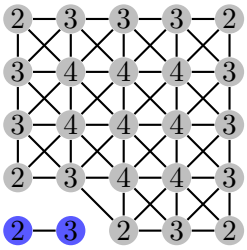
(b) Select a point with minimal m_i



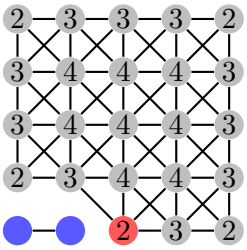
(c) Select neighbor j such that $a_{ij} = \min_{k \in \bar{U}} a_{ik}$, where $\bar{U}$ is the list of all unmarked unknowns



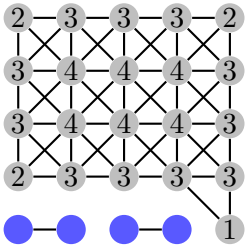
(d) Pair up i and j and form an aggregate



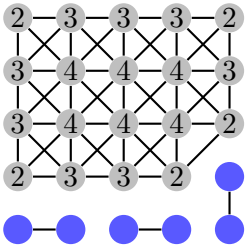
(e) For all $l \in S_i \cup S_j$, update the measure $m_l = m_l - 1$.



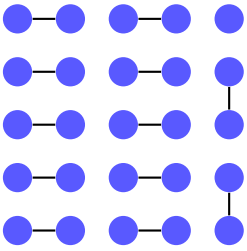
(f) Repeat (b): Select a point with minimal m_i



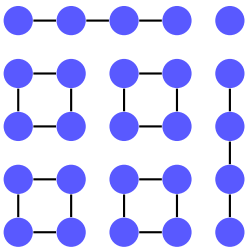
(g) Repeat (d) - (e), then select neighbor j such that $a_{ij} = \min_{k \in \bar{U}} a_{ik}$



(h) Repeat (b) - (e) to form more aggregates



(i) Final structure after first pass



(j) 2nd pass to obtain larger aggregates

nonzero entries per row in coarse grid matrices does not increase as rapidly as it does in classical algebraic multigrid..

Smoothed aggregation, as introduced in [31, 33], is proposed to overcome intrinsic difficulties arising from aggregation by smoothing the prolongation operator. Here the initial piecewise constant prolongation, say P_0, is further improved by some "smoothing process", multiplying by $\hat{G}$ to form the final prolongation P:

$$P = \hat{G}P_0 \tag{1.23}$$

In [31, 33] this smoothing is done by applying one step of weighted Jacobi iteration and $\hat{G} = (I - \omega D^{-1}A)$ where ω is the relaxation parameter.

Another approach to improving the performance of aggregation-based AMG is to use a more elaborate cycle strategy. In [24] a K-cycle (which will be described in the next section) is used, and the algorithm appears to converge at a rate that is independent or nearly independent of the grid size for the problems tested therein.

1.4 Cycle Strategy

1.4.1 V-cycle, W-cycle and K-cycle

In the discussion of previous sections, we have seen different approaches and procedures to construct ingredients, such as coarse grid matrix, prolongation/restriction operators and smoothers, of multigrid methods. These steps constitute the setup phase of multigrid methods. Once the grid hierarchy is built up in setup phase, multigrid methods iterate using these ingredients to compute the solution. When multigrid methods transfer between grids, there are different patterns of visiting coarse levels. The patterns are also called cycle strategies. Before describing the different cycle strategies, we first describe one cycle of a two-grid method with one pre- and post-smoothing (smoother G), and with prolongation and restriction operators P and R, respectively.

Two-grid method

Given an approximate solution $u^{(0)}$ to $A^{-1}f$, and $\mathbf{r}^{(0)} = f - Au^{(0)}$,

1. Presmooth one time using G : $u^{(0)} = u^{(0)} + G^{-1}\mathbf{r}^{(0)}$.

2. Compute new residual : $\tilde{\mathbf{r}}^{(0)} = f - Au^{(0)}$.

3. Restrict residual : $\mathbf{r}^{(1)} = R\tilde{\mathbf{r}}^{(0)}$.

4. Compute solution $\mathbf{d}^{(1)}$ to $A^{(1)}\mathbf{d}^{(1)} = \mathbf{r}^{(1)}$.

5. Prolongate and add correction to $u^{(0)}$: $u^{(0)} = u^{(0)} + P\mathbf{d}^{(1)}$.

6. Postsmooth one time using G : $u^{(0)} = u^{(0)} + G^{-1}(f - Au^{(0)})$.

Here the superscripts represent the grid level and the finest level is 0. The linear system in the coarse grid correction (step 4) is solved exactly. If instead one solves the linear system $A^{(1)}d^{(1)} = r^{(1)}$ approximately by one cycle of the same two-grid method, then one obtains a three-grid method, where the coarsest level (level 2) is solved exactly. Proceeding recursively in this way, we obtain the multigrid V-cycle, which has already been described in Algorithm 1.1. The graphical depiction is shown in Figure 1.7. Blue circles represent a visit to a particular grid level and a smoothing step on that level. A solid line with arrow indicates the direction of grid transfer (restriction or prolongation). This cycle strategy is called a V-cycle because the graph has the shape of the letter V.

While the V-cycle replaces the coarse grid solve (step 4) by one cycle of the two-grid method, the W-cycle replaces each coarse grid solve by two V-cycles, resulting in a pattern depicted in Figure 1.8 with four grid levels. The W-cycle may reduce the number of multigrid iterations to compute the solution, but each single W-cycle is more expensive than a single V-cycle.

Another cycle strategy, the K-cycle, replaces the coarse grid solve (step 4) by one or two steps of a Krylov space method such as the conjugate gradient (CG) method for SPD problems or the GMRES method for more general nonsymmetric problems.

Any of these cycle strategies can be used as preconditioners for the conjugate gradient method for SPD problems. However, while the V-cycle and W-cycle represent fixed preconditioners, the K-cycle uses a *different* preconditioner at each step since the coefficients

(and possibly the number of iterations) of the CG method on coarse grids will be different at each (outer) step. Thus, if the K-cycle is to be used as a preconditioner for CG, then a variant of CG called *flexible CG* must be used. In the next subsections we discuss the CG method and the "flexible CG" variant, before discussing the detailed implementation of the K-cycle.

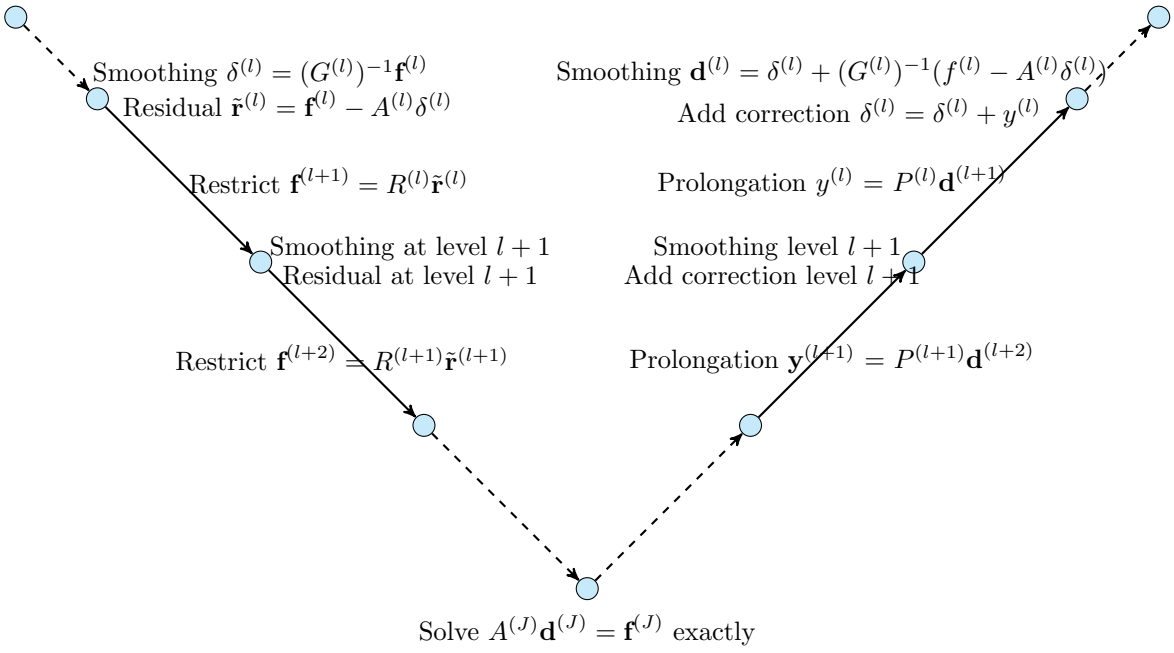


Figure 1.7: V-cycle

1.4.2 Conjugate Gradient Method

The preconditioned CG method (PCG) for solving an SPD linear system $Au = f$ with preconditioner M is as follows:

Algorithm 1.2P. Preconditioned Conjugate Gradient Method (PCG)

(for a Hermitian positive definite linear system with a Hermitian positive definite preconditioner M)

Given an initial guess u_0, compute the residual $r_0 \equiv f - Au_0$ and solve $Mz_0 = r_0$.

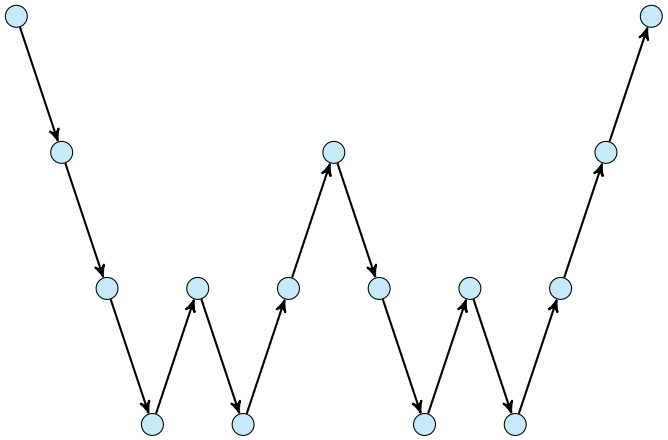


Figure 1.8: W-cycle

Set $p_0 = z_0$.

For $k = 1, 2, ...,$

Compute Ap_{k-1}.

Set $u_k = u_{k-1} + a_{k-1}p_{k-1}$, where $a_{k-1} = \dfrac{< r_{k-1}, z_{k-1} >}{< p_{k-1}, Ap_{k-1} >}$.

Compute $r_k = r_{k-1} - a_{k-1}Ap_{k-1}$.

Solve $Mz_k = r_k$.

Set $p_k = z_k + b_{k-1}p_{k-1}$, where $b_{k-1} = \dfrac{< r_k, z_k >}{< r_{k-1}, z_{k-1} >}$.

end for

The PCG method is *optimal* in the sense that of all vectors $\tilde{u}_k$ such that

$$\tilde{u}_k - u_0 \in \text{span}\{z_0, (M^{-1}A)z_0, (M^{-1}A)^2 z_0, \ldots, (M^{-1}A)^{k-1} z_0\},$$

the PCG approximation u_k is the one for which the A-norm of the error is minimal. This follows from various orthogonality properties that are derived, for example, in [16]. However, it relies on the fact that the preconditioner M does not change.

The preconditioning step, solving $Mz_k = r_k$, can be replaced by a multigrid cycle for the problem $Az = r_k$. That is, starting with initial guess 0 for z, one goes through one cycle of Algorithm 1.1 or some variant that we have discussed and the resulting approximation to z is taken as z_k. For example, if a two-grid method with one pre-smoothing step with

smoother G_1 and one post-smoothing step with smoother G_2 is used as a preconditioner, then the following steps are performed:

1. Presmooth: $\zeta^{(0)} = G_1^{-1} r_k$.

2. Compute new residual: $\tilde{s}^{(0)} = r_k - A\zeta^{(0)} = (I - AG_1^{-1})r_k$.

3. Restrict residual: $s^{(1)} = R\tilde{s}^{(0)} = R(I - AG_1^{-1})r_k$.

4. Solve $A^{(1)}d^{(1)} = s^{(1)}$ to get $d^{(1)} = {A^{(1)}}^{-1} R(I - AG_1^{-1})r_k$.

5. Prolongate and add correction: $\zeta^{(0)} = \zeta^{(0)} + Pd^{(1)} = G_1^{-1} r_k + P{A^{(1)}}^{-1} R(I - AG_1^{-1})r_k$.

6. Postsmooth: $z_k = \zeta^{(0)} + G_2^{-1}(r_k - A\zeta^{(0)}) = G_2^{-1} r_k + (I - G_2^{-1}A)\zeta^{(0)}$; i.e.,

$$z_k = \left[G_2^{-1} + (I - G_2^{-1}A) \left(G_1^{-1} + P{A^{(1)}}^{-1} R(I - AG_1^{-1}) \right) \right] r_k. \tag{1.24}$$

The matrix in brackets in (1.24) is the preconditioning matrix M^{-1}. Writing this in the form

$$M^{-1} = G_2^{-1} + G_1^{-1} - G_2^{-1} A G_1^{-1} + (I - G_2^{-1}A)(P{A^{(1)}}^{-1} R)(I - AG_1^{-1}),$$

it can be seen that this matrix is symmetric if A and $A^{(1)}$ are symmetric, $R = P^T$, and $G_2 = G_1^T$. The same holds if ${A^{(1)}}^{-1}$ is replaced by a recursively defined multigrid cycle. Thus, we obtain an appropriate preconditioner for CG if we use a V-cycle or a W-cycle with, say, damped Jacobi smoothing. If the Gauss-Seidel method is used for smoothing, then we must take G_1 to be, say, the lower triangle of A and G_2 to be the upper triangle of A in order to maintain symmetry.

The coarse grid solution step 4 in the above preconditioning step can itself be replaced by one or two steps of PCG with a multigrid preconditioner. However, because the parameters in the PCG algorithm depend on the right-hand side vector, theses parameters will be different at each (outer) iteration k and so the preconditioner will change with k.

In this case, *global* optimality of the CG approximation is lost. However, a steepest-descent-like convergence estimate was given in [28], assuming that the preconditioners M_k do not differ too much from a fixed preconditioner M. See [28] for details.

1.4.3 K-cycle

As mentioned at the end of Section 1.4.1, a multigrid K-cycle can be used in place of the coarse grid solve in the preconditioning step of PCG. A detailed description of the K-cycle with at most two PCG steps at each level l is shown in the following algorithm [24]. Note that the restriction operator is set to be the transpose of the prolongation operator.

K-cycle at level l

Input : The right-hand side vector $r^{(l)}$ (the restriction of the residual at the finer level).
Output : The preconditioned residual $\mathbf{c}^{(l)} = \mathrm{MG}(r^{(l)}, l)$.

1. Presmooth using $G^{(l)}$ (with 0 initial guess) : $\delta^{(l)} = (G^{(l)})^{-1} r^{(l)}$.

2. Compute new residual : $\tilde{r}^{(l)} = \mathbf{r}^{(l)} - A^{(l)}\delta^{(l)}$.

3. Restrict residual : $r^{(l+1)} = (P^{(l)})^T \tilde{r}^{(l)}$.

4. Compute an (approximate) solution $\tilde{x}^{(l+1)}$ to $A^{(l+1)}x^{(l+1)} = r^{(l+1)}$:
   If $l+1$ is the coarsest level, **then** $\tilde{x}^{(l+1)} = (A^{(l+1)})^{-1} r^{(l+1)}$
   Else K-cycle,

   (a) $c^{(l+1)} = \mathrm{MG}(r^{(l+1)}, l+1)$ (preconditioned residual), $g^{(l+1)} = A^{(l+1)} c^{(l+1)}$

   (b) $\rho_1 = (c^{(l+1)})^T g^{(l+1)}$, $\alpha_1 = (c^{(l+1)})^T r^{(l+1)}$ (α_1/ρ_1 is the coefficient a_{k-1} in Algorithm 1.2P).

   (c) $\tilde{r}^{(l+1)} = \mathbf{r}^{(l+1)} - \frac{\alpha_1}{\rho_1} g^{(l+1)}$ (This makes the new residual orthogonal to the previous preconditioned residual.).

   (d) **If** $\|\tilde{r}^{(l+1)}\| \leq t\|r^{(l+1)}\|$, **then** $\tilde{x}^{(l+1)} = \frac{\alpha_1}{\rho_1} c^{(l+1)}$ (if residual is reduced sufficiently, then update $\tilde{x}^{(l+1)}$ and stop after one step). **Else**

   $d^{(l+1)} = \mathrm{MG}\ (\tilde{\mathbf{r}}^{(l+1)}, l+1)$ (new preconditioned residual),
   $h^{(l+1)} = A^{(l+1)} d^{(l+1)}$, $\gamma = (d^{(l+1)})^T g^{(l+1)}$,
   $\beta = (d^{(l+1)})^T h^{(l+1)}$, $\alpha_2 = (d^{(l+1)})^T \tilde{r}^{(l+1)}$,
   $\rho_2 = \beta - \frac{\gamma^2}{\rho_1}$

$\tilde{x}^{(l+1)} = (\frac{\alpha_1}{\rho_1} - \frac{\gamma\alpha_2}{\rho_1\rho_2})c^{(l+1)} + \frac{\alpha_2}{\rho_2}d^{(l+1)}$. (These coefficients make the new residual orthogonal to the previous two preconditioned residuals.)

5. Prolongation : $y^{(l)} = P^{(l)}\tilde{x}^{(l+1)}$.

6. Add correction to $\delta^{(l)}$: $\tilde{\delta}^{(l)} = \delta^{(l)} + y^{(l)}$.

7. Postsmooth using $G^{(l)}$: $c^{(l)} = \tilde{\delta}^{(l)} + (G^{(l)})^{-1}(r^{(l)} - A^{(l)}\tilde{\delta}^{(l)})$.

As seen from the algorithm, at each level, the vector $c^{(l+1)}$ is used to determine the coefficients (e.g., α, ρ) in the Krylov space iteration. This is in contrast to the procedure in the V-cycle, where coefficients in coarse grid iterations are fixed and do not vary with the right-hand side. As mentioned in the previous section, this means that *global* optimality of the CG approximation is lost. Despite this, a bound on the reduction in A-norm of the error was derived in [28] , and, although the bound is overly pessimistic, numerical experiments in [28] show that the K-cycle exhibits much better convergence than the V-cycle or W-cycle on 1D and 2D model problems. The convergence of K-cycle flexible CG appears to be nearly grid size independent. To some extent, it remedies the disadvantages of piecewise constant prolongation.

1.5 Techniques of Analysis

Multigrid methods are based on the recursive use of a two-grid scheme. To analyze multigrid methods, one starts with the analysis of the two-grid method. If the two-grid method converges sufficiently well, then the multigrid method with W-cycle will have similar convergence properties (page 77, [30]). This is not the case for the V-cycle since there are known examples where the two-grid aggregation-based AMG method converges relatively well, whereas the multigrid method with V-cycle scales poorly with the number of levels [20]. Despite this, several works [19, 21] discuss the relation between two-grid convergence and the convergence of V-cycle. Two-grid analysis still provides important information for the convergence of multigrid methods.

