UCRL-LR-108186
Distribution Category UC-705

UCRL-LR--108186

DE92 000096

A Domain Decomposition Algorithm for Solving Large Elliptic Problems

Matt Patrick Nolan
(M.S. Thesis)

Manuscript date: January 1991

LAWRENCE LIVERMORE NATIONAL LABORATORY
University of California • Livermore, California • 94551

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Abstract

An algorithm which efficiently solves large systems of equations arising from the discretization of a single second-order elliptic partial differential equation is discussed. The global domain is partitioned into not necessarily disjoint subdomains which are traversed using the Schwarz Alternating Procedure. On each subdomain the multigrid method is used to advance the solution. The algorithm has the potential to decrease solution time when data is stored across multiple levels of a memory hierarchy. Results are presented for a virtual memory, vector multiprocessor architecture. A study of choice of inner iteration procedure and subdomain overlap is presented for a model problem, solved with two and four subdomians, sequentially and in parallel. Microtasking multiprocessing results are reported for multigrid on the Alliant FX-8 vector-multiprocessor. A convergence proof for a class of matrix splittings for the two-dimensional Helmholtz equation is given.

Acknowledgements

I would like to express my gratitude to those individuals and institutions that helped make this work possible. I appreciate the comments and criticism of Dr. John Feo who was especially helpful during the early stages of organizing this thesis. I thank Dr. Joel Dendy Jr. for his helpful comments and for providing his multigrid code which became the basis the multigrid solver used in my thesis. I thank Dr. Garry Rodrigue for patiently waiting for me to complete my thesis. I express heart felt appreciation towards a fellow student, now Dr. Bracy Elton; without his expertise in document preparation, helpful discussions, and moral support I would not finished my Masters Degree. Finally, to Dr. John Bolstad, I am indebted for all that he taught me regarding the technical aspects of multigrid and technical writing.

I acknowledge and greatly appreciate the financial support of the Lawrence Livermore National Laboratory through its Student Employee Program. I thank the Computing and Mathematics Research Division of Lawrence Livermore National Laboratory and the Advanced Computing Resource Facility of the Mathematics and Computer Science Division at Argonne National Laboratory for the use and availability of their computer resources.

Contents

Contents

Chapter 1

The Schwarz Alternating Procedure

In the 1860's Schwarz [Sch69] found that for a region consisting of the union of two rectangular regions or disks, he could construct a sequence of solutions of the Laplace equation on these subregions which would converge to the solution of the Laplace equation on the union. Picard [Pic90] called it "the Schwarz alternating procedure" and used it to solve a nonlinear elliptic equation in 1890. Mathematicians now refer to his method as the SAM, Schwarz alternating method, or SAP, Schwarz alternating procedure. We use the terminology SAP. The following is description of a simple version of SAP taken from [Tan87]:

1.1 A Two Subdomain Example

Consider the Dirichlet problem for an elliptic operator L

$$\begin{cases} L(u) = f & x \in \Omega \\ u \,|_{\Gamma_\Omega} = \psi & x \in \Gamma_\Omega \end{cases} \tag{1.1}$$

where Ω is a bounded region in d-dimensional space, Γ_Ω is the boundary of Ω and $\mathbf{x} = \{x_1, x_2, \cdots, x_d\}$ is the independent variable, u is a function which maps $\mathbf{x}$ to the real numbers, $\Re$, is twice continuously differentiable and continuous up to the boundary. It is assumed that the solution of this problem exists and is unique.

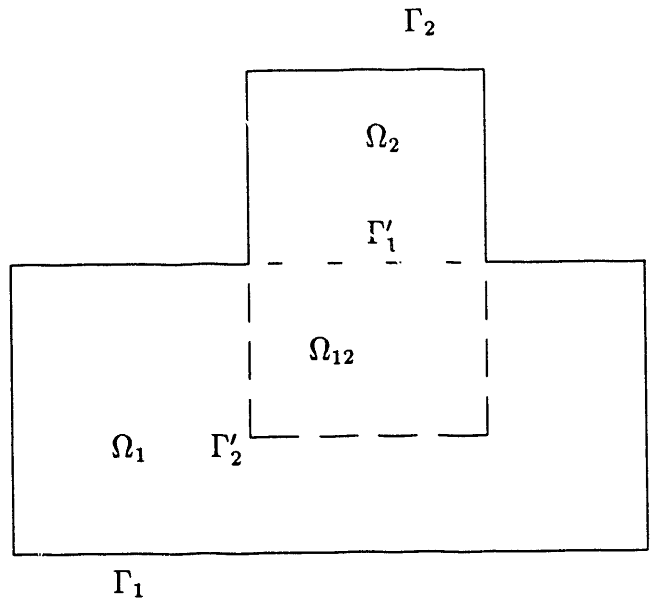


Figure 1.1: Two overlapping subdomains.