In this section the iteration matrix of the two-grid method is derived. Then we state an important theorem from [22]. A slight extension of the theorem will be the essential tool for analyzing the two-grid method on a rotated anisotropic diffusion equation in Chapter 2. Finally the theorem is applied on an example for a demonstration of use of the theorem.

1.5.1 Two Grid Iteration Matrix

Suppose the linear system we wish to solve is $Au = f$. Let the coefficient matrix A be an $N \times N$ symmetric positive definite matrix. In this subsection we will consider the aggregation-based two grid method and derive the iteration matrix. Let P denote the prolongation operator and the restriction operator is $R = P^T$. Assume there are N_C aggregates formed from an automatic aggregation algorithm. Furthermore, the rows that are strongly diagonally dominant are excluded from the aggregation process [24]. Then the prolongation operator P is an $N \times N_C$ matrix of the form (with proper ordering on variables):

$$P = \begin{pmatrix} \mathbf{0}_{m_0} & \mathbf{0}_{m_0} & \dots & \mathbf{0}_{m_0} \\ \mathbf{1}_{m_1} & & & \\ & \mathbf{1}_{m_2} & & \\ & & \ddots & \\ & & & \mathbf{1}_{m_{N_C}} \end{pmatrix} \tag{1.25}$$

where $\mathbf{1}_{m_k}$ are vectors of length m_k with all entries one. The kth vector corresponds to the kth aggregate consisting of m_k fine grid variables. The top block row of 0's corresponds to variables that are not involved in the aggregation procedure. If the number of top block rows is m_0, then we have $\sum_{k=0}^{N_C} m_k = N$. Let G_1 and G_2 denote pre- and post-smoothing matrices. As mentioned in Section 1.2, the preferred smoothers are the weighted Jacobi method where $G_1 = G_2 = \omega^{-1}\text{diag}(A)$, or the Gauss-Seidel method, where $G_1 = G_2^T =$ lower triangle(A). Denote the number of pre- and post-smoothing steps by ν_1 and ν_2, respectively.

The two-grid method starts with performing ν_1 pre-smoothing steps with matrix G_1. Given an approximation at the kth iteration, $u_k \equiv u_{k,0}$, pre-smoothing steps generate ν_1

new approximations $u_{k,j}$ via the formula

$$u_{k,j} = u_{k,j-1} + G_1^{-1}(f - Au_{k,j-1}), \qquad j = 1, ..., \nu_1. \tag{1.26}$$

Subtracting (1.26) from the true solution $A^{-1}f$ on both sides, the error propagation equation for each pre-smoothing step is $e_{k,j} = (I - G_1^{-1}A)e_{k,j-1}$. Thus the error after ν_1 pre-smoothing steps satisfies $e_{k,\nu_1} = (I - G_1^{-1}A)^{\nu_1} e_{k,0}$. The residual is $r_{k,\nu_1} \equiv f - Au_{k,\nu_1}$.

In the "coarse grid correction" step, the residual r_{k,ν_1} is restricted to the coarse grid by forming $P^T r_{k,\nu_1}$, Then one solves for δ in the linear system

$$A_C \delta = P^T r_{k,\nu_1}, \tag{1.27}$$

prolongates δ, then adds the result:

$$u_{k,\nu} = u_{k,\nu_1} + P\delta \tag{1.28}$$

Since $\delta = A_C^{-1} P^T r_{k,\nu_1}$, the whole procedure is $u_{k,\nu} = u_{k,\nu_1} + P A_C^{-1} P^T r_{k,\nu_1}$ and the error propagation relation for "coarse grid correction" is $e_{k,\nu} = (I - P A_C^{-1} P^T A) e_{k,\nu_1}$. Combining with the error equation arising from pre-smoothing, we have error propagation equation for "pre-smoothing" and "coarse grid correction": $e_{k,\nu} = (I - P A_C^{-1} P^T A)(I - G_1^{-1}A)^{\nu_1} e_{k,0}$. Then the post-smoothing steps are performed with splitting G_2 for ν_2 times and the error equation is $e_{k,\nu_2} = (I - G_2^{-1}A)^{\nu_2} e_{k,\nu}$. Letting $e_{k,\nu_2} \equiv e_{k+1}$, we finally have the error propagation equation for the two-grid method:

$$e_{k+1} = (I - G_2^{-1}A)^{\nu_2}(I - P A_C^{-1} P^T A)(I - G_1^{-1}A)^{\nu_1} e_k. \tag{1.29}$$

The matrix in equation (1.29) is the iteration matrix of two-grid method, and will be denoted by E_{TG}. A very important quantity is the spectral radius of E_{TG}, denoted by $\rho(E_{TG})$. The spectral radius $\rho(E_{TG})$ governs the asymptotic convergence of the two-grid method. Although, in general, it is **not** the relevant quantity for describing short-time behavior (which, hopefully, is all that we will see!), in some cases the spectral radius gives a measure of the amount by which the A-norm of the error is reduced at each iteration. To see this, recall the A-norm of a vector v is $\|v\|_A \equiv < Av, v >^{1/2} = \|A^{1/2} v\|$, where $\|\cdot\|$

without a subscript denotes the 2-norm. Multiplying by $A^{1/2}$ on both sides of (1.29), it follows that $A^{1/2}e_{k+1} = A^{1/2}E_{TG}A^{-1/2}(A^{1/2}e_k)$ and hence

$$\|e_{k+1}\|_A \leq \|A^{1/2}E_{TG}A^{-1/2}\| \cdot \|e_k\|_A$$

The matrix $A^{1/2}E_{TG}A^{-1/2}$ has the same eigenvalues as E_{TG} so that *if* this matrix is symmetric then its 2-norm is $\rho(E_{TG})$. Furthermore, the matrix $A^{1/2}E_{TG}A^{-1/2}$ can be written in the form

$$(I - A^{1/2}G_2^{-1}A^{1/2})^{\nu_2}(I - A^{1/2}PA_C^{-1}P^TA^{1/2})(I - A^{1/2}G_1^{-1}A^{1/2})^{\nu_1}. \tag{1.30}$$

Hence if $\nu_1 = \nu_2$ and $G_1 = G_2^T$, the matrix in (1.30) is symmetric and the A-norm of the error is reduced at each step by at least the factor $\rho(E_{TG})$.

1.5.2 Analysis of Napov and Notay

In this subsection we describe the analysis of Napov and Notay in [22]. For convenience, let an N by N matrix X be defined by

$$I - X^{-1}A = (I - G_1^{-1}A)^{\nu_1}(I - G_2^{-1}A)^{\nu_2}. \tag{1.31}$$

Matrix X can be viewed as a "global smoother", it has the same effect in one iteration as ν_2 of post-smoothing followed by ν_1 of pre-smoothing. In the case that only one post-smoothing is done, $\nu_1 = 0$ and $\nu_2 = 1$ then $X = G_2$. If one step of pre-smoothing and one step of post-smoothing is done then $\nu_1 = \nu_2 = 1$ and

$$I - X^{-1}A = (I - G_1^{-1}A)(I - G_2^{-1}A). \tag{1.32}$$

so that $X^{-1} = G_1^{-1}(G_2 + G_1 - A)G_2^{-1}$. We always assume that this matrix is invertible and SPD. An example is when $G_1 = G_2^T =$ the lower triangular part of the SPD matrix A then $X = G_2(\text{diag}(A))^{-1}G_1$, an SPD matrix. For any N by N matrix Y , define

$$\pi_Y \equiv P(P^TYP)^{-1}P^TY \equiv PY_C^{-1}P^TY. \tag{1.33}$$

The first theorem states the bound on $\rho(E_{TG})$ and is proved in [22]. It is a slight generalization of Theorem 4.2 in [14].

Theorem 1.1. Let A be an N by N SPD matrix and let P be an N by N_C matrix of rank $N_C < N$. Let G_1, ν_1, G_2, ν_2 be such that X, defined in (1.32), is SPD and let E_{TG} be the two-grid iteration matrix defined in (1.29). Then

$$\rho(E_{TG}) \leq \max\left\{\lambda_{max}(X^{-1}A) - 1, 1 - \frac{1}{\mu_X}\right\} \tag{1.34}$$

where $\lambda_{max}(\cdot)$ denotes the (algebraically) largest eigenvalue, and

$$\mu_X = \lambda_{max}(A^{-1}X(I - \pi_X)). \tag{1.35}$$

Moreover, for any N by N SPD matrix D,

$$\mu_X \leq \lambda_{max}(D^{-1}X) \cdot \mu_D, \qquad \mu_D \equiv \lambda_{max}(A^{-1}D(I - \pi_D)). \tag{1.36}$$

In particular, if $G_1 = G_2 = \omega^{-1}D$ with $\omega^{-1} \geq \lambda_{max}(D^{-1}A)$, then

$$\rho(E_{TG}) = 1 - \frac{1}{\mu_X}, \tag{1.37}$$

and $\mu_X \leq \omega^{-1}\mu_D$, with equality if $\nu_1 + \nu_2 = 1$. □

In Chapter 2, we will apply this theorem on the two-grid method with weighted Jacobi smoothing, $G_1 = G_2 = \omega^{-1}D$ where D is the diagonal of A. Then the bound on μ_X depends on two factors: $\lambda_{max}(D^{-1}X)$ and μ_D, and the first factor is relatively easy to find. The analysis is carried out on two cases. In the first case, $\nu_1 + \nu_2 = 1$, where only one pre- or post-smoothing is performed. Then $X = \omega^{-1}D$ and $\lambda_{max}(D^{-1}X) = \omega^{-1}$. If $\nu_1 = \nu_2 = 1$, then $X = \omega^{-1}D(2D - \omega A)^{-1}D$ and $\lambda_{max}(D^{-1}X) = \omega^{-1}\lambda_{max}((2D - \omega A)^{-1}D)$. When $(2D - \omega A)^{-1}D$ is strongly diagonally dominant, $\lambda_{max}(D^{-1}X)$ is easy to estimate using Gerschgorin's theorem. For example, consider the 1D discretization of the Laplacian operator with N variables:

$$A = \begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix}. \tag{1.38}$$

$D = 2I$ and $\lambda_{max}(D^{-1}A) < 2$, so $\omega^{-1} = 2$ is a reasonable choice. Now the matrix $2D - \omega A$ has 3 on its main diagonal and 1/2 on the first sub- and super-diagonal. By Gerschgorin's theorem the smallest eigenvalue of this matrix is greater than or equal to 2, hence the largest eigenvalue of its inverse is less than 1/2. This leads to $\lambda_{max}(D^{-1}X) \leq 2 = \omega^{-1}$.

The second factor $\mu_D = \lambda_{max}(A^{-1}D(I - \pi_D))$ involves prolongation matrix P. It gives a measure of how effectively eigenvectors associated with the large eigenvalues of A^{-1} are damped by the coarse grid correction. As the definition of μ_D involves the inverse of the matrix A, it seems difficult to determine the quantity on the right-hand side of (1.36), so the usage of the theorem on estimating the spectral radius $\rho(E_{TG})$ is limited. However, the next theorem in [22] explains how to estimate the quantity μ_D by evaluating a measure associated with individual aggregates.

The theorem is based on finding a suitable splitting of matrix A. Suppose $A = A_b + A_r$ such that A_b and A_r are both symmetric and nonnegative definite and A_b is block diagonal, with block size equal to the corresponding aggregate size:

$$A_b = \begin{pmatrix} A^{(0)} & & & \\ & A^{(1)} & & \\ & & \ddots & \\ & & & A^{(N_C)} \end{pmatrix} \tag{1.39}$$

where $A^{(k)}$, $k = 0, 1, ..., N_C$, is of size m_k by m_k. When A is diagonally dominant with positive diagonal entries, one possible splitting of A is to let the block $A^{(k)}$ correspond to the variables in the kth aggregate and be equal to the corresponding diagonal block of A, with something subtracted from the diagonal to keep both A_b and A_r weakly diagonally dominant. For example, if

$$A = \begin{pmatrix} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & -1 & 2 & -1 & & \\ & & & -1 & 2 & -1 & \\ & & & & -1 & 2 \end{pmatrix} \tag{1.40}$$

and successive pairwise aggregation is used, then a possible splitting is

$$A_b = \left(\begin{array}{cc|ccc|ccc} 2 & -1 & & & & & & \\ -1 & 1 & & & & & & \\ - & - & - & - & - & - & - & - \\ & & & 1 & -1 & & & \\ & & & -1 & 1 & & & \\ - & - & - & - & - & - & - & - \\ & & & & & & 1 & -1 \\ & & & & & & -1 & 2 \end{array}\right) \tag{1.41}$$

$$A_r = \left(\begin{array}{cc|ccc|ccc} & & & & & & & \\ & 1 & & -1 & & & & \\ - & - & - & - & - & - & - & - \\ & -1 & & 1 & & & & \\ & & & & 1 & & -1 & \\ - & - & - & - & - & - & - & - \\ & & & & -1 & & 1 & \\ & & & & & & & \end{array}\right) \tag{1.42}$$

The diagonal blocks of A_b are the corresponding diagonal blocks of A, with the sum of absolute values of the remaining entries in each row or column of A subtracted from the corresponding diagonal entry of A_b. In this manner, both A_b and A_r are weakly diagonally dominant, hence nonnegative definite.

The aggregation algorithm in [24] treats the rows which are diagonally dominant as one aggregate, denoted by $A^{(0)}$. In this situation we reorder the variables in matrix A so that

the first variable and the last variable are placed first, followed by variables 2 and 3 then 4 and 5. The matrix A takes the form

$$A = \begin{pmatrix} 2 & 0 & -1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & -1 \\ -1 & 0 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & -1 & 0 & 0 & -1 & 2 \end{pmatrix}. \tag{1.43}$$

With the same principle as the previous example, treating first two variables as one aggregate, we can split A as

$$A_b = \left(\begin{array}{cc|cc|cc} 1 & & & & & \\ & 1 & & & & \\ \hline & & 1 & -1 & & \\ & & -1 & 1 & & \\ \hline & & & & 1 & -1 \\ & & & & -1 & 1 \end{array}\right) \tag{1.44}$$

$$A_r = \left(\begin{array}{cc|cc|cc} 1 & & -1 & & & \\ & 1 & & & & -1 \\ \hline -1 & & 1 & & & \\ & & & 1 & -1 & \\ \hline & & & -1 & 1 & \\ & -1 & & & & 1 \end{array}\right) \tag{1.45}$$

Another splitting of A can be constructed in the finite element context when each aggregate contains all and only the nodes belonging to one element or to the union of several elements. We describe such splitting in chapter 2. Once the splitting is established, the following theorem from [22] gives a bound on the 'global' parameter μ_D in (1.36) using

'local' quantities associated with each aggregate k.

Theorem 1.2. Using the notation of Theorem 1.1, let $A = A_b + A_r$ be a splitting of the SPD matrix A such that A_b and A_r are both nonnegative definite and A_b has the form (1.39). Assume that D in Theorem 1.1 also has the block diagonal form:

$$D = \begin{pmatrix} D^{(0)} & & & \\ & D^{(1)} & & \\ & & \ddots & \\ & & & D^{(N_c)} \end{pmatrix}. \tag{1.46}$$

Each block $D^{(k)}$ is m_k by m_k, $k = 0, 1, ..., N_C$. Define

$$\mu_D^{(0)} = \begin{cases} 0 & \text{if } m_0 = 0 \\ \sup\limits_{\mathbf{v}^{(0)} \in \mathbb{R}^{m_0} \setminus \mathcal{N}(A^{(0)})} \dfrac{{\mathbf{v}^{(0)}}^T D^{(0)} \mathbf{v}^{(0)}}{{\mathbf{v}^{(0)}}^T A^{(0)} \mathbf{v}^{(0)}} & \text{if } m_0 > 0 \end{cases} \tag{1.47}$$

and for $k = 1$, ..., N_C,

$$\mu_D^{(k)} = \begin{cases} 0 & \text{if } m_k = 1 \\ \sup\limits_{\mathbf{v}^{(k)} \in \mathbb{R}^{m_k} \setminus \mathcal{N}(A^{(k)})} \dfrac{{\mathbf{v}^{(k)}}^T D^{(k)} (I - \pi_D^{(k)}) \mathbf{v}^{(k)}}{{\mathbf{v}^{(k)}}^T A^{(k)} \mathbf{v}^{(k)}} & \text{if } m_k > 1 \end{cases} \tag{1.48}$$

where

$$\pi_D^{(k)} = \mathbf{p}^{(k)} ({\mathbf{p}^{(k)}}^T D^{(k)} \mathbf{p}^{(k)})^{-1} {\mathbf{p}^{(k)}}^T D^{(k)} \tag{1.49}$$

and $\mathbf{p}^{(k)}$ is the vector of 1's in column k of P. Then

$$\mu_D \leq \max_{k=0,...,N_c} \mu_D^{(k)}. \tag{1.50}$$

□

As an example, we apply this theorem to the matrix (1.40) with splitting in (1.41- 1.42). Let D be the diagonal of A, so $D = 2I$. Then $\mathbf{p}^{(k)} = (1, 1)^T$ and

$$\pi_D^{(k)} = (1,1)^T \frac{1}{4} (1,1) \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \qquad k = 1, 2, 3. \tag{1.51}$$

Hence

(1.52)

$$D^{(k)}(I - \pi_D^{(k)}) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \left[\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \right] = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \qquad k = 1, 2, 3.$$

For $k = 2$, $A^{(2)}$ is singular but it is in fact $D^{(2)}(I - \pi_D^{(2)})$, so $\mu_D^{(2)}$ is clearly 1. When $k = 1$,

$$\lambda_{max} \left(\begin{pmatrix} 2 & -1 \\ -1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \right) = 1. \tag{1.53}$$

Hence $\mu_D^{(1)} = 1$. Similarly $\mu_D^{(3)} = 1$. Theorem 1.2 implies that $\mu_D \leq 1$. Combining with the bound on $\lambda_{max}(D^{-1}X)$ when $\nu_1 + \nu_2 = 1$ or $\nu_1 = \nu_2 = 1$, we obtain $\mu_X \leq 2$. Finally by Theorem 1.1

$$\rho(E_{TG}) \leq \frac{1}{2}.$$

The example in (1.40) - (1.42) has $N = 6$. For much larger N, $\mu_D \leq 1$ by Theorem 1.2 since for those aggregates not connected to the two end points it is obvious that $\mu_D^{(k)} = \mu_D^{(2)} = 1$. Theorem 1.1 again implies that $\rho(E_{TG}) \leq 1/2$. In fact, this is the limit as $N \to \infty$ of the spectral radius of the two-grid iteration matrix when A is an N by N matrix with 2's on its main diagonal and -1's on the first sub- and super-diagonals. The bound is sharp in this case.

Chapter 2

ANALYSIS OF AGGREGATION-BASED ALGEBRAIC TWO-GRID METHOD FOR PROBLEMS ARISING FROM A ROTATED ANISOTROPIC DIFFUSION EQUATION

In this chapter we use the analysis tool in [22], described in Section 1.5.2, to study an aggregation-based algebraic two-grid method applied to the equations arising from a bilinear finite element approximation for a rotated anisotropic diffusion equation. We first prove a slight generalization of Theorem 1.2, which is used to derive bounds on the spectral radius of the iteration matrix and to show why current automatic aggregation procedures often fail to identify appropriate aggregates when the direction of anisotropy is not aligned with the grid. This leads to poorly converging iterations, with the number of steps increasing with the grid size. We then use the analysis to devise a better procedure for choosing aggregates and we demonstrate its performance on problems with a range of angles and degrees of anisotropy. **The material in this chapter is new and has been submitted for publication [8]. This, and the material in Chapter 3 on parallelization, are the main contributions of this thesis**.