Schwarz split the solution domain Ω into two overlapping subdomains Ω_1 and Ω_2. Let $\Omega_{12} = \Omega_1 \bigcap \Omega_2 \neq 0$ and let Γ_{Ω_1}, Γ_{Ω_2} and $\Gamma_{\Omega_{12}}$ denote the boundaries of Ω_1, Ω_2 and Ω_{12} respectively. We denote by Γ_1' that part of Γ_{12} lying in Ω_2, and by Γ_2' that part of Γ_{12} lying in Ω_1. Then

$$\Gamma_{\Omega_1} = \Gamma_1 \cup \Gamma_1',$$

$$\Gamma_{\Omega_2} = \Gamma_2 \cup \Gamma_2',$$

as in Figure 1, where

$$\begin{aligned} \Gamma_1 &= \Gamma_\Omega \bigcap \Gamma_{\Omega_1}, \\ \Gamma_1' &= \Gamma_{\Omega_1} \bigcap \Gamma_{\Omega_{12}}, \\ \Gamma_2 &= \Gamma_\Omega \bigcap \Gamma_{\Omega_2}, \\ \Gamma_2' &= \Gamma_{\Omega_2} \bigcap \Gamma_{\Omega_{12}}. \end{aligned}$$

We refer to the dashed lines of Figure 1 which correspond to Γ_1' and Γ_2' as *pseudo-boundaries.*

From this splitting we formulate two coupled problems

$$\left\{\begin{array}{lcll} L(u_1) & = & f & x \in \Omega_1 \\ u_1|_{\Gamma_{\Omega_1}} & = & \left\{\begin{array}{ll} \psi & \mathrm{x} \in \Gamma_1 \\ u_2 & \mathrm{x} \in \Gamma_1' \end{array}\right. & \end{array}\right. \tag{1.2}$$

and

$$\left\{\begin{array}{lcll} L(u_2) & = & f & x \in \Omega_2 \\ u_2|_{\Gamma_{\Omega_2}} & = & \left\{\begin{array}{ll} \psi & \mathrm{x} \in \Gamma_2 \\ u_1 & \mathrm{x} \in \Gamma_2'. \end{array}\right. & \end{array}\right. \tag{1.3}$$

Clearly, u, the solution of (1.1), is the solution of (1.2) and (1.3). It is also clear that:

$$\begin{array}{lcll} u_1 & = & u_2, & \mathrm{x} \in \Omega_{12} \\ u_1 & = & u, & \mathrm{x} \in \Omega_1 \\ u_2 & = & u, & \mathrm{x} \in \Omega_2. \end{array}$$

Thus, problem (1.1) is equivalent to the pair of problems (1.2) and (1.3). Since there are unknowns which are coupled in the boundary conditions, we cannot solve the two problems independently. By constructing an initial guess $u|_{\Gamma_1'} = \psi_0$, we can form a sequence $\{u_1^{(i)}, u_2^{(i)}\}$, $i \geq 1$, as follows:

$$\left\{\begin{array}{lcll} L(u_1^{(0)}) & = & f & x \in \Omega_1 \\ u_1^{(0)}|_{\Gamma_{\Omega_1}} & = & \left\{\begin{array}{ll} \psi & \mathrm{x} \in \Gamma_1 \\ \psi_0 & \mathrm{x} \in \Gamma_1' \end{array}\right. & \end{array}\right. \tag{1.4}$$

$$\left\{\begin{array}{lcll} L(u_2^{(i)}) & = & f & x \in \Omega_2 \\ u_2^{(i)}|_{\Gamma_{\Omega_2}} & = & \left\{\begin{array}{ll} \psi & \mathrm{x} \in \Gamma_2 \\ u_1^{i-1} & \mathrm{x} \in \Gamma_2' \end{array}\right. & \end{array}\right. \tag{1.5}$$

$$\left\{\begin{array}{lcll} L(u_1^{(i)}) & = & f & x \in \Omega_1 \\ u_1^{(i)}|_{\Gamma_{\Omega_1}} & = & \left\{\begin{array}{ll} \psi & \mathrm{x} \in \Gamma_1 \\ u_2^{i} & \mathrm{x} \in \Gamma_1' \end{array}\right. & \\ & & & i = 1, 2, \cdots \end{array}\right. \tag{1.6}$$

We can now show that the sequence $\{u_1^{(i)}, u_2^{(i)}\}$ will converge to the solutions $\{u_1, u_2\}$ of (1.2) and (1.3) under certain conditions mentioned in §1.3. Then, from the solution of (1.2) and (1.3), we construct the solution of (1.1).

Convergence proofs for the continuous problem rely on finding the analytical solution as each subdomain is visited. For the discrete problem one solves the subdomain problem using some numerical method. One need not solve with the same numerical method on each subdomain. We call the act of solving on a subdomain (with a particular numerical method) the *inner iteration procedure.* We call a sequence of inner iteration procedures applied to each of the subdomains an *outer iteration.* Also, an inner iteration procedure isn't restricted to an iterative method; it may be some type of direct solution method, in which case we say the inner iteration procedure has an iteration count of one. An outer iteration advances the solution for all grid points in the domain Ω. An inner iteration procedure is a method used to advance the solution of a subdomain problem. Furthermore, in chaotic SAP schemes the inner iteration procedure on Ω_i may vary from one outer iteration to the next. For some outer iteration, an inner iteration procedure on Ω_i could be a null iteration where no update of the unknowns in Ω_i occurs.

"Unlike some other techniques which usually are precise procedures for solving problems, the SAP basically gives us only a philosophy for solving a problem. The freedoms inherent in the SAP provide great opportunities to incorporate many other techniques in order to obtain good performance.

- Freedom in the geometrical shapes of the subproblems. This freedom makes it possible to tailor the subregions to meet the requirements imposed by fast solvers or by grids.

- Freedom in the solution techniques for subproblems. We are able to choose different solution techniques for different subproblems. It is also possible to use different ways to obtain the solution of the same subproblem in the different stages of computation, allowing us to use an optimal approach at any particular moment or in any particular location. This is a unique feature of the SAP.

- Freedom in the numerical model for each of subproblem. Special boundary shapes or local behavior of the solution need a special treatment in the modeling. The decoupled subproblems allow us to localize the special treatment to the place where it is needed. Composite grids are a good example of this.

- Freedom in the number of subproblems. This freedom will permit us to adapt this algorithm to different degrees of parallelism". [Tan87]

1.2 Development and History

In 1869 Schwarz [Sch69] first developed a method he called an alternating method to prove the existence of the solution of the Dirichlet problem for the Laplace equation on a union of two overlapping domains. Soon Neumann [Neu90] observed that a similar idea could be applied to the solution of the Dirichlet problem in a region that is the intersection of two other overlapping regions. Later Poincaré [Poi90] developed his balayage method, which is similar to Schwarz's method. Poincaré was also concerned with existence proofs rather than computation.

During the 1930's many Russian mathematicians applied Schwarz's method to problems in elastostatics. They treated the solution process of SAP as a search for the minimum of a variational problem. This new way of thinking provided possibilities for enlarging application areas. Gorgidz applied the SAP to a plane problem in the theory of elasticity. Almost at the same time, Mikhlin [Mik34] generalized this idea to a biharmonic problem. He proved the convergence of the SAP to the solution of the second elastostatic boundary value problem. A more general proof of this method for the second boundary value problems of elasticity in three dimensions was sketched out by Sobolev [Sob36]. He reduced the consideration of convergence of the sequence of approximations to a study of convergence to the minimum of the integral of strain energy. He applied the variational method to prove the convergence of the Schwarz algorithm for the Laplace equation and the equations of elasticity theory, but the convergence of the Schwarz algorithm was established only in the mean.

In the early 1950's, Kantorovich and Krylov [KK58] gave a set of five sufficient conditions which guarantee the convergence of the SAP in the continuous domain. These conditions encompass most of the areas to which the SAP can be applied. We give the conditions in §1.3. In 1951, Mikhlin [Mik51, Mik65] applied Sobolev's method to establish the convergence of the algorithm for the general second order linear elliptic equation.

After the 1960's people began to apply the SAP to numerical computations rather than to existence proofs or theoretical analysis. Some new algorithms such as alternating direction implicit methods or Fourier analysis/cyclic reduction methods were the state of the art at the time, but they could only be applied to rectangular regions. The SAP was a natural way of applying these methods to a union of rectangular regions.