2.1 Rotated Anisotropic Diffusion Equation

In this chapter we deal with the rotated anisotropic diffusion equation:

$$-(\epsilon\cos^2\theta+\sin^2\theta)\frac{\partial^2 u}{\partial x^2}-2(1-\epsilon)\cos\theta\sin\theta\frac{\partial^2 u}{\partial y\partial x}-(\cos^2\theta+\epsilon\sin^2\theta)\frac{\partial^2 u}{\partial y^2}=f \tag{2.1}$$

$$\text{on}\quad \Omega=[0,1]\times[0,1],\qquad u(x,0)=u(x,1)=u(0,y)=u(1,y)=0$$

By rotating the (x,y) coordinates an angle θ counterclockwise,

$$\begin{pmatrix}\xi\\ \eta\end{pmatrix}=\begin{pmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix}, \tag{2.2}$$

the derivatives in equation (2.1) in the new coordinates (ξ, η) are:

$$\begin{aligned}
\frac{\partial^2 u}{\partial x^2} &= \cos^2\theta \frac{\partial^2 u}{\partial \xi^2} + 2\cos\theta\sin\theta \frac{\partial^2 u}{\partial \xi \partial \eta} + \sin^2\theta \frac{\partial^2 u}{\partial \eta^2} \\
\frac{\partial^2 u}{\partial y \partial x} &= -\sin\theta\cos\theta \frac{\partial^2 u}{\partial \xi^2} + (\cos^2\theta - \sin^2\theta) \frac{\partial^2 u}{\partial \xi \partial \eta} + \sin\theta\cos\theta \frac{\partial^2 u}{\partial \eta^2} \\
\frac{\partial^2 u}{\partial y^2} &= \sin^2\theta \frac{\partial^2 u}{\partial \xi^2} - 2\cos\theta\sin\theta \frac{\partial^2 u}{\partial \xi \partial \eta} + \cos^2\theta \frac{\partial^2 u}{\partial \eta^2}.
\end{aligned}$$

After some algebra, equation (2.1) becomes $-\epsilon u_{\xi\xi} - u_{\eta\eta} = f$, an anisotropic diffusion equation whose direction of anisotropy is aligned with the (ξ, η) axes. Problems of this sort can be a challenge for aggregation-based algebraic multigrid methods. The major difficulty arises from the fact that the direction of anisotropy is not aligned with the grid lines. The existing automatic aggregation algorithm in [23], consisting of two or three consecutive pairwise aggregation procedures, does not choose the appropriate aggregates (as will be shown later). However, for the case $\theta = \pi/4$, numerical experiments show that if the aggregates can be chosen to align with the direction of anisotropy, then grid-size independent and ϵ-independent convergence rates can be obtained with a two-grid method. This will be proved in this chapter.

2.2 Derivation of the Finite Element Discretization

Equation (2.1) can be solved using a piecewise bilinear finite element approximation on a square grid with uniform spacing h in each direction. To avoid lengthy and complicated notations, letting

$$a = \epsilon\cos^2\theta + \sin^2\theta,$$

$$b = (1-\epsilon)\cos\theta\sin\theta,$$

$$c = (\cos^2\theta + \epsilon\sin^2\theta),$$

equation (2.1) can be written in the form:

$$-\nabla \cdot \begin{pmatrix} a & b \\ b & c \end{pmatrix} \nabla u = f \tag{2.3}$$

Let $\phi_j(x,y)$ denote the standard bilinear basis function with value 1 at node j and 0 at all other nodes, and write the approximate solution $u(x,y) = \sum_{j=1}^{N} u_j \phi_j(x,y)$. The coefficients u_j can be determined by solving the equations

$$-\sum_{j=1}^{N} u_j \int_0^1 \int_0^1 (\nabla\phi_i) \cdot \left[\begin{pmatrix} a & b \\ b & c \end{pmatrix} \nabla\phi_j \right] dxdy = \int_0^1 \int_0^1 f\phi_i \, dxdy, \qquad i = 1, ..., N. \tag{2.4}$$

The 4×4 element matrix $\mathcal{A}$ is

$$\frac{1}{6} \begin{pmatrix} 2a+3b+2c & -2a+c & a-2c & -a-3b-c \\ -2a+c & 2a-3b+2c & -a+3b-c & a-2c \\ a-2c & -a+3b-c & 2a-3b+2c & -2a+c \\ -a-3b-c & a-2c & -2a+c & 2a+3b+2c \end{pmatrix}. \tag{2.5}$$

When these element matrices are assembled into a global stiffness matrix (and multiplied by 6), the stencil of the finite element discretization becomes

$$\begin{pmatrix} -a+3b-c & 2a-4c & -a-3b-c \\ -4a+2c & 8(a+c) & -4a+2c \\ -a-3b-c & 2a-4c & -a+3b-c \end{pmatrix} =$$

$$\begin{bmatrix} -(1+\epsilon)+3(1-\epsilon)\cos\theta\sin\theta & (-4+2\epsilon)\cos^2\theta+(-4\epsilon+2)\sin^2\theta & -(1+\epsilon)-3(1-\epsilon)\cos\theta\sin\theta \\ (-4\epsilon+2)\cos^2\theta+(-4+2\epsilon)\sin^2\theta & 8(1+\epsilon) & (-4\epsilon+2)\cos^2+(-4+2\epsilon)\sin^2\theta \\ -(1+\epsilon)-3(1-\epsilon)\cos\theta\sin\theta & (-4+2\epsilon)\cos^2\theta+(-4\epsilon+2)\sin^2\theta & -(1+\epsilon)+3(1-\epsilon)\cos\theta\sin\theta \end{bmatrix}. \tag{2.6}$$

For $\theta = \pi/4$, the stencil is

$$\begin{bmatrix} \frac{1}{2}-\frac{5}{2}\epsilon & -1-\epsilon & -\frac{5}{2}+\frac{1}{2}\epsilon \\ -1-\epsilon & 8+8\epsilon & -1-\epsilon \\ -\frac{5}{2}+\frac{1}{2}\epsilon & -1-\epsilon & \frac{1}{2}-\frac{5}{2}\epsilon \end{bmatrix}. \tag{2.7}$$

Note that when $\epsilon < 1/5$, the coefficient matrix A has some positive off-diagonal entries (northwest and southeast neighbors). The strongest negative connection is along the southwest-northeast direction when $\epsilon < 1$.

2.3 An Extension of Analysis

As mentioned in Section 1.5.2, the effectiveness of Theorem 1.2 depends on the matrix splitting $A = A_b + A_r$. By a method suggested to us by Y. Notay [26], we will form "tentative aggregates" from 4 by 4 squares of nodes, as pictured in Figure 2.1. In the

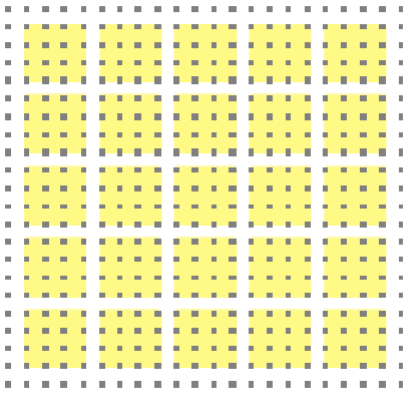


Figure 2.1

coarsening process the rows that are strongly diagonally dominant are not included in a tentative aggregate, because they couple to Dirichlet boundary points; these make up the block $A^{(0)}$. Each tentative aggregate contains nine elements and sixteen nodes. The blocks $A^{(k)}$ are therefore 16×16. Element matrices corresponding to each of the nine elements in a tentative aggregate will be assembled to form a block of A_b, and the remaining element matrices will be assembled to form A_r. Since each of the element matrices is nonnegative definite, the matrices A_b and A_r will be as well. Note that each block of A_b (except $A^{(0)}$) is the finite element discretization of a Neumann problem on the potential aggregate, hence has null space consisting of the constant vectors (assuming $0 < \epsilon \leq 1$ in (2.1)).

The tentative aggregate will be split into several subsets of nodes to form the actual aggregates. For example, when $\theta = \pi/4$, a natural choice is to aggregate nodes along diagonal lines, as pictured in Figure 2.2. This means that the matrix structure is slightly different

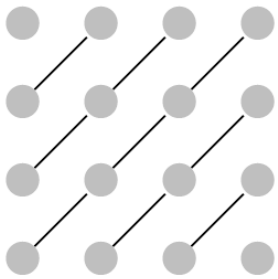


Figure 2.2

from that described in [22], where it was assumed that each aggregate size was the same as the block size in A_b. This requires a minor change in the statement of Theorem 1.2.

Theorem 1.2'. Using the notation of Theorem 1.1, let $A = A_b + A_r$ be a splitting of the SPD matrix A such that A_b and A_r are both nonnegative definite and A_b has the form

$$A_b = \begin{pmatrix} A^{(0)} & & & \\ & A^{(1)} & & \\ & & \ddots & \\ & & & A^{(N_b)} \end{pmatrix} \tag{2.8}$$

where each block $A^{(k)}$ is s_k by s_k, $k = 0, ..., N_b$. Assume that D in Theorem 1.1 also has the block diagonal form:

$$D = \begin{pmatrix} D^{(0)} & & & \\ & D^{(1)} & & \\ & & \ddots & \\ & & & D^{(N_b)} \end{pmatrix} \tag{2.9}$$

where each block is s_k by s_k. Define

$$\mu_D^{(0)} = \begin{cases} 0 & \text{if } s_0 = 0 \\ \sup\limits_{\mathbf{v}\in\mathbb{R}^{s_0}\setminus\mathcal{N}(A^{(0)})} \dfrac{\mathbf{v}^T D^{(0)}\mathbf{v}}{\mathbf{v}^T A^{(0)}\mathbf{v}} & \text{if } s_0 > 0 \end{cases} \tag{2.10}$$

and for $k = 1, ..., N_b$,

$$\mu_D^{(k)} = \begin{cases} 0 & \text{if } s_k = 1 \\ \sup\limits_{\mathbf{v}\in\mathbb{R}^{s_k}\setminus\mathcal{N}(A^{(k)})} \dfrac{\mathbf{v}^T D^{(k)}(I-\pi_D^{(k)})\mathbf{v}}{\mathbf{v}^T A^{(k)}\mathbf{v}} & \text{if } s_k > 0 \end{cases} \tag{2.11}$$

where

$$\pi_D^{(k)} = P^{(k)}({P^{(k)}}^T D^{(k)} P^{(k)})^{-1} {P^{(k)}}^T D^{(k)} \tag{2.12}$$

and $P^{(k)}$, represents the *block* of P corresponding to the kth tentative aggregate. Then

$$\mu_D \leq \max_{k=0,...,N_b} \mu_D^{(k)}. \tag{2.13}$$

To obtain useful information from the theorem when $s_k > 1$, $k = 1, ..., N_b$, the null space of $A^{(k)}$ must be a subset of the null space of $D^{(k)}(I - \pi_D^{(k)})$; otherwise $\mu_D^{(k)} = \infty$ in (2.11). The proof of this slightly generalized theorem is almost identical to the one in [22].

Proof of the theorem: Assume without loss of generality that each $\mu_D^{(k)} < \infty$, $k = 0, 1, , ..., N_b$; otherwise the result is trivial. If $s_0 > 0$ then this implies that $A^{(0)}$ is positive definite so the expression for $\mu_D^{(0)}$ becomes

$$\mu_D^{(0)} = \begin{cases} 0 & \text{if } s_0 = 0 \\ \sup\limits_{\mathbf{v}\neq 0} \dfrac{\mathbf{v}^T D^{(0)}\mathbf{v}}{\mathbf{v}^T A^{(0)}\mathbf{v}} & \text{if } s_0 > 0 \end{cases}. \tag{2.14}$$

To establish (2.13), first note that

$$D(I-\pi_D) = \begin{pmatrix} D^{(0)} & & & \\ & D^{(1)}(I-\pi_D^{(1)}) & & \\ & & \ddots & \\ & & & D^{(N_b)}(I-\pi_D^{(N_b)}) \end{pmatrix}. \tag{2.15}$$

Hence expression (1.36) for μ_D can be written as

$$\begin{aligned} \mu_D &= \lambda_{\max}(A^{-1}D(I-\pi_D)) = \max_{\mathbf{v}\in\mathbb{R}^N,\mathbf{v}\neq 0} \frac{\mathbf{v}^T D(I-\pi_D)\mathbf{v}}{\mathbf{v}^T A\mathbf{v}} \\ &= \max_{\mathbf{v}\neq 0} \frac{\mathbf{v}^T D(I-\pi_D)\mathbf{v}}{\mathbf{v}^T A_b\mathbf{v} + \mathbf{v}^T A_r\mathbf{v}} \\ &= \max_{\mathbf{v}\neq 0} \frac{\mathbf{v}^{(0)^T} D^{(0)}\mathbf{v}^{(0)} + \sum_{k=1}^{N_b} \mathbf{v}^{(k)^T} D^{(k)}(I-\pi_D^{(k)})\mathbf{v}^{(k)}}{\sum_{k=0}^{N_b} \mathbf{v}^{(k)^T} A^{(k)}\mathbf{v}^{(k)} + \mathbf{v}^T A_r\mathbf{v}}. \end{aligned} \tag{2.16}$$

where we have written $\mathbf{v}$ in the block form $\mathbf{v} = (\mathbf{v}^{(0)^T}, \mathbf{v}^{(1)^T}, ..., \mathbf{v}^{(N_b)^T})^T$. Let $w = (w^{(0)^T}, w^{(1)^T}, ..., w^{(N_b)^T})^T$ be the vector that maximizes the quotient in (2.16). First note that if $w^{(k)^T} A^{(k)} w^{(k)} = 0$ for all $k = 0, 1, ..., N_b$ in the sum in the denominator of (2.16), then the numerator of (2.16) would also be 0 since each $\mu_D^{(k)}$ is finite. If this were the case, then since A is SPD, the other term in the denominator, $w^T A_r w$, would have to be positive and then $\mu_D = 0$. In this case, the result (2.13) holds trivially. Therefore we can assume that at least one of the terms $w^{(k)^T} A^{(k)} w^{(k)}$ is nonzero and, since each $A^{(k)}$ is nonnegative definite, this implies that the sum in the denominator of (2.16) is positive. Since A_r is nonnegative definite we can write

$$\mu_D \leq \frac{w^{(0)^T} D^{(0)} w^{(0)} + \sum_{k=1}^{N_b} w^{(k)^T} D^{(k)}(I-\pi_D^{(k)}) w^{(k)}}{\sum_{k=0}^{N_b} w^{(k)^T} A^{(k)} w^{(k)}}. \tag{2.17}$$

Since the numerator and denominator each consist of a sum of nonnegative terms (or positive terms if we eliminate any zeros), the ratio of these sums is less than or equal to the largest ratio of the terms. [To see this, note that if $a_1, ..., a_n, b_1, ..., b_n > 0$, then $(\sum_{i=1}^n a_i)/(\sum_{i=1}^n b_i) = \sum_{i=1}^n (a_i/b_i) \cdot (b_i/\sum_{j=1}^n b_j)$, which is a convex combination of the

ratios a_i/b_i, hence less than or equal to the maximum of these ratios.] Thus we can write

$$\begin{aligned}
\mu_D &\leq \max_{k=0,\ldots,N_b} \frac{w^{(k)^T} D^{(k)}(I - \pi_D^{(k)})w^{(k)}}{w^{(k)^T} A^{(k)} w^{(k)}} \qquad (\text{ where } \pi_D^{(0)} \equiv 0) \\
&\leq \max_{k=0,\ldots,N_b} \sup_{\mathbf{v}\in\mathbb{R}^{s_k}\setminus\mathcal{N}(A^{(k)})} \frac{\mathbf{v}^T D^{(k)}(I - \pi_D^{(k)})\mathbf{v}}{\mathbf{v}^T A^{(k)}\mathbf{v}} \\
&= \max_{k=0,\ldots,N_b} \mu_D^{(k)}. \qquad (2.18)
\end{aligned}$$

□

For the diagonal aggregates pictured in Figure 2.2, if the 4 nodes along the main diagonal within a tentative aggregate are numbered first, followed by numbering along the first sub-diagonal and super-diagonal, each consisting of 3 nodes, then numbering along the diagonals two spaces below and above the main one, etc., then the block of P corresponding to the kth tentative aggregate is

$$P^{(k)^T} = \begin{pmatrix}
1 & 1 & 1 & 1 & & & & & & & & & & & \\
 & & & & 1 & 1 & 1 & & & & & & & & \\
 & & & & & & & 1 & 1 & 1 & & & & & \\
 & & & & & & & & & & 1 & 1 & & & \\
 & & & & & & & & & & & & 1 & 1 & \\
 & & & & & & & & & & & & & & 1 \\
 & & & & & & & & & & & & & & & 1
\end{pmatrix}. \qquad (2.19)$$

Thus each $P^{(k)}$ is a 16 by 7 matrix and $\pi_D^{(k)}$ is a 16×16 matrix.

2.4 Result for $\theta = \pi/4$

In this section we consider the rotated anisotropic diffusion equation (2.1) with $\theta = \pi/4$ and $\epsilon \in [0, 1]$ and investigate the bound on $\rho(E_{TG})$. For simplicity of analysis the smoother is damped Jacobi relaxation. The quantities $\mu_D^{(k)}$, $k = 0, 1, ..., N_b$, in Theorem 1.2' are computed for various aggregates chosen from nodes of the tentative aggregate (diagonal aggregates, box aggregates) to obtain a bound on μ_D via (2.13). Combining μ_D and an

estimate of $\lambda_{max}(D^{-1}X)$ one obtains a bound on μ_X via equation (1.35). Finally these values are used in (1.37) to establish the bound for spectral radius $\rho(E_{TG})$. The bounds are then compared with actual computed values to determine if they are realistic. We consider a two-grid method when one pre-smoothing and one post-smoothing are performed and also when just one pre- or one post-smoothing is performed. The damping factor ω should satisfy the requirement in Theorem 1.1. Using the stencil in (2.7), the sum of the absolute values of off-diagonal terms is $8 + 8\epsilon$ when $\epsilon > 1/5$ and $10 - 2\epsilon$ when $\epsilon \leq 1/5$. Hence by Gerschgorin's theorem

$$\lambda_{max}(D^{-1}A) \leq \begin{cases} (9 + 3\epsilon)/(4 + 4\epsilon) & \text{if } \epsilon \leq 1/5 \\ 2 & \text{if } \epsilon > 1/5 \end{cases}, \tag{2.20}$$

so these will be the values used for ω^{-1}. Furthermore, $D = \text{diag}(A)$ and the 16×16 block diagonal $D^{(k)}$ corresponding to kth aggregate is $D^{(k)} = 8(1 + \epsilon)I_{16\times 16}$ for all $k = 1, ..., N_b$.