D'Jakonov [D'J62] derived some work estimates for solving Poisson's equation to a given precision on overlapping rectangular regions using the SAP. The rectangular solutions are found by the alternating direction implicit method, or a similar method of D'Jakonov's, applied to the five-point difference approximation.

Werner [Wer60, Wer63] considered application of the SAP to any linear second-order elliptic partial differential equation with boundary conditions of the third type – also called Robin's or mixed boundary conditions. He proved the existence of a continuous solution and gave error bounds for a solution which satisfies the differential equation, but only approximates the boundary data. He presented numerical results for the Laplace equation on an L-shaped region with mixed boundary conditions. He expressed the rectangular solutions as a double finite Fourier series. Mysovskih [Mys59], Kang and Wang [KW59] and Miller applied the finite difference analogs to the SAP. In [Mil65], Miller shows four conditions are sufficient for convergence of the solution of the numerical SAP to the original continuous problem. We mention them in the section to follow. In the same paper he also gives work estimates for several cases. Fairweather and Mitchell [FM66] applied the SAP to a nine-point difference approximation on an L-shaped region. Their inner iteration procedure was a modified alternating direction implicit method.

Dupont [Dup67] generalized their idea to the equation $\nabla \cdot (a \nabla u) = \rho$, and derived work estimates on overlapping rectangular regions. Stoutemyer [Sto72, Sto73] applied the SAP and Neumann's variant to the Laplace equation on the union and intersection of two overlapping spherical balls to compute the capacity of a lens. He paid particular attention to the treatment of singularities in the Poisson kernel and at the corners of the region. Lions [Lio78] applied the SAP to the variational problem.

As we mentioned earlier, applications of the SAP to composite mesh methods have attracted people's attention for some time. Volkov [Vol68] first presented a second order composite method for the Dirichlet problem for the Laplace equation; he also used the SAP to solve the system of linear equations. Later, Starius [Sta77] generalized this idea to linear second order elliptic equations.

Also working with composite mesh methods, Linden [Lin81] compared and contrasted two ways to combine multigrid techniques with the SAP for a model problem. Stüben and Trottenberg expound upon Linden's work in [ST82a]. One method uses multigrid as the inner iteration procedure of the SAP. Though the efficiency of this method is better that of the corresponding SOR method, the total efficiency is limited by the convergence properties of the SAP. The convergence rate of the SAP depends on the geometrical situation; roughly, the smaller the overlap of the subdomains, the slower the convergence. Appendix 4 contains tables which show outer iteration counts for various Schwarz splittings with a multigrid inner iteration procedure. Because of the relative slowness of this method Stüben and Trottenberg refer to it as the "naive" combination of the SAP with multigrid. In their method a multigrid hierarchy of composite meshes is used; the principle of the SAP is applied only within the relaxation process for smoothing. The efficiency of this method is empirically observed to be essentially independent of the overlap. Hackbush [Hac85] also considers the combination of multigrid and the SAP. These authors mention the concept of multiprocessing multigrid using the SAP. However, they dismiss the idea of high level multiprocessing through domain decomposition of the global domain, the "naive" approach, as ineffective, in part, at least, because of the associated slower convergence rate [Den88].

Glowinski, Dinh and Periaux [DGP80, GDP80] formulated a conjugate gradient variant of the SAP for solving the Navier-Stokes equations. Essentially they reduced the problem to a minimization problem on the intersection of two overlapping regions.

When computer technology advanced to parallel processing, the inherent parallelism in this algorithm obtained new appeal. Kang [KCSQ85] extended the variational form of the SAP to general second order elliptic partial differential equations and tried to apply it to parallel computations. Unfortunately, his proof for convergence for an asynchronous version of the SAP was wrong [Tan87].

Rodrigue [RS84a, RS84b, Rod86, RS85] recast the SAP in terms of numerical linear algebra so that classical techniques of acceleration could be applied. A Jacobi splitting of the modified matrix problem v·as studied in these papers.

Analysis and experiments show that the convergence rate of the plain SAP can be further improved. Many authors have independently found that SOR acceleration of the SAP works efficiently. Oliger, Skamarock and Tang [OST86] also noticed the sensitivity of the relaxation parameter is related to the overlap. Theoretical estimates of the convergence rate and choice of the best relaxation parameter for the model problem are given. In the same paper mentioned above [KCSQ85], Kang also proved the convergence of the SOR acceleration for the finite element method. Meier [Mei86] had also proposed a parallel SOR variant of the SAP.