Note that from Figure 2.1, the number of interior nodes in each direction is $4m + 2$, where m is a positive integer. This number allows that each tentative aggregate is of size 4 by 4. On the other hand, the values in the stencil do not depend on the location in the domain, so the blocks $A^{(k)}$ and $D^{(k)}$, $k = 1, ..., N_b$, in Theorem 1.2' are all identical. Thus, except for the computation of $\mu_D^{(0)}$ which is easily estimated with Gerschgorin's theorem, the problem of bounding μ_D reduces to the problem of finding the largest (determined) generalized eigenvalue of a single pair of 16 by 16 matrices. (The fact that the expression on the right-hand side of (2.11) is a generalized eigenvalue follows from a generalization of the Courant-Fischer minimax theorem [15].)

To obtain useful information from $\mu_D^{(k)}$, it must be the case that the null space of $A^{(k)}$ is a subset of the null space of $D^{(k)}(I - \pi_D^{(k)})$. Symbolic computations indicate that when $\epsilon > 0$ the null space of $A^{(k)}$ consists of the constant vectors, while when $\epsilon = 0$ the null space of $A^{(k)}$ consists of linear combinations of the constant vectors and vectors of the form

$$(0, 0, 0, 0, 1, 1, 1, -1, -1, -1, 2, 2, -2, -2, 3, -3)^T.$$

(ordering nodes along the main diagonal of the tentative aggregate first, then along the first upper and first lower diagonal, etc.) On the other hand, the null space of $D^{(k)}(I - \pi_D^{(k)})$

consists of all vectors that are constant along diagonals of tentative aggregates. Therefore for the aggregates pictured in Figure 2.2, $\mathcal{N}(A^{(k)}) \subset \mathcal{N}(D^{(k)}(I - \pi_D^{(k)}))$; but for any other aggregates, such as box aggregates, this will not be the case.

Knowing the null spaces of the matrices involved in (2.11) of Theorem 1.2' makes the computation of generalized eigenvalues easy. The problem can be converted to a problem involving a positive definite matrix $Z^* A^{(k)} Z$ and the semidefinite matrix $Z^* D^{(k)}(I - \pi_D^{(k)})Z$ where Z has dimensions s_k by $\text{rank}(A^{(k)})$ and the columns of Z span the range of $A^{(k)}$ [15]. The relation between $\mu_D^{(k)}$ and ϵ is shown in Figure 2.3:

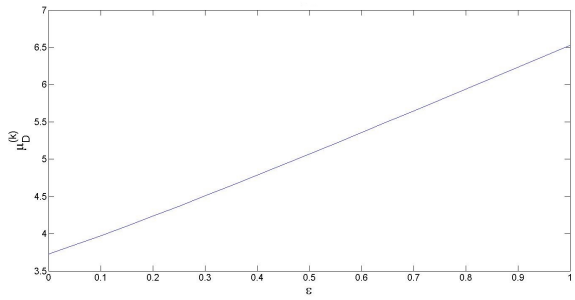


Figure 2.3: $\mu_D^{(k)}$ for aggregates in Figure 2.2.

The bound on $\mu_D^{(0)}$ based on the stencil (2.7) can be estimated using Gerschgorin's theorem. Each node in $A^{(0)}$ is next to the boundary. Hence it couples to at least three boundary nodes. Therefore the smallest eigenvalue of $A^{(0)}$ is at least

$$8 + 8\epsilon - \left[3 + 3\epsilon + \frac{5}{2} - \frac{1}{2}\epsilon + \left|\frac{1}{2} - \frac{5}{2}\epsilon\right|\right] = \begin{cases} 2 + 8\epsilon & \text{if } \epsilon \leq 1/5 \\ 3 + 3\epsilon & \text{if } \epsilon > 1/5 \end{cases} . \tag{2.21}$$

Since $D^{(0)} = 8(1+\epsilon)I$, the right-hand side of (2.10) is $8(1+\epsilon)$ over the smallest eigenvalue of $A^{(0)}$; hence

$$\mu_D^{(0)} \leq \begin{cases} (4 + 4\epsilon)/(1 + 4\epsilon) & \text{if } \epsilon \leq 1/5 \\ 8/3 & \text{if } \epsilon > 1/5 \end{cases} . \tag{2.22}$$

With (2.22) and computed $\mu_D^{(k)}$ the bound on $\rho(E_{TG})$ can be established using Theorems 1.1 and 1.2'. The result is plotted in Figure 2.4. For comparison, the actual values of

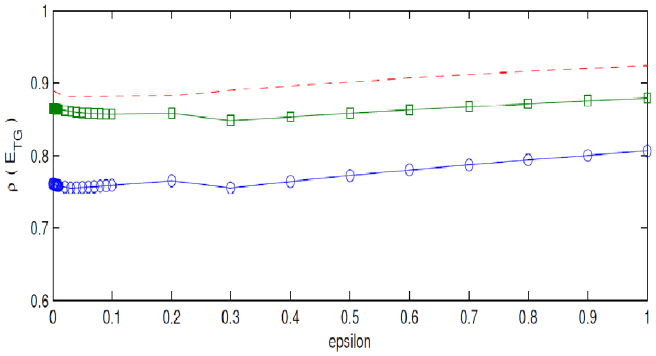


Figure 2.4: Bounds from Theorems 1.1 and 1.2' on $\rho(E_{TG})$ (dashed) and actual values of $\rho(E_{TG})$ on a 42 by 42 grid. Curve marked with circles is for pre- and post-smoothing; curve marked with squares is for pre-smoothing only or post-smoothing only.

$\rho(E_{TG})$ for a grid with 42 by 42 interior nodes, using one pre- and one post-smoothing step per iteration (curve marked with circles) is plotted. Another plotted curve (marked with squares) indicates the actual $\rho(E_{TG})$ for the two-grid method using only one pre- or one post-smoothing step.

In general the target coarsening ratio is 1/4 for a 2D problem ([24]). The coarsening ratio for the aggregates shown in Figure 2.2, unfortunately, is only 7/16. For the standard isotropic diffusion equation, box aggregates are standard and used in the coarsening process, as shown in Figure 2.5 where a tentative aggregate is split into four box-shaped aggregates. The coarsening ratio is 1/4. For this case $P^{(k)}$ (now 16×4) is different from that in (2.19) and the matrix $D^{(k)}(I - \pi_D^{(k)})$ in (2.11) is (ordering the nodes such that the lower left aggregate consists of nodes 1-4, the lower right aggregate consists of nodes 5-8, etc.)

$$8(1+\epsilon)\left[I - \begin{pmatrix} \frac{1}{4}e_4e_4^T & & & \\ & \frac{1}{4}e_4e_4^T & & \\ & & \frac{1}{4}e_4e_4^T & \\ & & & \frac{1}{4}e_4e_4^T \end{pmatrix}\right]. \tag{2.23}$$

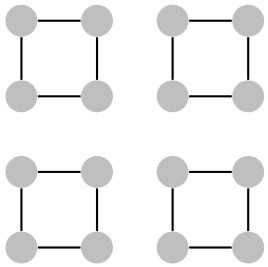


Figure 2.5: Box aggregate

Consider again the case when $\epsilon = 0$. In the box aggregate ordering, the vector $(0, 1, -1, 0, 2, 3, 1, 2, -2, -1, -3, -2, 0, 1, -1, 0)^T$ (which is $(0, 0, 0, 0, 1, 1, 1, -1, -1, -1, 2, 2, -2, -2, 3, -3)^T$ using a diagonal ordering of nodes) in the null space of $A^{(k)}$ is *not* in the null space of $D^{(k)}(I - \pi_D^{(k)})$. The product is a nonzero vector $2(1+\epsilon)\cdot(0, 4, -4, 0, 0, 4, -4, 0, 0, 4, -4, 0, 0, 4, -4, 0)^T$. It can be expected that for small ϵ the right hand side of (2.11) will be large; $\mu_D^{(k)}$ approaches ∞ as $\epsilon \to 0$. The large value of $\mu_D^{(k)}$ implies that the bound on $\rho(E_{TG})$ is close to 1.

In Figure 2.6 the spectral radius $\rho(E_{TG})$ on a 42 by 42 grid is computed using box aggregates. The curve marked with circles shows $\rho(E_{TG})$ using one pre- and one post-smoothing step per cycle while the curve marked with squares represents the method using only one pre-smoothing or one post-smoothing step per cycle. Although both curves show that when ϵ is small the spectral radius is close to 1, the upper bound in terms of $\mu_D^{(k)}$ appears to be a large overestimate for the actual value of μ_D. For $\epsilon = 0.001$, for example, the value of μ_D on the 42 by 42 grid was only 17.95 while the computed value of $\mu_D^{(k)}$ was 595.12. The spectral radius of the iteration matrix was 0.9655 for $\nu_1 = \nu_2 = 1$ and 0.9752 for $\nu_1 + \nu_2 = 1$, while the estimate based on $\mu_D^{(k)}$ would have been $\mu_X \leq (9.003/4.004) \cdot 595.12 = 1338.1$, $\rho(E_{TG}) \leq 1 - 1/1338.1 = 0.9993$. For larger size of grid the values of μ_D are computed and the result is depicted in Figure 2.7. μ_D increases with grid size and is asymptotically approaching a value significantly less than $\mu_D^{(k)} = 595.12$, but still large enough to indicate poor convergence of the two-grid method.

In general, when $D^{(k)}$ is a scalar multiple of the identity, or, more generally, a diagonal matrix with positive diagonal entries, the null space of $D^{(k)}(I - \pi_D^{(k)})$ consists of vectors

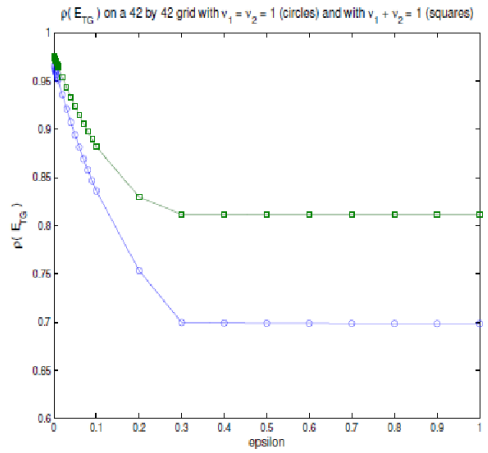


Figure 2.6: Actual values (circles and squares) for $\rho(E_{TG})$ with box aggregates on a 42 by 42 grid. Curve with circles is for pre- and post-smoothing, curve with squares is for pre-smoothing only or post-smoothing only.

that are constant on individual aggregates associated with $P^{(k)}$. When $\epsilon = 0$, the null space of $A^{(k)}$ contains a vector that is constant only on the diagonal aggregates. Thus for small $\epsilon > 0$, any aggregates other than those shown in Figure 2.2 will cause a large value of $\mu_D^{(k)}$ and prediction of poor convergence behavior of the two-grid method.

The automatic aggregation procedure described in [23] and implemented in [27] consists of two or three passes of pairwise aggregation procedures. Each pass tries various paired aggregates in order to minimize a quantity related to the aggregate quality. For the problem with $\theta = \pi/4$ and $\epsilon = 0.001$, in the first pass it forms pairwise aggregates aligned with the anisotropy, as pictured in Figure 2.8 (a). On the second pass, however, the algorithm grouped pairs to the north and south above the 45° line and pairs to the east and west below the 45° line, forming the parallelogram shaped aggregates shown in Figure 2.8 (b). This is because the strongest negative connections in the tentative matrix formed after the first pass of pairwise aggregation are aligned with north-south or east-west direction rather than the diagonal lines. In this case, the spectral radius of the iteration matrix with pre- and post-smoothing on a 42 by 42 grid was 0.9307, indicating rather slow convergence of

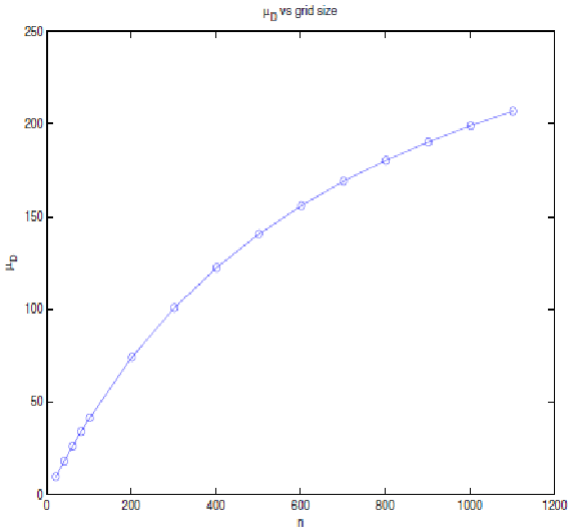


Figure 2.7: Computed values of μ_D vs grid size N for box aggregates, $\theta = \pi/4$, $\epsilon = 0.001$.

the two-grid method. This spectral radius increased to 0.9614 on an 82 by 82 grid, and while it will asymptote to something less than 1, the asymptotic value as $N \to \infty$ may be quite close to 1.

2.5 A New Procedure for Determining Appropriate Aggregates

The result for $\theta = \pi/4$ in the previous section suggests that the direction of (line) aggregates should align with the direction of anisotropy in order that the convergence of the two-grid method is grid size and ϵ independent. Perfect alignment may not possible when θ is other than $\pi/4$. But it is still advisable that the alignment of aggregates should be as close to the direction of anisotropy as possible. This optimal alignment is indicated in the eigensystem of $A^{(k)}$. After the splitting $A = A_b + A_r$ and tentative aggregates are formed, to determine appropriate actual aggregates one possible approach is to compute the eigensystem of $A^{(k)}$, look at the eigenvectors corresponding to small eigenvalues, and aggregate those points on which those eigenvectors are near constant. For example, when $\theta = \pi/6$ and $\epsilon = 0.001$, $A^{(k)}$ has, in addition to a constant eigenvector corresponding to a zero eigenvalue, an eigenvector

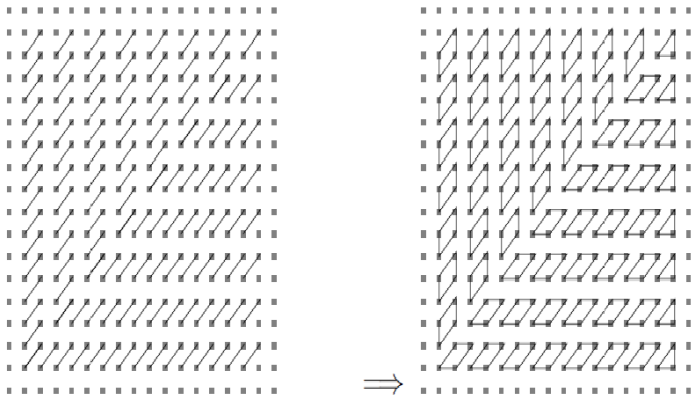


Figure 2.8: First pass and second pass of the automatic aggregation procedure in [22]

corresponding to a small eigenvalue, 0.0027, which has the following values at the grid points in the aggregates

$$\begin{matrix} 0.4586 & 0.2644 & 0.0706 & -0.1227 \\ 0.3463 & 0.1526 & -0.0409 & -0.2343 \\ 0.2343 & 0.0409 & -0.1526 & -0.3463 \\ 0.1227 & -0.0706 & -0.2644 & -0.4586 \end{matrix} . \tag{2.24}$$

In (2.24) the pairs of nearly equal components are (using the natural ordering of the nodes): 1 and 10 (0.1227 and 0.1526), 2 and 11 (-0.0706 and -0.0409), 3 and 12 (-0.2644,-0.2343), 5 and 14 (0.2343 and 0.2644), 6 and 15 (0.0409 and0.0706), 7 and 16 (-0.1526 and -0.1227). Other pairs such as 9 and 13 (0.3463 and 0.4586) or 4 and 8 (-0.3463 and -0.4586) are fairly close, so these might be considered for aggregation as well. The resulting aggregates are shown in Figure 2.9.

Since the eigenvector of $A^{(k)}$ associated with the small eigenvalue is not extremely close to the null space of $D^{(k)}(I - \pi_D^{(k)})$, the computed $\mu_D^{(k)}$ using these aggregates is large: 50. The actual value of μ_D, however, was much smaller for a 42 by 42 grid: $\mu_D = 5.08$. This resulted in a reasonable size spectral radius of the two-grid iteration matrix for small ϵ, as pictured in Figure 2.10.

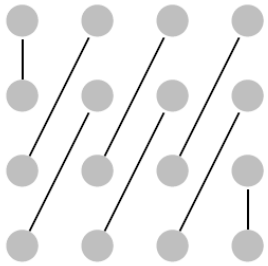


Figure 2.9: Aggregates based on nearly equal components of eigenvector of $A^{(k)}$ corresponding to eigenvalue 0.0027 when $\theta = \pi/6$, $\epsilon = 0.001$

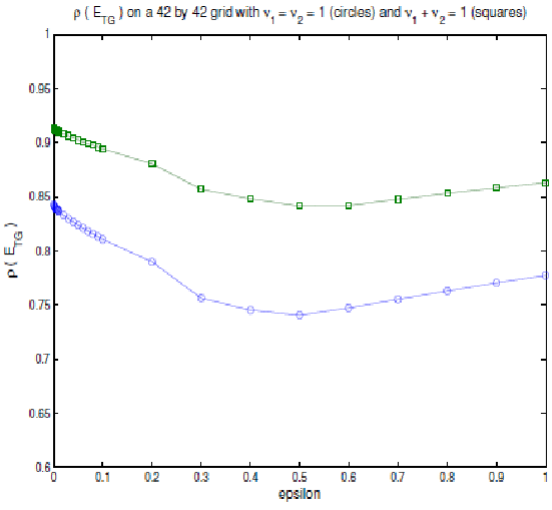


Figure 2.10: Actual values for $\rho(E_{TG})$ for aggregates corresponding to $\theta = \pi/6$ on a 42 by 42 grid, using pre- and post-smoothing (circles) and using only pre-smoothing or only post-smoothing (squares).

Clearly, the eigenvectors corresponding to small eigenvalues of $A^{(k)}$ indicate the direction of anisotropy. Using these aggregates does not necessarily lead to a small value of $\mu_D^{(k)}$, but it does seem to result in a smaller value of μ_D. Figure 2.11 shows values of μ_D for larger grid sizes, when $\epsilon = 0.001$. Again μ_D grows with the grid size, but it asymptotically approaches a value which appears to be less than about 20, even though $\mu_D^{(k)} = 50$.

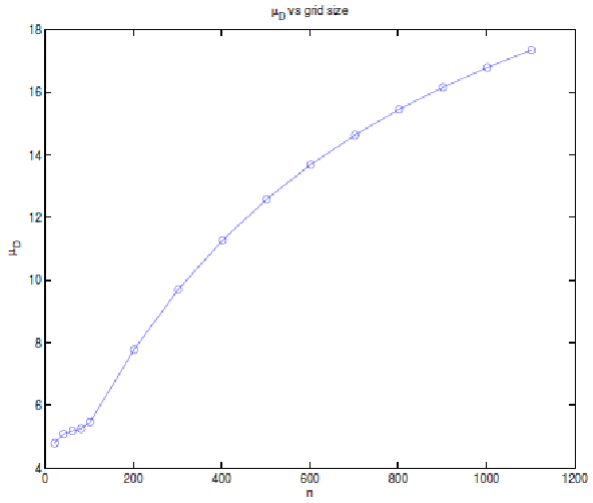


Figure 2.11: μ_D vs grid size n for aggregates corresponding to $\theta = \pi/6$; $\epsilon = 0.001$.