Some computer scientists are studying the SAP as a method for decomposing a problem such that different solvers can be used on different subdomains [PR89], with the hope of diminishing the work needed to solve some computationally expensive problems. With this strategy, one hopes that on some subdomains computationally less expensive inner iteration procedures can be used.

1.3 Sufficient Conditions for Convergence

Several proofs of convergence for the SAP exist [KK58, CH62, KCSQ85]. The most general case was given by the Russian mathematicians Kantorovich and Krylov in the 1950's [KK58]. This elucidation of Kantorovich's and Krylov's sufficient conditions is also taken from [Tan87]. They showed that five conditions suffice for the SAP to converge to the solution of the continuous boundary value problem (1.1).

The five conditions are:

- **Uniqueness.** Two solutions u and u' which satisfy equation (1.1) in Ω, are bounded, and have identical values on the boundary Γ_Ω (except, perhaps, at a finite set of points), are identically equal in Ω.

- **Monotonicity.** Two bounded functions u and u' which satisfy equation (1.1) in Ω and have $u \geq u'$ on Γ_Ω (except, perhaps, at a finite set of points) will satisfy $u \geq u'$ everywhere in Ω.

- **Limit solution.** The limit of any monotone and uniformly bounded sequence of solutions to equation (1.1) is also a solution of (1.1).

- **Maximum principle.** A solution to (1.1) cannot have either a positive interior maximum or a negative interior minimum. For linear problems this implies the monotonicity condition.

- **Continuity onto the boundary.** If $u = f$ on a boundary segment except at a point P inside the segment, where f is continuous on this segment, then the solution $u(Q)$ for Q in Ω approaches $f(P)$ as $Q \to P$.

1.4 Conditions for Numerical Convergence

The numerical analog to the SAP is straightforward. We can discretize the problems (1.4)-(1.6), and then solve them numerically. Miller [Mil65] showed that the following conditions suffice for convergence of the discrete SAP solution to the continuous problem:

- **Existence of a continuous solution.** The solution of the continuous problem (1.1) exists. This implies that the solutions of problems (1.2) and (1.3) exist.

- **Existence of the discrete solutions.** Solutions of the discretized problems (1.2) and (1.3) exist.

- **Convergent discretization.** Discrete approximations of (1.2) and (1.3) are convergent to the continuous solution of (1.2) and (1.3).

- **Contraction mapping.** There exist contraction numbers $Q_1 \leq 1, Q_2 \leq 1$, such that $Q_1 Q_2 < 1$ and

$$\|u_1 - \tilde{u_1}\| \leq Q_1 \epsilon_1,$$

$$\|u_2 - \tilde{u_2}\| \leq Q_2 \epsilon_2,$$

  where ϵ_1, ϵ_2 are perturbations of the boundary data on Γ'_1, Γ'_2, and $\tilde{u_1}, \tilde{u_2}$ are the perturbed solutions which correspond to u_1 and u_2.

For linear elliptic differential equations we can also express problem (1.1) in an equivalent variational form; then it is possible to prove that the solution sequence of the corresponding finite element method is a convergent minimization sequence. The independence between convergence and the ordering of the solutions of the subregions can be easily shown in variational form [KCSQ85]. We can also recast the numerical analog of the SAP as a modified matrix problem, then prove its convergence. From analysis of the linear algebra analog of the SAP for the model problem many new results can be obtained by applying classical acceleration schemes.

1.5 Minimizing Spatial Locality of Reference

In this thesis, we take a new approach in applying the SAP. For a class of problems we use the SAP to create an algorithm which is faster and which allows one to solve problems that were previously unsolvable on a given architecture due to both physical and virtual memory constraints.

An SAP algorithm can decrease the solution time when it limits the degree of spatial locality[1] to an amount smaller than the main memory of the computer. In this case, we are solving a large system of equations, one that ordinarily we could not solve without storing data on disk. A problem is large if for any reason increasing its size causes a decrease in the rate of solution in terms of some normalized quantity such as floating point operations per second. We can decrease the solution time because accessing data on disk is one to four orders of magnitude slower than access from the memory closest to the CPU. For example, the memory access time on an Alliant FX/8 is 170 nanoseconds, whereas the sum of the average rotational latency and positioning times for the disk is 25.5 milliseconds.[2] A properly constructed SAP algorithm will result in fewer disk accesses.