The coarsening ratio of the splitting in Figure 2.9 is only 1/2. One can obtain a better coarsening ratio by choosing appropriate aggregates from larger size tentative aggregates. For example, on an 8 by 8 block the size of appropriate aggregates that closely align with $\theta = \pi/6$ line can be 4, and the overall coarsening ratio is close to 1/4. The cost is the computation of the eigenvalues/vectors of larger blocks.

The aggregation procedure discussed above depends on the eigenvectors associated with small eigenvalues of $A^{(k)}$. The final aggregates are chosen based on "nearly equal" components of eigenvectors associated with "small" eigenvalues. The automatic aggregation algorithm based on this principle strongly depends on how "small" and "nearly equal" are defined. The results may differ significantly between different definitions. Another approach for automating the choice of aggregates is to try various aggregates from among possible

groups of nodes in a tentative aggregate, then choose the one which gives the best value for $\mu_D^{(k)}$. As mentioned previously, using such aggregates may not result in a small value of $\mu_D^{(k)}$ even when μ_D is small. But it is still reasonable to expect that the aggregates that make $\mu_D^{(k)}$ less large will be more likely to yield a good value for μ_D. We consider 8 possible candidates from a 4×4 tentative aggregate:

1. Vertical aggregates: (1,5,9,13), (2,6,10,14), (3,7,11,15), (4,8,12,16) (where numbers in parentheses represent nodes that were aggregated, using the natural ordering of nodes in a potential aggregate);

2. Near vertical aggregates: (1,14), (2,15), (3,16), (4,8,12), (5,9,13), (6,10), (7,11).

3. 30° (with the vertical) aggregates: (1,10), (2,11), (3,12), (4,8), (5,14), (6,15), (7,16), (9,13);

4. 45° aggregates: (1,6,11,16), (2,7,12), (5,10,15), (3,8), (9,14), (4), (13);

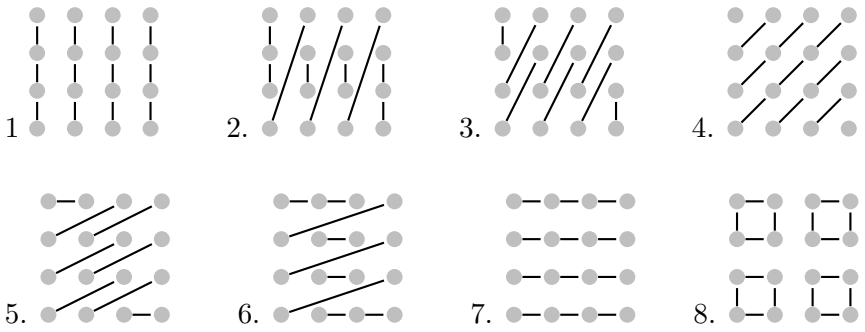
5. 60° (with the vertical) aggregates: (1,7), (2,8), (3,4), (5,11), (6,12), (9,15), (10,16), (13,14);

6. Near horizontal aggregates: (1,8), (2,3,4), (5,12), (6,7), (9,16), (10,11), (13,14,15);

7. Horizontal aggregates: (1,2,3,4), (5,6,7,8), (9,10,11,12), (13,14,15,16);

8. Box aggregates: (1,2,5,6), (3,4,7,8), (9,10,13,14), (11,12,15,16);

The aggregates are pictured in the following figures :



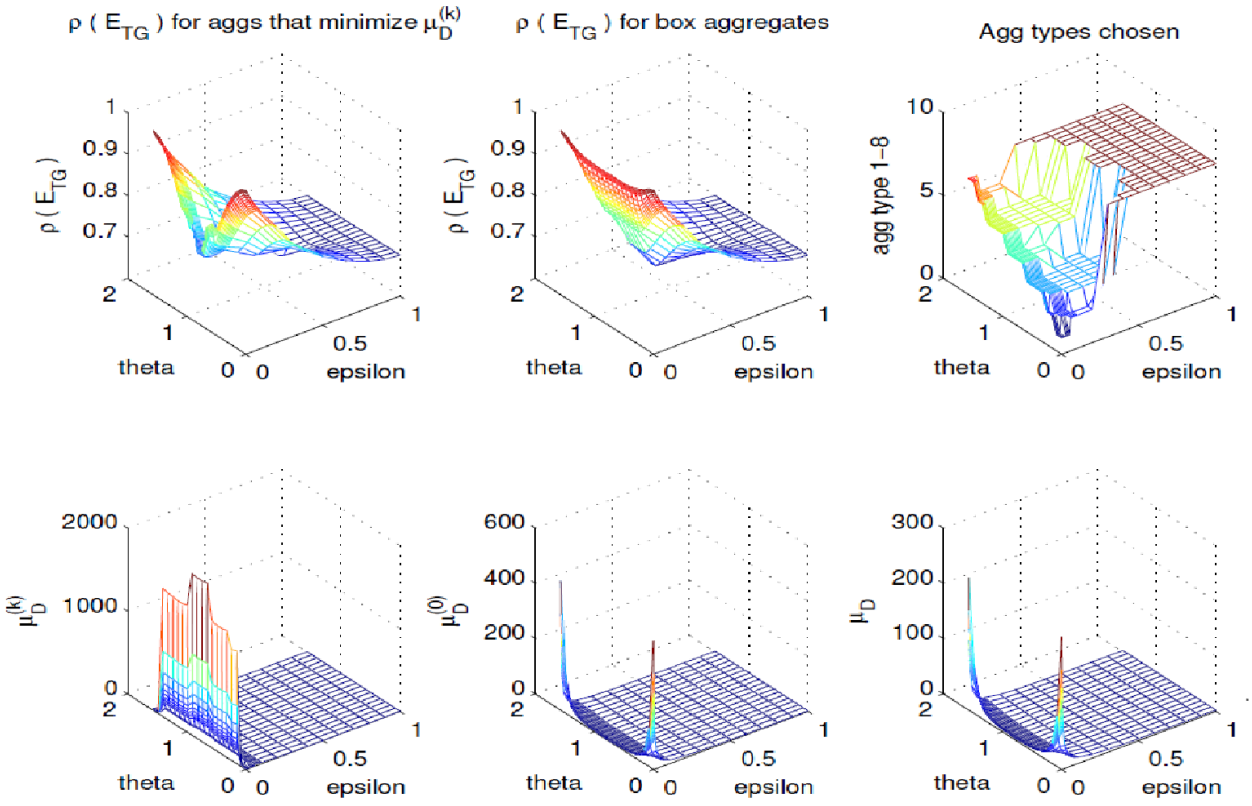


Figure 2.12: Upper left: $\rho(E_{TG})$ on a 42 by 42 grid using pre- and post-smoothing and aggregates that minimize $\mu_D^{(k)}$. Upper middle: $\rho(E_{TG})$ using box aggregates only. Upper right: Aggregate types (1-8) that were used for upper left results. Lower left: $\mu_D^{(k)}$ is still very large for ϵ near 0. Lower middle: $\mu_D^{(0)}$. Lower right: μ_D is much more like $\mu_D^{(0)}$ than $\mu_D^{(k)}$ for ϵ near 0.

The automatic approach based on choosing aggregates that minimize $\mu_D^{(k)}$ is tested on a 42 by 42 grid with different values of θ and ϵ. The spectral radius of the iteration matrix using pre- and post-smoothing is plotted in the upper-left block of Figure 2.12. For comparison, the spectral radius of the two-grid iteration matrix on the same 42 by 42 grid using pre- and post-smoothing, with box aggregates only, is depicted in the upper-middle block of Figure 2.12. When ϵ is small and θ is away from 0 or $\pi/2$, the spectral radius is considerably worse when only box aggregates are used. The upper-right block of Figure 2.12 exhibits the aggregate types that were chosen for various pairs of (ϵ, θ). For $\epsilon > 0.5$, the anisotropy is mild and the box aggregates were always chosen. For problems with stronger anisotropy, the aggregates chosen tend to align with the direction of anisotropy.

The lower part of Figure 2.12 exhibits the values of $\mu_D^{(k)}$ (lower left), $\mu_D^{(0)}$ (lower middle) and μ_D (lower right) for different pairs of (ϵ, θ). As mentioned previously, $\mu_D^{(k)}$ is still large for small values of ϵ even when the appropriate aggregates are chosen. The value of μ_D is much more like the value of $\mu_D^{(0)}$. When ϵ is small, the values of μ_D appear to be large when θ is near 0 or $\pi/2$ and become smaller otherwise. In this case, for θ near 0 or $\pi/2$, the matrix $A^{(0)}$ is not diagonally dominant, and one might expect to do better by including the points next to the boundary among the tentative aggregates. To investigate this, the same set of experiments are carried out on a 44 by 44 grid, with all the interior nodes included among tentative aggregates. The results are shown in Figure 2.13. The computed values of $\rho(E_{TG})$ without these points ranged from 0.70 to 0.996, while in the case of including these points, the lowest value of spectral radius decreased from about 0.7 to about 0.53 and the largest spectral radius (corresponding to $\epsilon = 0.001$, $\theta = \pi/19$ or $\theta = 17\pi/38$) was 0.936 instead of 0.996.

The new procedure for determining appropriate aggregates improves the convergence of the two-grid method. An interesting issue is to see if such improvement can be carried over in K-cycle multigrid. We have tested this new procedure in the code implemented in [23]. The new procedure is applied on the finest grid and the original automatic aggregation procedure is applied on the coarser levels. The timing results from the new procedure and the original aggregation algorithm are compared. We tested the cases when $\epsilon = 0.001$ for

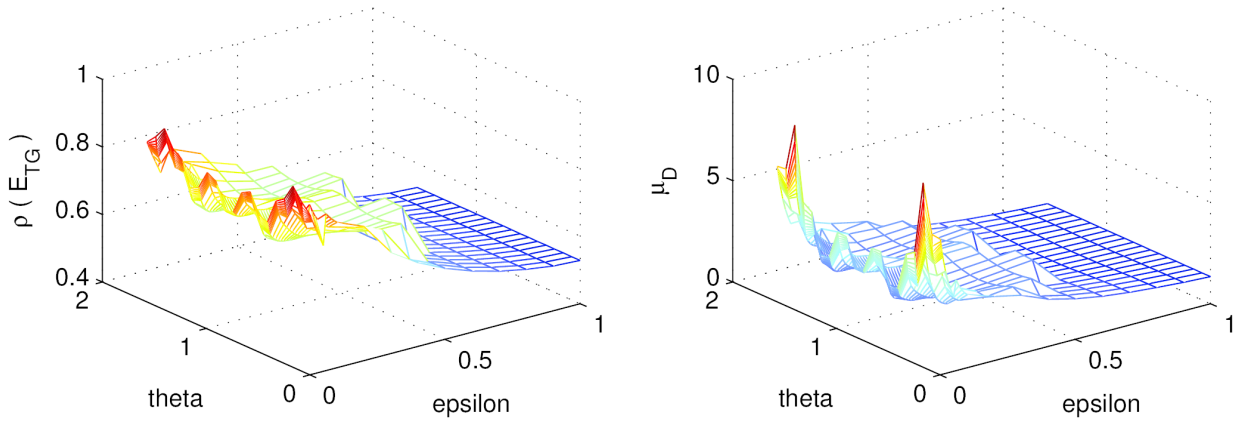


Figure 2.13: Left: $\rho(E_{TG})$ on a 44 by 44 grid (with all nodes included in the potential aggregates) using pre- and post-smoothing and aggregates that minimize $\mu_D^{(k)}$ (shown in Figure 2.12). Right: μ_D still increases for ϵ small, especially near $\theta = 0$ and $\theta = \pi/2$, but is much smaller than in Figure 2.12.

angles $\theta = \pi/12$, $\pi/6$ and $\pi/4$. The results are shown in Tables 2.1 - 2.3.

As seen from these tables, the results verify that if aggregates that minimize $\mu_D^{(k)}$ are chosen, that is, the aggregates are aligned with the direction of anisotropy (exactly or closely), the convergence of K-cycle multigrid can be improved. Although there is a certain degree of grid size dependency, the number of multigrid cycles is reduced and appears to have an asymptotic limit in all cases. The improvement is dramatic when $\theta = \pi/12$.

2.6 Conclusion

We have used the analysis in [22] to show that when aggregates aligned exactly with the diagonal direction are used in a two-grid method for solving a rotated anisotropic diffusion equation with $\theta = \pi/4$, the bound on the convergence rate is grid size independent and also is independent of ϵ, the strength of anisotropy. See Figures 2.3 and 2.4. When aggregates cannot be aligned *exactly* with direction of anisotropy, we cannot expect ϵ-independent convergence rates, but for fixed $\epsilon > 0$, Theorem 1.2 establishes a *bound* on the convergence rate that is independent of the grid size. From Figure 2.11 it appears that if this bound

Table 2.1: Timing result; $\epsilon = 0.001$, $\theta = \pi/4$

n	# level	Time(iteration) New procedure	Time (iteration) AGMG code
502^2	5	1.65 (60)	1.87 (66)
1002^2	5	8.17 (72)	9.48 (80)
1502^2	6	20.5 (78)	23.9 (88)
2002^2	7	40 (82)	45.7 (93)
2502^2	7	69.5(85)	75.4 (96)
3002^2	7	92.4 (87)	111 (98)

is ever approached, it must be for an extremely fine grid. We proposed a strategy for choosing appropriate aggregates based on selecting the ones that give the smallest $\mu_D^{(k)}$ from among possible candidates. The procedure needs to solve a number of generalized eigenvalue problems associated with the tentative aggregate. This can be difficult if the null space of the block $A^{(k)}$ is not known, but for our problems we were able to determine this analytically. We also observed that even when $\mu_D^{(k)}$ cannot be made small with this choice, it generally leads to small values of μ_D, one of the factors determining the bound on the convergence of the two-grid method. The proposed strategy was tested in the code implementation in [23] and gave some improvement in the number of K-cycles required in a true multigrid method.

Table 2.2: Timing result; $\epsilon = 0.001$, $\theta = \pi/6$

n	# level	Time(iteration) New procedure	Time (iteration) AGMG code
502^2	5	2.93 (97)	3.53 (124)
1002^2	6	15.1 (121)	18.7 (155)
1502^2	6	40.7 (133)	48.1 (171)
2002^2	7	70.7 (139)	92.5 (182)
2502^2	7	135(144)	153 (189)
3002^2	7	201 (148)	262 (196)

Table 2.3: Timing result; $\epsilon = 0.001$, $\theta = \pi/12$

n	# level	Time(iteration) New procedure	Time (iteration) AGMG code
502^2	5	1.78 (65)	3.07 (103)
1002^2	5	8.53 (76)	15.7 (125)
1502^2	6	20.7 (81)	39.7 (137)
2002^2	7	38.5 (85)	77.4 (146)
2502^2	7	65.4(87)	127 (150)
3002^2	7	99.7 (89)	191 (155)

Chapter 3

PARALLELIZATION OF AN AGGREGATION-BASED MULTIGRID CODE

The need for high-resolution PDE simulation motivated the parallelization of the multigrid algorithm. As mentioned in Chapter 1, multigrid algorithms involve two phases of computation. First the grid hierarchy is determined in the setup phase, then V-, W- or K-cycles are iterated in the solution phase. In solving a linear system arising from discretization of partial differential equations, parallelism in the setup phase resides in the geometric information: the domain of interest is divided into several subdomains, and each subdomain is assigned to a processor. Each processor is then responsible for updating the unknowns associated with that subdomain only. Such parallelization principles can be extended to algebraic multigrid: The rows of a matrix are divided into several groups and then each group is assigned to a processor. The processor is responsible for updating variables corresponding to its rows. The assignment of rows to groups is designed to minimize the interprocessor communication during the solution phase.

In this chapter, the parallelization issues of the aggregation-based multigrid algorithm in [22] are discussed. These issues include coarsening, smoothing and the coarsest grid solving. Numerical experiments, as will be shown in the following sections, exhibit that the coarsest grid solving step can be a bottleneck for parallel efficiency, while the previous two issues appear to be less difficult to deal with. To explore the most efficient way to solve the coarsest grid equations in parallel multigrid, we compare different solvers. The solvers compared are MUMPS, a direct solver, and the preconditioned conjugate gradient method, where the preconditioner is the sparse approximate inverse of the coarsest grid matrix.

3.1 Parallelization Issues

In general, multigrid methods store sparse matrices in compressed sparse row (CSR) format, where one vector stores the matrix entries, one vector stores the column indices and one

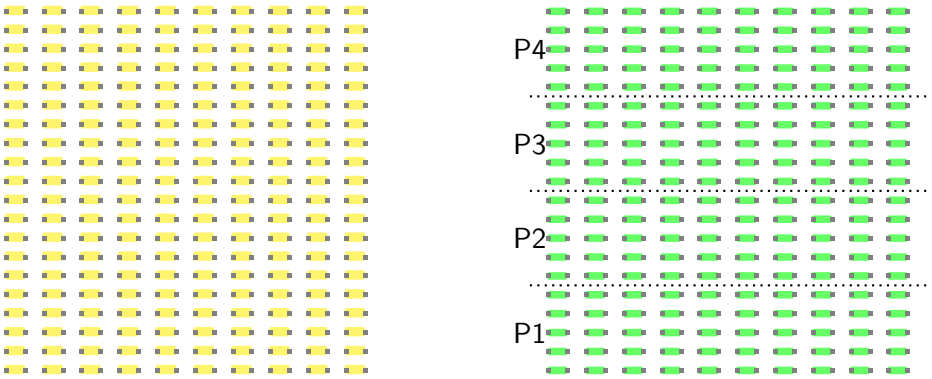
vector stores the number of nonzero entries in each row. In parallel multigrid rows of the matrix that one wants to solve are distributed or initialized in the each processor. In this manner the local matrix in a processor is a rectangular matrix with as many rows as local variables, but more columns corresponding to non-local matrix entries. One also needs information to indicate the neighbors of each processor and vectors that indicate connections between variables of processors at each level. These vectors are needed for matrix-vector multiplications in solution phase.

3.1.1 Parallel Coarsening

The coarsening algorithm accesses the matrix only row by row. It treats the local matrix as if the non-local variables have been marked and excluded from the aggregation process at the start of the coarsening. Hence each task computes locally the coarsening scheme for local variables. This simple approach for parallel coarsening may create different aggregates from that of serial coarsening. For example, consider an N by N matrix A of finite difference discretization of a 2D Model problem with 5-point stencil. Figure 3.1a and 3.1b show the aggregates after the first pass of the coarsening algorithm of the code AGMG [27]. While the formed aggregates are the same after the first pass, they differ after the second pass, as can be seen in Figure 3.2a and 3.2b. There are aggregates with " **bad quality** " which are forced to be formed near the boundaries (internal) between the subdomains (processors), such as line aggregates which are not appropriate when there is no anisotropy. These bad aggregates may affect the convergence.

3.1.2 Parallel Smoothing

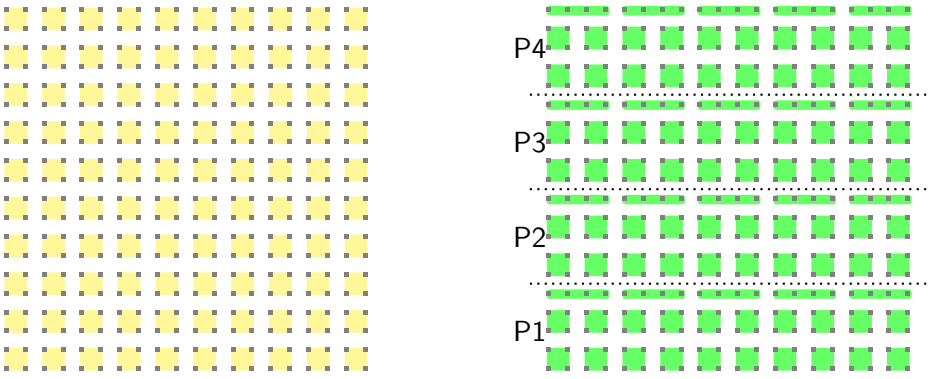
The solution phase involves the following steps: smoothing, computing residuals and restriction/prolongation. In general, smoothing steps involve inversion of a matrix albeit a simple one. For weighted Jacobi relaxation, the inversion of diagonal entries can be carried out independently for each row and is easily parallelizable. For Gauss-Seidel relaxation, however, one uses backward or forward substitution, which are well known to be inherently sequential, to solve the matrix equation where the smoothing matrix $G^{(l)}$ is the upper or



(a) Serial aggregations;first pass

(b) Parallel aggregations; first pass

Figure 3.1: First pass of pairwise aggregation



(a) Serial aggregations; second pass

(b) Parallel aggregations; second pass

Figure 3.2: Second pass of pairwise aggregation

lower triangular part of operator $A^{(l)}$ at level l. To parallelize Gauss-Seidel smoothing the processors ignore the matrix entries which are connected to the neighboring processors, and all processors solve their part of the residual equation simultaneously, without having to wait for the information from the other processors. The smoother matrix is thus block diagonal:

$$\begin{bmatrix} \mathbf{B}_1 & 0 & 0 & 0 \\ 0 & \mathbf{B}_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \mathbf{B}_p \end{bmatrix}$$

where $\mathbf{B}_i$ is the lower or upper triangular part of the local matrix A_i ($\lceil \frac{n}{p} \rceil \times n$), $i = 1, ..., p$. The convergence may suffer from this modification of the smoothers. In order to see how the parallelization of aggregation-based multigrid affects the convergence, we perform numerical experiments using the code implementation in [27] for the 2D Model problem with Dirichlet boundary condition on a 8000×8000 grid. The numbers of V-cycle preconditioned conjugate gradient iterations for different numbers of processors are shown in Table 3.1. As seen from

Table 3.1: Effect of parallelization on the convergence of aggregation-based multigrid, V-cycle preconditioned CG

No. of processors	# level	Time setup + solution	No. of CG iterations
1	8	53.2+354	56
4	8	13.9+93.1	76
8	8	7.45+58.2	77
16	8	4.16+29.7	78
25	8	2.87+20.9	107
64	8	2.04+8.9	110

the table, the number of CG cycles increases significantly, and the parallel aggregation and

smoothing degrades the parallel efficiency. However, if the K-cycle is used, the number of CG cycles of the code in [27] doesn't change significantly. Furthermore, for all the tested problems in this chapter (which will be introduced later), neither parallel coarsening nor parallel smoothing affect the convergence of K-cycle multigrid significantly, as can been seen from Figure 3.3. The difference of number of cycles between serial and parallel multigrid is at most 3 on each of the problems shown.

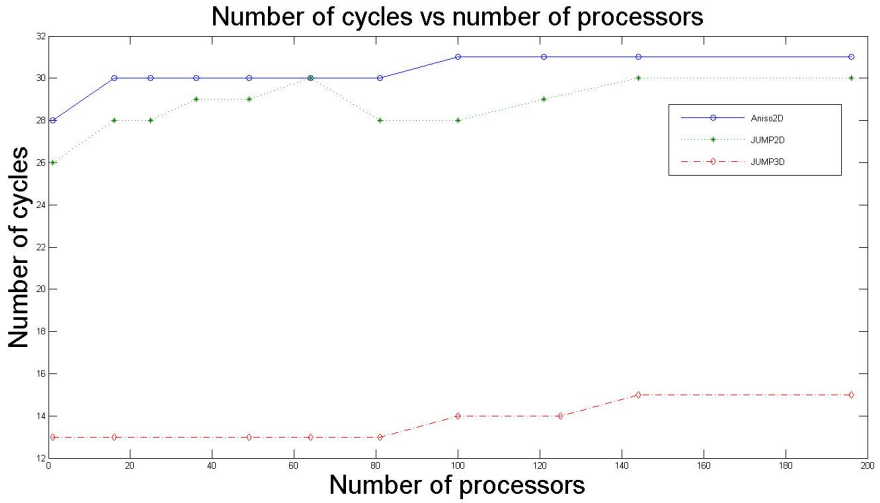


Figure 3.3

3.1.3 Solving the Coarsest Grid Problem

Solving the coarsest grid problem in the multigrid cycle may be critical to the performance of the parallel multigrid method, as mentioned in [9]. The size of the portions of the operator stored in each processor at this level is generally small, and the time required for communication may be higher than the time required to perform the solve on a single processor. Furthermore, the coarsest grid operator may couple all pieces of the global problem (i.e., it is dense, or nearly dense), and thus global communication of the right-hand side or other data may be necessary. This is generally what happens in classical algebraic multigrid, where the linear prolongation operator and Galerkin formula increase the average

number of entries per row at each coarser level. For aggregation-based algebraic multigrid, however, the average number of entries per row in most cases does not increase significantly since the prolongation operator is simple [24]. The major problem for the coarsest grid solve in aggregation-based algebraic multigrid is the number of visits on the coarsest grid, because for the cases studied and parameters used we observed that at each level the K-cycle has to visit the coarser level twice to get good convergence. Therefore the number of visits to the coarsest grid is 2^{l-2}, where l represents the number of levels. Hence the performance of the coarsest grid solve is crucial to the parallel efficiency of AGMG.

The coarsest grid solver may be a direct solver or an iterative solver. In this chapter we compare the two approaches based on the parallel efficiency of an aggregation-based algebraic multigrid code.

3.1.4 Multifrontal Massively Parallel Sparse Direct Solver (MUMPS)

When the coarsest grid problem is small enough, a direct solver is a natural choice. In the setup phase the LL^T factorization (assuming symmetric positive definite matrix) of the coarsest grid matrix can be computed in the root processor. The LL^T factorization, stored in the root processor, is used when the multigrid cycle reaches the coarsest level, where all processors send their portions of the right-hand side vector to the root node and the solution is computed. The computed solution is then distributed back to the processors. However, when the linear system gets larger, the coarsest grid matrix may become too large to fit in one processor.

The AGMG code in [27] uses the parallel direct solver MUMPS to solve the coarsest grid problem. When the coarsest grid is reached the root processor gathers portions of the right hand side vector from all processors and the solution is scattered back after solving. This step is inherently sequential and the amount of communication is proportional to the total number of coarsest grid unknowns and to the number of processors that must send/receive this information. This tends to degrade parallel efficiency.

3.1.5 Preconditioned Conjugate Gradient (PCG) and SAI preconditioner

Iterative methods for the coarsest grid solve are less sequential, requiring matrix-vector products for each solve. However, if the matrices are quite dense, it is important that very few iterations are required, or the accumulation of communication costs can become very high. To this end, preconditioning may be used, especially since the cost of constructing the preconditioner will be amortized. Similarly, it is advantageous to exploit previous solves with the same matrix, e.g., Krylov subspace vectors from previous solves may be used as an initial approximation space.

When a preconditioner M is applied in the conjugate gradient algorithm, one needs to solve the equation $Mz_k = r_k$ in each CG iteration, where r_k represents the residual at iteration k. Either we solve the equation directly, for example by backward or forward substitution if M is triangular, or if M^{-1} is available we can do matrix-vector multiplication. In most cases the inverse of M is dense. Many popular general-purpose preconditioners, such as those based on incomplete factorizations of A, are fairly robust and result in good convergence rates, but are highly sequential and it is difficult to implement them efficiently on parallel computers. This motivates the development of another type of preconditioner: sparse approximate inverses [5]. Using sparse approximate inverses turns the preconditioning step in CG iterations to matrix-vector multiplication, $z_k = M^{-1}r_k$, which can be parallelized efficiently with interprocessor communication of matrix entries and vector entries connecting to neighboring processors.

For symmetric positive definite problems, A^{-1} is approximated by the factorization G^TG, where G is a sparse lower triangular matrix approximating the inverse of the lower triangular Cholesky factor, L, of A [10]. Thus the sparse approximate inverse is computed as the matrix G^TG so that G minimizes $\|I - GL\|_F$, where $\|\cdot\|_F$ denotes the Frobenius norm, subject to some sparsity constraint. Minimizing $\|I - GL\|_F$ can be accomplished without knowing L by solving the normal equations :

$$(GLL^T)_{ij} = (L^T)_{ij}, \qquad (i,j) \in S_L, \tag{3.1}$$

where S_L is a lower-triangular nonzero pattern for G. Equation (3.1) can be replaced by

$$(\hat{G}A)_{ij} = I_{ij}, \qquad (i,j) \in S_L, \tag{3.2}$$

where $\hat{G} = D^{-1}G$ and D is the diagonal of L.

The sparsity pattern can be dynamically determined according to some criterion during the setup phase, or predetermined by a fixed sparsity pattern, for example, the sparsity pattern of the lower triangle of the original matrix A, or A^2. For simplicity, we use the sparsity pattern of the lower triangle of A to construct the approximate inverse. By using the Frobenius norm, the constrained minimization problem decouples into n independent linear least squares problems (one for each row of G). The number of unknowns for each problem is equal to the number of nonzeros allowed in each row of G. This immediately follows from the identity:

$$\|I - GL\|_F^2 = \sum_{i=1}^{n} \|I(i,:) - G(i,:)L\|_2^2. \tag{3.3}$$

These linear least squares problems can be solved independently for each row $G(i,:)$ directly or iteratively. This feature is suitable for parallel processing. Processors communicate entries needed for neighbors and then solve the least squares equations independently.

On the other hand, it is possible that the coarsest grid solution obtained from a few preconditioned CG iterations may not be accurate enough for some difficult problems and might cause an increase in the number of multigrid iterations. The balance between these factors to obtain an optimal algorithm is an interesting issue. Rather than using a fixed number of preconditioned CG iterations, we use a tolerance on the relative residual (2-norm of the residual at step k divided by the 2-norm of the initial residual) as the stopping criterion for CG iterations.

3.2 Description of Tested Linear Systems

The following test problems are considered. In all cases we use a uniform mesh with constant mesh size h in all directions of the domain.

Problem ANI2D : linear system resulting from the five-point finite difference discretization of PDE:

$$-\frac{\partial^2 u}{\partial x^2} - b\frac{\partial^2 u}{\partial y^2} = 1 \text{ in } \Omega = (0,1) \times (0,1) \tag{3.4}$$

with coefficient $b = 0.001$ and Dirichlet boundary condition:

$$u = 0, \text{ on } \partial\Omega. \tag{3.5}$$

Problem JUMP2D : linear system resulting from the five-point finite difference discretization of PDE:

$$-\frac{\partial}{\partial x}(a\frac{\partial u}{\partial x}) - \frac{\partial}{\partial y}(b\frac{\partial u}{\partial y}) = f \text{ in } \Omega = (0,1) \times (0,1) \tag{3.6}$$

with coefficients given by

$$\begin{cases} a = 1, & b = 100, & f = 0, & \text{in } (0.65, 0.95) \times (0.05, 0.65), \\ a = 100, & b = 1, & f = 0, & \text{in } (0.25, 0.45) \times (0.25, 0.45), \\ a = 100, & b = 100, & f = 1, & \text{in } (0.05, 0.25) \times (0.65, 0.95), \\ a = 1, & b = 1, & f = 0, & \text{in elsewhere}, \end{cases} \tag{3.7}$$

and Dirichlet boundary condition:

$$u = 0, \text{ on } \partial\Omega. \tag{3.8}$$

Problem JUMP3D : linear system resulting from the five-point finite difference discretization of PDE:

$$-\frac{\partial}{\partial x}(a\frac{\partial u}{\partial x}) - \frac{\partial}{\partial y}(b\frac{\partial u}{\partial y}) - \frac{\partial}{\partial z}(c\frac{\partial u}{\partial z}) = f \text{ in } \Omega = (0,1) \times (0,1) \times (0,1) \tag{3.9}$$

with coefficients given by

$$\begin{cases} a = b = c = d & f = 1, & \text{in } (0.25, 0.75) \times (0.25, 0.75) \times (0.25, 0.75) \\ a = b = c = 1, & f = 0, & \text{elsewhere} \end{cases} \tag{3.10}$$

where d is a parameter, and Dirichlet boundary condition:

$$u = 0, \text{ on } \partial\Omega. \tag{3.11}$$

3.3 Description of the TACC Lonestar Parallel Processor

All numerical experiments were carried out on TACC Lonestar Linux Cluster consisting of 1,888 compute nodes, with two 6-Core processors per node, for a total of 22,656 cores. It is configured with 44 TB of total memory and 276TB of local disk space. The theoretical peak compute performance is 302 TFLOPS. Each node houses two 6-core Intel Hexa-Core 64-bit Westmere processors (12 cores in all). The core frequency is 3.33GHz and supports 4 floating-point operations per clock period with a peak performance of 13.3 GFLOPS/core or 160 GFLOPS/node. Each node contains 24GB of memory (2GB/core). The memory subsystem has 3 channels from each processor's memory controller to 3 DDR3 ECC DIMMS, running at 1333 MHz. The processor interconnect, QPI, runs at 6.4 GT/s.

3.4 Parameters

In all experiments, the coarsening step will stop as soon as the global number of unknowns is below $400N_p$, where N_p is the number of concurrent tasks. The PCG iteration will stop when the 2-norm of the relative residual is ≤ 0.90. This very loose tolerance avoids large number of PCG iterations of the coarsest grid solve, but may cause extra K-cycle iterations because of the inaccurate solution. It is stated in [30] that it is often not necessary to solve the coarsest grid problems exactly. For each visit of the coarsest grid, one may expect that a defect reduction of $\rho^{1/\kappa}$ will be sufficient, where ρ denotes the expected multigrid convergence factor and κ is the number of coarsest grid visits [18]. Each timing result is the average of multiple experiments. When using multiple cores per computing node, the preliminary experiments indicate a lack of efficiency due to memory conflicts. Hence we carried out experiments using one core per node. The number of unknowns per computing node was kept approximately constant. For 2D problems the number of variables per node is 9×10^6, while for 3D problems each node contains approximately 27×10^6 variables. Fine grid variables were initially divided between processors by the AGMG code. After this initial division, synchronization between processors is required, but there is no exchange of data until the coarsest grid is reached.

3.5 Timing Results for Tested Problems

1. Problem ANI2D : The experimental results for problem ANI2D are shown in Table 3.2 and Figure 3.4. The parallel efficiency for using SAI preconditioned CG as coarsest grid solver is better than that of using MUMPS. We observed the benefit of replacing MUMPS with SAI preconditioned CG. This difference is especially large when the number of processes is more than 100. Note that for all the tested problems we evaluate the parallel efficiency of each coarsest grid solver approach by comparing the parallel execution time against the sequential time of the original code in [27]. All the timing results are measured in seconds.

Table 3.2: Timing results: ANI2D

Numbers in parenthesis are numbers of outer iterations

n $\times 10^6$	#p	# level	Time (setup+solve) (# it) MUMPS	Time (setup+solve) (# it) SAI CG
9	1	7	6.58 + 28.6 (28)	6.25 + 30.6(30)
144	16	8	8.3 + 37.2 (30)	7.54 + 30.8 (30)
225	25	8	8.19 + 40.6 (30)	8.1 + 33.9 (31)
324	36	9	8.53 + 41.2 (30)	7.25 + 31 (30)
441	49	9	9.31 + 40.7(30)	7.18 + 32.4 (31)
576	64	9	9.55 + 40.8 (30)	7.08 + 31.2 (31)
900	100	9	13.7 + 46.8 (31)	6.53 + 33 (31)
1089	121	10	11.2 + 170.8 (31)	6.53 + 33 (31)
1296	144	10	12.9 + 183 (31)	6.38 + 35.7 (31)
1764	196	10	16.9 + 187 (30)	6.92 + 42.1 (30)
2025	225	10	> 360	6.43 + 43 (31)
2304	256	10	> 360	6.44 + 44.2 (31)

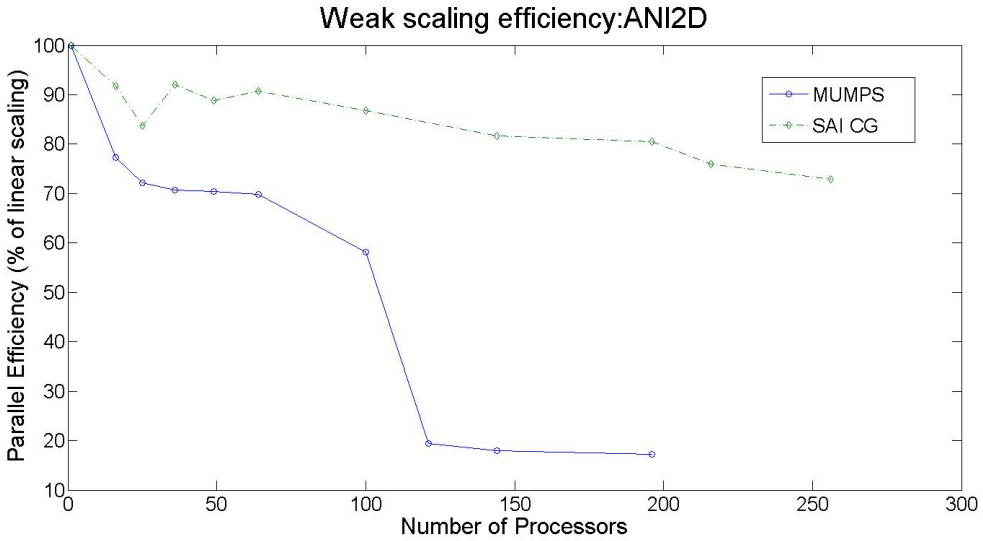


Figure 3.4: Parallel efficiencies for using MUMPS and using SAI CG as coarsest grid solver. ANI2D

2. Problem JUMP2D :

Figure 3.6 shows the coefficients of the test problem JUMP2D on the $(0,1) \times (0,1)$ domain. Table 3.3 and Figure 3.5 show the timing results of the experiment comparing MUMPS and SAI preconditioned CG. For small to medium number of processors (≤ 25), MUMPS provides a better speedup. To avoid large number of SAI PCG iterations in the coarsest grid solve, the stopping criterion for SAI PCG was set to be very low. The solution of the coarsest grid problem is not very accurate and hence causes more K-cycle iterations than in the case using MUMPS. The number of K-cycles in the case using SAI PCG coarsest grid solving is 1 or 2 more than that of using MUMPS coarsest grid solver. The performance of SAI PCG coarsest grid solver is better when the number of processors is large (≥ 64). In these cases, the sequential nature of the solution stage in MUMPS reduced the speedup of the multigrid significantly. As mentioned in the previous section, the number of visits to the coarser grid at each level is 2, thus the number of visits to the coarsest grid is 2^{l-1}. The setup phase time is not affected and it seems that MUMPS is more costly than constructing a sparse approximate inverse.