We could apply the same strategy at a different level of the memory hierarchy by constructing an SAP algorithm that localizes spatial references to cache or local memory. The advantage of faster memory access is smaller at this level, usually offering a two to tenfold improvement. We remark that even though a memory access differential exists, an SAP algorithm isn't guaranteed to be an improvement over the old algorithm. The outer iteration count of the SAP method used in this thesis exceeds that of doing the inner iteration procedure at the global level. The differential in memory access time must breach some threshold value.

[1]See Appendix A.

[2]Hardware specifications for 10 1/2 inch Winchester disk drive used: 379 megabytes, data transfer rate 1.9 megabytes/sec, 842 cylinders, 20 tracks per cylinder, 44 sectors per track, 512 bytes per sector, average rotational latency 7.5 milliseconds and positioning times – 5.5 milliseconds track to track, 18 milliseconds average and 35 milliseconds maximum.

Chapter 2

Solving Elliptic Differential Equations

We consider a single second-order partial-differential equation (in two independent variables)

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + D(x, y, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}) = 0. \tag{2.1}$$

When the value of $B^2 - 4AC < 0$, the equation is classified as elliptic. Here u is a function from $\Re^2$ to $\Re$, which is twice differentiable.

2.1 Continuous Boundary Value Problem

Linear boundary value problems are denoted by

$$\begin{aligned} \mathcal{L}^{\Omega} U &= F^{\Omega}(x) \qquad x \in \Omega \\ \mathcal{L}^{\Gamma} U &= F^{\Gamma}(x) \qquad x \in \Gamma \equiv \partial\Omega. \end{aligned} \tag{2.2}$$

Here $x = \{x_1, \cdots, x_d\}$,with d the dimension of the space, and for our purposes $d = 2$. Ω is a given domain with boundary Γ. $\mathcal{L}^{\Omega}$ is a linear (elliptic) differential operator on Ω and $\mathcal{L}^{\Gamma}$ stands for one or several linear boundary operators. F^{Ω} denotes a given function on Ω and F^{Γ} one or several functions on Γ. We let $\mathbf{u}^*$ denote the solution to the continuous problem (2.2), the "exact" solution.

One of the simplest elliptic problems is Poisson's equation in the unit square,

$$\begin{cases} -\Delta U = F \\ U\,|_{\Gamma} = \phi \\ \Omega = \{(x,y) \mid 0 < x < 1, 0 < y < 1\}, \end{cases} \tag{2.3}$$

where F and ϕ are given. For the model problem of our computational studies in Chapter 5 we chose Poisson's equation with $F = 10\sin(3x + y)$ and $\phi = \sin(3x + y)$ on the unit square.

2.2 Discrete Boundary Value Problem

For discrete problems[1], we use the terminology *grid functions, grid operators* and *grid equations.* The discrete analog of (2.2) is denoted by

$$\begin{aligned} L_h^{\Omega} u_h &= f_h^{\Omega}(x) \qquad & x \in \Omega_h \\ L_h^{\Gamma} u_h &= f_h^{\Gamma}(x) \qquad & x \in \Gamma_h. \end{aligned} \tag{2.4}$$

Here h is a (formal) discretization parameter. The discrete solution to problem 2.4, which we denote $\mathbf{u}_h^*$, is a grid function defined on $\Omega_h \bigcup \Gamma_h$. f_h^{Ω} and f_h^{Γ} are discrete analogs of F^{Ω} and F^{Γ}. L_h^{Ω} and L_h^{Γ} are grid operators, i.e. mappings between spaces of grid functions. L_h^{Ω} is also called a *discrete* or *difference operator*, L_h^{Γ} a *discrete boundary operator.* For simplicity we assume that the discrete boundary equations are eliminated from (2.4). This is, for example, quite natural in the case of second order equations with Dirichlet boundary conditions. In this thesis, Ω is a rectangular domain and Ω_h a rectangular grid "matching well" with Ω. Ω_h is described by

$$\Omega_h \equiv \Omega \cap G_h. \tag{2.5}$$

where G_h denotes the infinite grid

$$G_h \equiv \{x = \kappa \cdot h \ : \ \kappa \in Z^2\}. \tag{2.6}$$

[1]Derived from [ST82a].

Here $\kappa \cdot h = (\kappa_1 h_x, \kappa_2 h_y)$. More specifically, the computational grid Ω_h for the discrete analog of (2.3) is defined by:

$$\Omega_h = \{(x_i, y_j) \mid x_i = ih_x,\ y_j = jh_y,\ 0 \le i \le n_x + 1,\ 0 \le j \le n_y + 1\ \} \tag{2.7}$$

where $h_x = 1/(n_x + 1)$, $h_y = 1/(n_y + 1)$, and n_x, n_y are integers which represent the number of unknowns in each direction. For simplicity we assume $n_h \equiv n_x = n_y$ and $h \equiv h_x = h_y$, a square, uniform mesh.