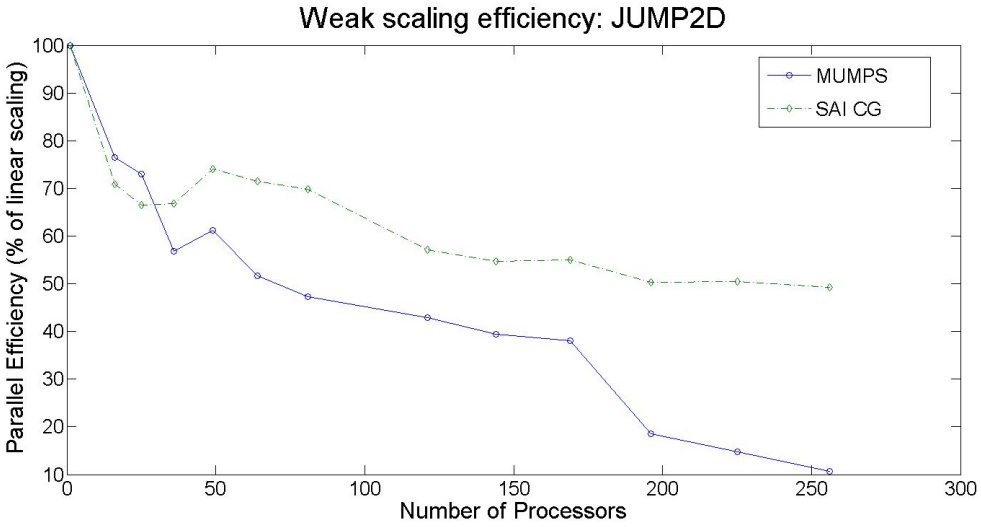


Figure 3.5: Parallel efficiencies for using MUMPS and using SAI CG as coarsest grid solver. JUMP2D

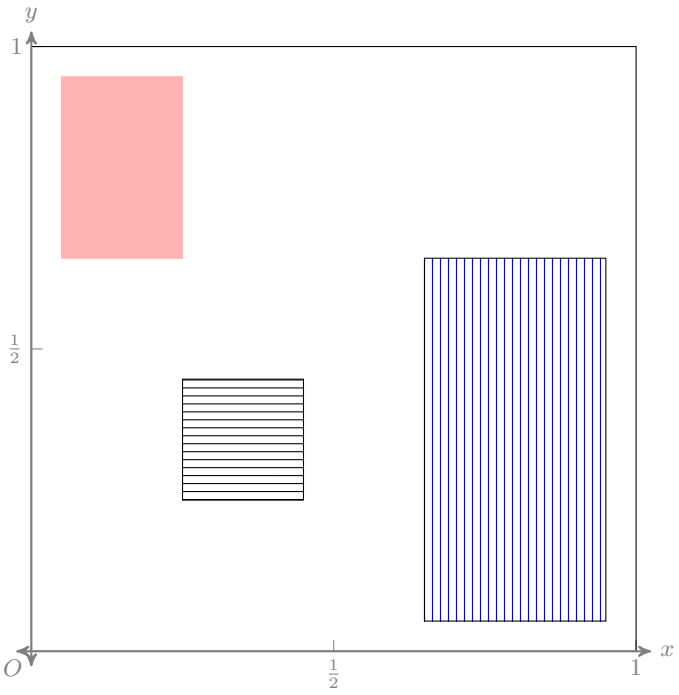


Figure 3.6: JUMP2D

Table 3.3: Timing results: JUMP2D

Numbers in parenthesis are numbers of outer iterations

n $\times 10^6$	#p	# level	Time (setup+solve) (# it) MUMPS	Time (setup+solve) (# it) SAI CG
9	1	7	6.5 + 27.4 (26)	6.5 + 39.3 (29)
144	16	9	8.4 + 35.9 (28)	7.8 + 40 (29)
225	25	9	9.5 + 36.9 (28)	8.1 + 42.9 (31)
324	36	10	9.0 + 50.7 (29)	7.9 + 42.8 (29)
441	49	9	9.3 + 46.1(29)	7.8 + 38 (31)
576	64	9	10.0 + 55.7 (30)	7.6+ 39.8 (30)
729	81	9	10.1 + 61.6 (28)	7.5+41.1 (31)
900	100	10	10.6 + 105 (28)	not converged in 50 cycles
1089	121	10	12.7 + 98.1 (29)	7.7 + 51.7 (31)
1296	144	10	13.9 + 144 (30)	7.5 + 54.5 (31)
1521	169	10	12.6 + 151 (29)	6.4 + 55.3 (33)
1764	196	11	13.9 + 210 (29)	7.8 + 59.5 (30)
2025	225	11	15.2 + 331 (29)	6.9 + 62 (34)

3. Problem JUMP3D: For a 3D problem we consider two different ways of partitioning the domain. The first partitioning approach divides the 3D domain into several slabs, as shown in Figure 3.7. In this partitioning every process has two neighbors. The parallel efficiency of different coarsest grid solvers for JUMP3D is shown in Figure 3.8 and Table 3.4. For smaller number of tasks, the parallel efficiency of MUMPS is better. If the number of tasks is more than 100, using SAI preconditioned CG to solve the coarsest grid is a better alternative.

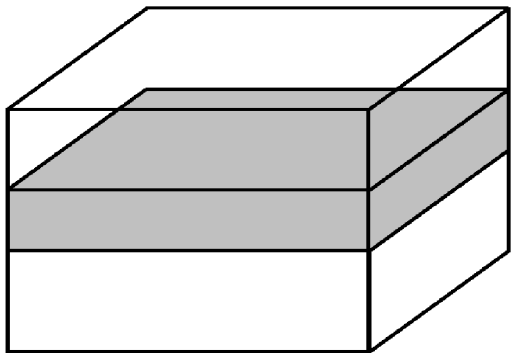


Figure 3.7

Table 3.4: Timing results: JUMP3D

Numbers in parenthesis are numbers of outer iterations

n $\times 10^6$	#p	# level	Time (setup+solve) (# it) MUMPS	Time (setup+solve) (# it) SAI CG
27	1	7	39.9 + 30.3 (13)	39.9+50.3 (19)
432	16	7	40.4 + 37.6 (14)	39.5 + 53.5 (19)
1323	49	8	40.3+44.9 (13)	38.1 + 46.1(15)
1728	64	9	39.3+53.2 (13)	37.4 + 64.6 (15)
2187	81	8	41.6 + 48.7 (13)	37.3 + 59 (16)
2700	100	9	42.6 +55.9 (14)	35.8 + 60.6 (17)
3375	125	10	42 + 59.3(14)	33.8 + 61.5 (17)
3888	144	10	38.1 + 72.6 (15)	35.8 + 61 (19)
5292	196	10	41 + 90 (15)	36.3 + 80.2(21)
5832	216	11	50 + 113 (15)	42.7 + 91.1(21)
6912	256	11	46.6 + 116 (15)	36.4 + 93.3 (21)

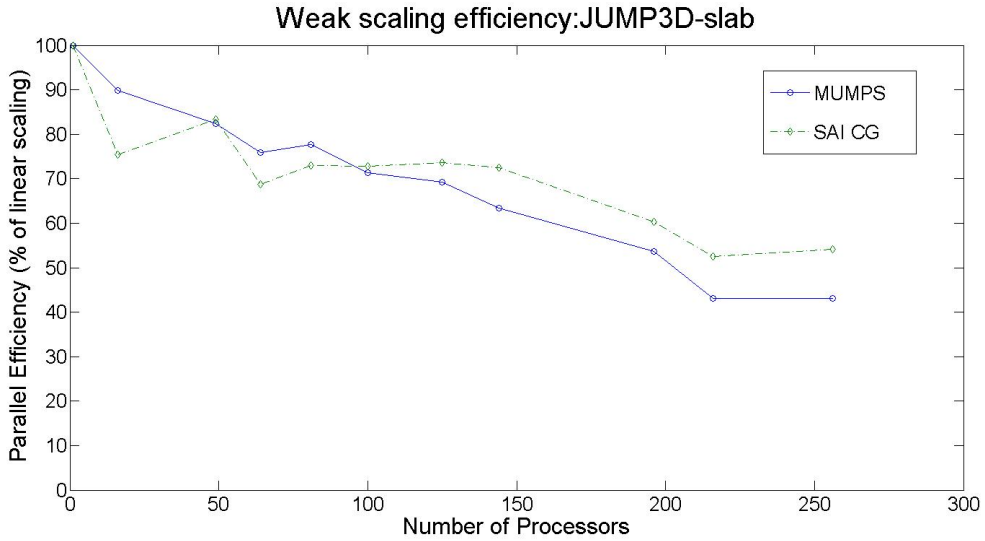


Figure 3.8: Parallel efficiencies for using MUMPS and using SAI CG as coarsest grid solver.

The second partition approach divides the 3D domain into several "pencil" shape blocks (Figure 3.9), and the result is shown in Table 3.5. In this case each processor has four neighbors. Several improvements are obtained: The number of K-cycles is grid size independent. The coarsening ratio is better and the number of levels is reduced, which also reduces the number of visits to the coarsest grid level. For the same number of unknowns in a processor, the communication cost is about 1/2 of that in the case using the first approach in partitioning the domain. The timing results again exhibit that SAI CG is a better coarsest grid solver than MUMPS for larger numbers of concurrent processes. Even for smaller numbers of concurrent tasks, SAI CG provides better parallel efficiency. Also the convergence is not affected by the low accuracy of the coarsest grid solution in SAI CG.

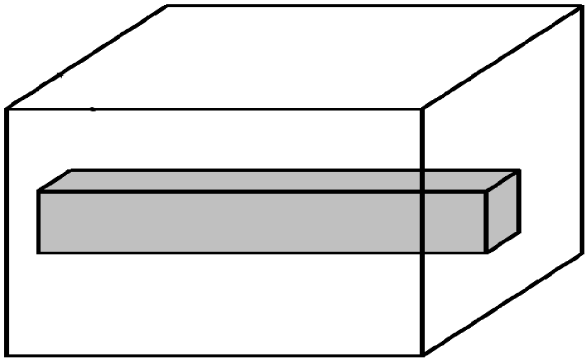


Figure 3.9

Table 3.5: Timing results: JUMP3D-pencil

Numbers in parenthesis are numbers of outer iterations

n $\times 10^6$	#p	# level	Time (setup+solve) (# it) MUMPS	Time (setup+solve) (# it) SAI CG
27	1	7	39.9 + 30.3 (13)	39.9+30.3 (13)
677	25	7	51.6 + 41.6 (13)	42.2+44.6 (13)
973	36	8	53 + 48 (13)	43.5+43.8 (13)
1323	49	8	52.9 + 49.3 (13)	43.8+44.9 (13)
1728	64	8	49.3+53.2 (13)	45 + 45.4 (13)
2187	81	8	50.4 + 52.8 (13)	46.4+ 46.8 (13)
2700	100	9	52.6 +55.9 (13)	47.5 + 48 (13)
3375	125	9	51.3 + 54.3(13)	48 + 47.2 (13)
3888	144	9	49 + 52.1 (13)	47.8 + 47 (13)
5292	196	9	54.2 + 61.9 (13)	47.3 + 47.6(13)
5832	225	9	56 + 113 (13)	48.8 + 49(13)
6913	256	9	54.8 + 212 (13)	51.6 + 50.2 (13)

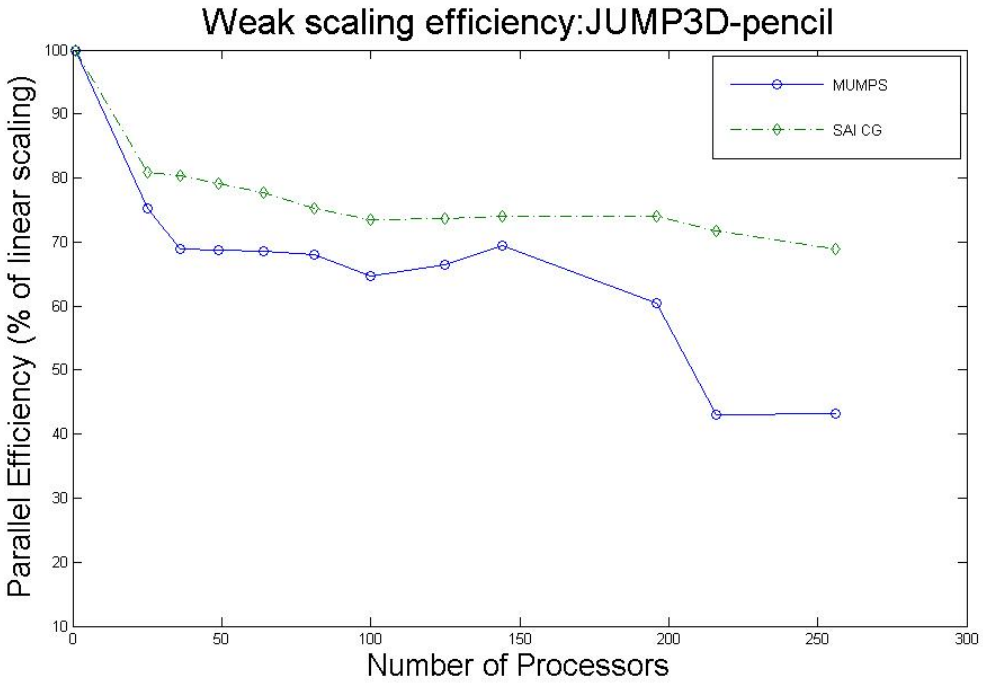


Figure 3.10: Parallel efficiencies for using MUMPS and using SAI CG on the coarsest grid solve.

3.5.1 Comparison between BoomerAMG and AGMG with SAI CG coarsest grid solver

In this subsection we investigate the performance and parallel efficiency of BoomerAMG ([35], [36]) and the AGMG using SAI CG as the coarsest grid solver. BoomerAMG is a parallel implementation of classical algebraic multigrid, as mentioned in Section 1.2. It is a part of the package Hypre, which can be found at `https://computation-rnd.llnl.gov/linear_solvers/software.php`. For comparison we use the classical algebraic multigrid V-cycle as a preconditioner for CG iterations. This can be done by choosing the appropriate parameters in the BoomerAMG package. The AGMG code used here was from [27] and we replaced the direct MUMPS solver with our own parallel SAI CG routine. A fragment of code showing the SAI preconditioned CG on the coarsest grid is included in the Appendix A. The subroutines `dagmgpar_csrmv` ,`dagmgpar_csrmv2`, `dagmgpar_sendrec` are used to perform parallel matrix-vector multiplications. The subroutine `MPI_Isend` communicates vector

entries needed for the computations between neighboring processors, and `MPI_WAITALL` synchronizes the behavior of `MPI_Isend`. All of the codes used for the SAI CG coarsest grid solving experiments can be found at `https://students.washington.edu/mchen01/PCG.txt`. The sparse approximate inverse of the coarsest grid matrix is constructed after the grid hierarchy is established in the setup phase. As mentioned in Section 3.1.5, we implement subroutines to compute the matrix G under the assumption that the coarsest grid matrix is symmetric and G^TG is the resulting sparse approximate inverse. In the preconditioning step of CG, following the notation in **Algorithm 1.2**, the formula is

$$z_k = G^T G r_k. \tag{3.12}$$

The PDE problem considered in the comparison is JUMP2D. In the numerical experiments the number of unknowns in each processor is kept as 4×10^6. The timing results are shown in Table 3.6. The results exhibit that the setup phase in classical algebraic multigrid degrades the parallel efficiency significantly, while the solution phase does not scale well in AGMG with SAI CG. These results are as expected. The total solving time favors the AGMG with SAI CG. The setup time increases as the number of processors increases in classical algebraic multigrid, which indicates that the setup phase affects the parallel efficiency significantly, suggesting that the number of unknowns in each processor should be smaller, for example, around 10^5 for 2D Model problems or 40^3 for 3D Model problems ([35], [36]). These numbers are significantly less than that in the test problems in the previous subsection. The other interesting fact for BoomerAMG is the larger number of coarser levels compared with that of AGMG. Since the cycle strategy used in BoomerAMG is the V-cycle, the number of visits to the coarsest grid level in each multigrid cycle is always 1. The coarsest grid solve seems not to affect the parallel efficiency of the solution phase. This is in contrast to that in the AGMG code where the efficiency of the coarsest grid solve is crucial to the parallel performance since in most the cases the number of visits to the coarser level in AGMG is 2 (similar to a W-cycle), as mentioned in Section 3.1.3.

Table 3.6: Timing results: JUMP2D. BoomerAMG vs AGMG with SAI CG

Numbers in parenthesis are numbers of outer iterations

n $\times 10^6$	#p	# level BoomerAMG	Time (setup+solve) (# it) BoomerAMG	# level AGMG	Time (setup+solve) (# it) AGMG with SAI CG
4	1	18	42.9 + 13.6 (10)	8	3.6 + 14.7 (29)
64	16	18	46.2 + 14.5 (11)	9	3.9 + 15.6 (31)
100	25	18	56.2 + 15.4 (11)	9	3.8 + 16.3 (29)
144	36	18	66.2 + 16.6 (11)	9	3.8 + 17.7 (30)
196	49	18	80.7 + 17.8 (12)	9	3.7 + 18.6 (31)
256	64	18	84.8 + 17.9 (12)	10	3.8 + 19.9 (29)
324	81	19	94.6 + 18.2 (12)	9	3.6 + 22.7 (31)
400	100	19	100.3 + 18.3 (12)	10	3.8 + 24.2 (32)
484	121	19	120 + 18.5 (12)	11	3.8 + 26.5 (33)
676	169	19	140 + 19.12 (13)	11	3.8 + 31.5 (35)

3.5.2 Conclusion

Three issues which may affect the parallel efficiency of multigrid methods have been discussed. Numerical experiments exhibit that the degraded convergence caused by parallel aggregation and smoothing can be recovered by using K-cycles in aggregation-based algebraic multigrid. The coarsest grid solver affects the parallel efficiency significantly. We found that the sparse approximate inverse preconditioned conjugate gradient method is a better alternative than the direct solver MUMPS on the coarsest grid solve for larger numbers of concurrent processes. For the 3D problem tested, different approaches for partitioning the 3D domain affect the coarsening efficiency in parallel aggregation. The number of multigrid levels and the convergence can be improved significantly if one partitions the domain properly. We have compared the parallel efficiency between a classical algebraic multigrid code (BoomerAMG) and AGMG with SAI CG on a 2D PDE problem. As expected, the parallel efficiency of each approach is affected by different factors. The setup phase appears to have

big impact on the performance of parallel classical algebraic multigrid, while the solution phase scales better than that in AGMG with SAI CG.