For the computational grid Ω_h a corresponding set of grid functions U_h and F_h are defined by $U_h = \{u_h : \Omega_h \to \Re\}$ and $F_h = \{f_h : \Omega_h \to \Re\}$. Let L_h be the linear operator $L_h : G(\Omega_h) \to G(\Omega_h)$, where $G(\Omega_h)$ denotes the linear space of grid functions on Ω_h.

Replacing the partial derivatives u_{xx} and u_{yy} in (2.3) by centered difference approximations at each grid point (x_i, y_j), we have the following corresponding second order finite difference equations:

$$\begin{gathered} -u_{i,j-1} - u_{i-1,j} + 4u_{i,j} - u_{i+1,j} - u_{i,j+1} = h^2 f_{i,j} \quad 1 \le i \le n_x \quad 1 \le j \le n_y \\ u_{i,0} = \phi_{i,0};\ u_{0,j} = \phi_{0,j};\ u_{i,n_y+1} = \phi_{i,n_y+1};\ u_{n_x+1,j} = \phi_{n_x+1,j}. \end{gathered} \tag{2.8}$$

Given these difference equations, we formulate a matrix problem with each row of the matrix L_h corresponding to a difference equation at one of the grid points, $u_{i,j}$. Let $n = n_x \times n_y$ denote the rank of L_h. The corresponding linear system of equations, the *grid equation*, is denoted

$$L_h u_h = f_h. \tag{2.9}$$

Finally, we define the n-vector whose components are the values of the continuous problem solution $\mathbf{u}^*$ at the grid point locations in Ω_h. Let

$$\mathbf{w}_h = \{\mathbf{w}_{i,j} = \mathbf{u}^*(x_i, y_j), \forall\ (x_i, x_j) \in \Omega_h\}. \tag{2.10}$$

2.3 Norms

In this section we define norms which we use throughout this thesis.

Definition 2.1 *The* l_2 **norm** *or* **Euclidean norm** *of any n-vector v is given by*

$$||v||_E \equiv \sqrt{v^T v} = \sqrt{\sum_{i=1}^{n} |v_i|^2}.$$

Definition 2.2 *The* **infinity** *or* **maximum norm** *of any n-vector v is defined by*

$$||v||_\infty \equiv \max(|v_i|), \quad 1 \le i \le n.$$

Definition 2.3 *When v is any n-vector arising from the 2-D discretization of a partial differential equation the* **discrete** L_2 **norm** *of v is given by*

$$||v||_h \equiv h\sqrt{\sum_{i=1}^{n} |v_i|^2}.$$

The maximum and discrete L_2 norms are related by the property $||v||_h \le \sqrt{h}||v||_\infty$.

Definition 2.4 *The* **spectral radius** *of a matrix A is defined by*

$$\rho(A) \equiv \sup\{\ |\lambda| : \lambda \textit{ an eigenvalue of } A\}.$$

Definition 2.5 *The* **spectral norm** *of a matrix A is defined by*

$$||A|| \equiv \sup_{||x||_E \neq 0} \frac{||Ax||_E}{||x||_E} = \sup_{||x||_h \neq 0} \frac{||Ax||_h}{||x||_h}.$$

Two properties of the spectral norm are

$$\begin{cases} ||A|| = & \sup\{\sqrt{\lambda} : \lambda \text{ an eigenvalue of } A^*A\}, \\ ||A|| = & \rho(A), \text{ if } A = A^*. \end{cases} \tag{2.11}$$

2.4 Error Definitions

In this section we define terms needed for a discussion of the error, convergence behavior and choice of stopping criterion.

Depending on the differencing scheme and grid selected, the accuracy is generally discussed in terms of powers of h, the grid spacing. The difference equation is derived from a Taylor series expansion. To arrive at the final representation the series is truncated after a certain number of terms. Therefore, the difference equation is an approximation. The magnitude of the truncated terms determines the accuracy of the approximation. Second order accuracy, or $O(h^2)$ accuracy, means that, for h sufficiently small, the magnitude of the truncated terms is bounded by $\beta \cdot h^2$, with β a constant.

The solutions of the continuous problem and the discretized problem differ at the grid point locations because the exact solution of the discrete problem $\mathbf{u}_h^*$ is only an approximation to the continuous solution $\mathbf{u}^*$.