Appendices

Appendix A : A fragment of code for SAI PCG solving the coarsest grid system

```
!----------------------------------------------------------------------
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!----------------------------------------------------------------------
  SUBROUTINE dagmgpar_MUMPSpar(n,f,ijob,nzero)
    USE dagmgpar_mem
    IMPLICIT NONE
    INTEGER :: n,ijob
    REAL(kind(0.0d0)), TARGET :: f(n)
    !
    INTEGER(selected_int_kind(8)), SAVE :: nglobl, nnn
    REAL(kind(0.0d0)), SAVE :: iflop
    INTEGER(selected_int_kind(8)), ALLOCATABLE, SAVE :: nloc(:), idispl(:)
    INTEGER :: ierr, i, j, k, kk, ii, ext, kext, it
    !!!!!!!!!!!!!! variable used for PCG iterations !!!!!!!!!!!!!!
    INTEGER :: idum(1)
    real(kind(0.0d0)), allocatable :: pk(:),rk(:),zk(:),Gzk(:)
    real(kind(0.0d0)), allocatable, SAVE :: xk(:)
    real(kind(0.0d0)), allocatable :: rko(:), Apk(:)
    REAL(kind(0.0d0)) :: Appdot,Appdotl,rkodot,rkodotl,rkdot,rkdotl
    REAL(kind(0.0d0)), external :: DDOT
    REAL(kind(0.0d0)), external :: DNRM2
    INTEGER,PARAMETER:: IONE =1
    integer :: ier, nzcc
    character*12 filename
    REAL(kind(0.0d0)) ::  relativef , residf, residf0, normr, normr0
    REAL(kind(0.0d0)) ::  normx, normx0
    INTEGER, OPTIONAL :: nzero
    REAL(kind(0.0d0)) :: gdsqrt, testf
```

```
!!----------------------  end of parameters --------------------!!!

nnn=n
IF (ijob == -2) THEN
   !
   !
ELSE IF (ijob == 1) THEN
   allocate(xk(n))
   xk(1:n)=0.0d0
ELSE IF (ijob == 2) THEN

  !! ------------------------------------------------------------------
  !! When ijob = 2, the coarsest level is reached and preconditioned
  !! conjugate gradient is used to solve the coarsest grid system.
  !! ------------------------------------------------------------------

  allocate(pk(n),rk(n),Apk(n),rko(n),zk(n),Gzk(n))
  pk(1:n)= 0.0d0
  Apk(1:n)= 0.0d0
  zk(1:n) = 0.0d0
  rk(1:n) = 0.0d0
  if(n.NE.0) then
      call dagmgpar_sendrec(xk,idum,-1,dt(nlev)%lstout,dt(nlev)%ilstout &
               ,dt(nlev)%ilstin)
      call dagmgpar_csrmv(n,dt(nlev)%a,dt(nlev)%ja,n+1,dt(nlev)%il,xk,1,Apk,1,flop)
      call dagmgpar_csrmv(n,dt(nlev)%a,dt(nlev)%ja,n+1,dt(nlev)%iu,xk,1,Apk,2,flop)
      do i = 1,n
        Apk(i) = Apk(i) + dt(nlev)%a(i)*xk(i)   !! diagonal part
      enddo
  endif
```

```
IF (nneighi > 0) CALL MPI_WAITALL(nneighi,ireqi,ISTATUS,ier)
if(n.NE.0) then
   call dagmgpar_csrmv2(n,dt(nlev)%a,dt(nlev)%ja,n+1,dt(nlev)%iext,buffi,n+1, &
                        Apk, 2, flop, dt(nlev)%lstin)
   rk = f-Apk

   !! ---------------------------------------------------------------------
   !! Initial step of sparse approximate inverse preconditioning conjugate
   !! gradient iteration.
   !! ---------------------------------------------------------------------

   buffo(1:dt(nlev)%ilstout(nneigh+1)-1) = 0.0d0
   buffi(1:dt(nlev)%ilstin(nneigh+1)-1) = 0.0d0

   !! ----------------------------------------
   !! Communicate the matrix entries from other
   !! neighboring processors.
   !! ----------------------------------------
   call dagmgpar_sendrec(rk,idum,-1,dt(nlev)%lstout ,dt(nlev)%ilstout  &
                         ,dt(nlev)%ilstin)

   !! ----------------------------------------
   !! Compute G*rk => Gzk,  where Gt*G is the sparse
   !! approximate inverse of the coarsest matrix.
   !! ----------------------------------------
   call dagmgpar_csrmv(n,G,dt(nlev)%ja,n+1,dt(nlev)%il,rk,1,Gzk,1,flop)
   do i = 1,n
     Gzk(i) = Gzk(i) + G(i)*rk(i)   !! diagonal part
   enddo
```

```
   IF (nneighi > 0) CALL MPI_WAITALL(nneighi,ireqi,ISTATUS,ier)
   call dagmgpar_csrmv2(n,G,dt(nlev)%ja,n+1,dt(nlev)%iext,buffi,n+1,Gzk,2 &
         flop ,dt(nlev)%lstin)

   !! -------------------------------------------
   !! Compute Gt*Gzk => zk, now zk = Gt*G*zk
   !! -------------------------------------------
   buffo(1:dt(nlev)%ilstout(nneigh+1)-1) = 0.0d0
   buffi(1:dt(nlev)%ilstin(nneigh+1)-1) = 0.0d0
   call dagmgpar_sendrec(Gzk,idum,-1,dt(nlev)%lstout,dt(nlev)%ilstout, &
                   dt(nlev)%ilstin)
   call dagmgpar_csrmv(n,Gt,dt(nlev)%ja,n+1,dt(nlev)%iu,Gzk,1,zk,1,flop)
   do i = 1,n
     zk(i) = zk(i) + Gt(i)*Gzk(i)   !! diagonal part
   enddo
   IF (nneighi > 0) CALL MPI_WAITALL(nneighi,ireqi,ISTATUS,ier)
   call dagmgpar_csrmv2(n,Gt,dt(nlev)%ja,n+1,dt(nlev)%iext,buffi,n+1,zk,2,flop &
                   ,dt(nlev)%lstin)
   !! -------------------------------------------
   pk = zk
else
   normr = 0.0d0
endif

normr = DDOT(N,rk,IONE,rk,ione)   !!! compute <rk,rk>

CALL MPI_ALLREDUCE(normr,normr0,1,MPI_DOUBLE_PRECISION,    &
                  MPI_SUM,ICOMM,ierr)
residf0 = dsqrt(normr0)
relativef = residf0/residf0
```

```
 !!! Broadcast the relative residual
 CALL MPI_BCAST(relativef,1,MPI_DOUBLE_PRECISION,0,ICOMM,ierr)

 j = 0
 it = 0

!! ------------------------------------------------------------------------
!! The DO WHILE loop carries out sparse approximate inverse preconditioned
!! conjugate gradient iterations.
!! The relative residual tolerance is set to be 0.9.
!! ------------------------------------------------------------------------
DO WHILE ( relativef > 0.9 )

   !! ------------------- compute A*pk = Apk --------------
   if(n.NE.0) then
     call dagmgpar_sendrec(pk,idum,-1,dt(nlev)%lstout,dt(nlev)%ilstout  &
                     ,dt(nlev)%ilstin)
     call dagmgpar_csrmv(n,dt(nlev)%a,dt(nlev)%ja,n+1,dt(nlev)%il,pk  &
                     ,1,Apk,1,flop)
     call dagmgpar_csrmv(n,dt(nlev)%a,dt(nlev)%ja,n+1,dt(nlev)%iu,pk, &
                     ,1,Apk,2,flop)
     do i = 1,n
       Apk(i) = Apk(i) + dt(nlev)%a(i)*pk(i)   !! diagonal part
     enddo
   endif
   !! ------------------------------------------------
   IF (nneighi > 0) CALL MPI_WAITALL(nneighi,ireqi,ISTATUS,ier)

   !! ------------ Compute <Apk, pk> = Appdot----------------
```

```
if(n.NE.0) then
  call dagmgpar_csrmv(n,dt(nlev)%a,dt(nlev)%ja,n+1,dt(nlev)%iext,buffi,n+1, &
                      Apk,2,flop, dt(nlev)%lstin)
  Appdotl = DDOT(N, pk, IONE, Apk, IONE)
else
  Appdotl=0.0d0
endif
  CALL MPI_ALLREDUCE(Appdotl,Appdot,1,MPI_DOUBLE_PRECISION,    &
              MPI_SUM,ICOMM,ierr)
  CALL MPI_BCAST(Appdot,1,MPI_INTEGER,0,ICOMM,ierr)
!! ------------------------------------------------------

!! --------------- Compute <rk, zk> = rkodot--------------
if(n.NE.0) then
  rkodotl = DDOT(N, rk, IONE, zk, IONE)
else
  rkodotl=0.0d0
endif

CALL MPI_ALLREDUCE(rkodotl,rkodot,1,MPI_DOUBLE_PRECISION,    &
              MPI_SUM,ICOMM,ierr)
CALL MPI_BCAST(rkodot,1,MPI_INTEGER,0,ICOMM,ierr)

if(n.NE.0) then

   xk = xk +(rkodot/Appdot)*pk   !!! compute new xk
   rk = rk -(rkodot/Appdot)*Apk  !!! compute new rk

!!! ---------- Sparse approximate inverse preconditioning step.  --------- !!!
!!! Multiply zk by Gt*G = M^{-1}, where Gt*G is the sparse approximate
```

```
!!!  inverse of the coarsest matrix.
!!! ------------------------------------------------------------------
   buffo(1:dt(nlev)%ilstout(nneighi+1)-1) = 0.0d0
   buffi(1:dt(nlev)%ilstin(nneighi+1)-1) = 0.0d0


   !! ----------------- Compute G*zk => Gzk ------------------
   call dagmgpar_sendrec(rk,idum,-1,dt(nlev)%lstout,dt(nlev)%ilstout, &
                         dt(nlev)%ilstin)
   call dagmgpar_csrmv(n,G,dt(nlev)%ja,n+1,dt(nlev)%il,rk,1,Gzk,1,flop)
   do i = 1,n
      Gzk(i) = Gzk(i) + G(i)*rk(i)   !! diagonal part
   enddo
   IF (nneighi > 0) CALL MPI_WAITALL(nneighi,ireqi,ISTATUS,ier)
   call dagmgpar_csrmv2(n,G,dt(nlev)%ja,n+1,dt(nlev)%iext,buffi,n+1,&
                         Gzk,2,flop,dt(nlev)%lstin)

   buffi(1:dt(nlev)%ilstin(nneighi+1)-1) = 0.0d0
   buffo(1:dt(nlev)%ilstout(nneighi+1)-1) = 0.0d0

   !! ----- Compute Gt*Gzk => zk, where zk = G*Gt*zk ---------
   call dagmgpar_sendrec(Gzk,idum,-1,dt(nlev)%lstout,dt(nlev)%ilstout, &
                         dt(nlev)%ilstin)
   call dagmgpar_csrmv(n,Gt,dt(nlev)%ja,n+1,dt(nlev)%iu,Gzk,1,zk,1,flop)
   do i = 1,n
     zk(i) = zk(i) + Gt(i)*Gzk(i)   !! diagonal part
   enddo
   IF (nneighi > 0) CALL MPI_WAITALL(nneighi,ireqi,ISTATUS,ier)
   call dagmgpar_csrmv2(n,Gt,dt(nlev)%ja,n+1,dt(nlev)%iext,buffi,n+1 &
                         ,zk,2,flop,dt(nlev)%lstin)
```

```
      !! ------ Compute <rk, zk> = rkdot1----------
      rkdotl = DDOT(N, rk, IONE, zk, IONE)
    else
      rkdotl = 0.0d0
    endif

    CALL MPI_ALLREDUCE(rkdotl,rkdot,1,MPI_DOUBLE_PRECISION,    &
                  MPI_SUM,ICOMM,ierr)
    CALL MPI_BCAST(rkdot,1,MPI_INTEGER,0,ICOMM,ierr)

    !! ----------- Compute new pk -----------------
    if(n.NE.0) then
      pk = zk +(rkdot/rkodot)*pk
    endif

    !! -------- Compute the relative residual -----.
    normr = DDOT(N,rk,IONE,rk,ione)
    CALL MPI_ALLREDUCE(normr,normr0,1,MPI_DOUBLE_PRECISION,    &
                    MPI_SUM,ICOMM,ierr)
    residf = dsqrt(normr0)
    relativef = residf/residf0

   ENDDO

   f = xk

END IF
```

```
   RETURN

  END SUBROUTINE dagmgpar_MUMPSpar

!-----------------------------------------------------------------------
!!!!!!!!!!!!!!!!!!
!-----------------------------------------------------------------------
```

BIBLIOGRAPHY

[1] R. E. Alcouffe, A. Brandt, J.E. Dendy, Jr and J. W. Painter *The multi-grid method for the diffusion equation with strongly discountinuous coefficients*, SIAM J. Sci. Stat. Comput., 2 (1981), pp. 430-454.

[2] A. Brandt. *Multigrid solvers for non-elliptic and singular-perturbation steady-state problems*. The Weizmann Institute of Science, Rehovot, Israel, 1981 (unpublished).

[3] A. Brandt. *Algebraic multigrid theory: the symmetric case*. Appl. Math. Comp. **19**, 23 - 56, (1986).

[4] D. Braess, *Towards algebraic multigrid for elliptic problems of second order*. Computing, 55 (1995), pp. 379 -393.

[5] M. Benzi, M. Tuma, *A comparative study of sparse approximate inverse preconditioners*. Applied Numerical Mathematics, 30 (1999), pp. 305 - 340.

[6] W. L. Briggs, V. E. Henson, S. F. McCormick *Multigrid tutorial*. SIAM 2000.

[7] D. Bulgakov. *Multilevel iterative technique and aggregation concept with semi-analytical preconditioning for solving boundary-value problems*. Comm. Numer. Method Engrng., 9 (1993), pp. 649-657.

[8] M. Chen, A. Greenbaum. *Analysis of an Aggregation-Based Algebraic Two-grid Method for a Rotated Anisotropic Diffusion Problem*. submitted Numer. Linear Algebra with Appl.

[9] E. Chow, R. D. Falgout, J. J. Hu, R. S. Tuminaro and U. M. Yang *A survey of parallelization techniques for multigrid solvers*, Parallel Processing For Scientific Computing, SIAM, series on Software, Environments, and Tools (2006).

[10] E. Chow. *Parallel Implementation and Performance Characteristics of Least Squares Sparse Approximate Inverse Preconditioners*, Int. J. High-Perform. Comput. Appl. (2000).

[11] J.E. Dendy. *Black box multigrid for nonsymmetric problems*. Appl. Math.and Comp., Volume 13, Issues 34, 1983, pp. 261283.

[12] J. E. Dendy and J. D. Moulton. *Black box multigrid with coarsening by a factor of three*. Numer. Linear Algebra with Appl. 17 : 577-598, (2010).

[13] R. D. Falgout, *An Introduction to Algebraic Multigrid*, Computing in Science and Eng. Lawrence L. Nat. Lab. Report 2006.

[14] R. D. Falgout and P. Vassilevski, *On two-grid convergence estimate*, Num. Lin. Alg. Appl., 12(2005), pp. 471-494.

[15] H. Avron, E. Ng, and S. Toledo, *A generalized Courant-Fischer minimax theorem*, technical report LBNL-6393E, Lawrence Berkeley National Lab, August, (2008).

[16] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, 1997.

[17] H. Kim, J. Xu and L. Zikatanov , *A multigrid method based on graph matching for convectiondiffusion equations*. Numer. Lin. Alg. Appl., **10** (2003), pp. 181-195.

[18] J. Linden, G. Lonsdale, H. Ritzdorf, A. Schller, *Scalability aspects of parallel multigrid*. Future Generation Computer Systems 10 (1994), pp. 429-439.

[19] S. F. McCormick, *Multigrid Methods for Variational Problems: General Theory for the V- Cycle*, SIAM J. Numer. Anal. 22 (1985), pp. 634 -643.

[20] A. Muresan and Y. Notay, *Analysis of aggregation-based multigrid*, SIAM J. Sci. Comput. 30 (2008), pp. 1082-1103.

[21] A. Napov and Y. Notay, *When does two-grid optimality carry over to the V-cycle?*, Numer. Linear Algebra Appl., 17 (2010), pp. 273-290.

[22] A. Napov and Y. Notay, *Algebraic analysis of aggregation-based multigrid*, Lin. Alg. Appl., 18 (2011), pp. 539-564.

[23] A. Napov and Y. Notay, *An algebraic multigrid method with guaranteed convergence rate*, SIAM J. Sci. Comput., (2012).

[24] Y. Notay, *An aggregation-based algebraic multigrid method*, ETNA, 37 (2010), pp. 123 - 146.

[25] Y. Notay, *Aggregation-based algebraic multigrid for convection-diffusion equations*, Tech. Rep. GANMN 11-01, Universite Libre de Bruxelles, Brussels, Belgium, (2011).

[26] Y. Notay, private communication.

[27] Y. Notay, *AGMG: Iterative solution with AGgregation-based algebraic MultiGrid*, http://homepages.ulb.ac.be/ ynotay/AGMG.

[28] Y. Notay and P. S. Vassilevski , *Recursive Krylov-based multigrid cycles*, Numer. Linear Algebra Appl. 2006; **0**: 1-20.

[29] K. Stuben. *A Review of Algebraic Multigrid*. J. of Computational and Applied Mathematics 128 (2001) 281309.

[30] U. Trottenberg, C. Oosterlee, A. Schuller. *Multigrid*. Academic Press, 2000.

[31] P. Vanek, M. Brezina and R. Tezaur, *Two-grid method for linear elasticity on unstructured meshs*. SIAM J. Sci. Comput., 21 (1999), pp. 900-923.

[32] P. Vanek, M. Brezina and J. Mandel, *Convergence of algebraic multigrid based on smoothed aggregation.* Numer. Math. (2001) 88: 559-579.

[33] P. Vanek, M. Brezinaand J. Mandel, *Algebraic multigrid based on smoothed aggregation for second and forth order elliptic problems*. Computing, 56 (1996), pp. 179-196

[34] P. Vassilevski *Multilevel Block Factorization Preconditioners*. Springer, New York, 2008.

[35] U. M. Yang, *Parallel Algebraic Multigrid Methods - High Performance Preconditioners*. Numerical Solution of Partial Differential Equations on Parallel Computers, Lecture Notes in Computational Science and Engineering Volume 51, 2006, pp 209-236.

[36] V. E. Henson, U. M. Yang, *BoomerAMG: A parallel algebraic multigrid solver and preconditioner*. Applied Numerical Mathematics, Volume 41, Issue 1, April 2002, Pages 155177.