Definition 2.6 *The* **local discretization (or truncation) error** *is defined to be the n-vector τ_h whose components are the amount by which the "exact" solution (continuous problem) $\mathbf{u}^*$ fails to satisfy the discrete equations, $\tau_h \equiv L_h\mathbf{w}_h - f_h = L_h\mathbf{u}^* - f_h$.*

The local discretization error is a measure of how well the discrete equations represent the continuous problem. However, this does not directly determine how well the discrete solution $\mathbf{u}_h^*$ approximates the continuous solution $\mathbf{u}^*$ at the grid points. [BM84]

Definition 2.7 *The* **global discretization (or truncation) error** *or the* **exact discretization error** [ST82b] *is defined to be the n-vector $\mathbf{e}_h$ whose components are the difference of the discrete problem solution $\mathbf{u}_h^*$ and the continuous problem solution $\mathbf{u}^*$ at grid point locations, $\mathbf{e}_h \equiv \mathbf{w}_h - \mathbf{u}_h^*$.*

The relationship between the global and local truncation error is given by

$$L_h\mathbf{e}_h = \tau_h. \tag{2.12}$$

Next we define the error in relation to u_h^k, the *dynamic approximation* to, or the *approximate solution* of, $\mathbf{u}_h^*$. The superscript k is the iteration index of an approximate solution generated by iterative method. Iterative methods are described in §2.6.

Definition 2.8 *The* **algebraic error** *is defined to be the n-vector ϵ_h^k whose components are the difference of the exact solution $\mathbf{u}_h^*$ of the discrete problem and its approximate solution u_h^k, $\epsilon_h^k \equiv \mathbf{u}_h^* - u_h^k$. It is also called the* **iteration error**.

Definition 2.9 *The* **residual** *or* **defect** *is defined to be the n-vector $\mathbf{r}_h^k$ whose components are the amount by which the approximate solution u_h^k fails to satisfy the discrete equations, $\mathbf{r}_h^k \equiv f_h - L_h u_h^k$.*

The *defect equation* is the relationship between the defect and the algebraic error

$$L_h \epsilon_h^k = \mathbf{r}_h^k. \tag{2.13}$$

The magnitude of the residual at a grid point is a measure of how closely the approximate solution at the grid point and at its neighboring grid points approximates the discretization of the partial differential equation at the grid point.

The error of the solution process is the sum of the global and algebraic errors

$$\mathbf{w}_h - u_h^k = \underbrace{(\mathbf{w}_h - \mathbf{u}_h^*)}_{\text{global error}} + \underbrace{(\mathbf{u}_h^* - u_h^k)}_{\text{algebraic error}}. \tag{2.14}$$

The global error in (2.14) is controlled by the size of h. In the case of a second order accurate discretization, for small enough h decreasing the grid spacing by a factor of two will result in a decrease of the L_2-norm of the exact discretization error by a factor of four. This property becomes useful when verifying the result of a computer code.

Definition 2.10 *The* **correction** *is defined to be the n-vector whose components are the difference of two consecutive approximate solutions, $u_h^k - u_h^{k-1}$.*

Definition 2.11 *Let h denote a fine grid and H denote a coarser grid. The* **relative truncation error** *is defined to be the n-vector τ_h^k whose components are obtained by subtracting the defect on grid h restricted by the operator I_h^H to grid H, from the n-vector found by operating on the injection of u_h (by the operator $\hat{I}_h^H$) to grid H, with L_H, a coarse grid L:*

$$\tau_h^k \equiv L_H(\hat{I}_h^H u_h) - I_h^H(L_h u_h - f_h).$$

Note, the definition of the relative truncation error includes a number of terms that haven't been defined yet. These terms are defined in Chapter 3. τ_h^k is that quantity which has to be added to the right hand side If_h to obtain the values of the fine-grid solution $\mathbf{u}_h^*$ (on $\Omega^{\ell-1}$) by solving the coarse-grid equation. Another measure of error compares the iterate u_h^k to the "exact" solution $\mathbf{w}_h$ at grid point locations.

Definition 2.12 *The* **root mean square error** *(RMS) is defined to be the Euclidean norm of the difference between the discrete problem iterate u_h^k and the solution to the continuous problem $\mathbf{u}^*$ at grid point locations,*

$$RMS \equiv \left\|u_h^k - \mathbf{w}_h\right\|_E \quad or \quad RMS \equiv \left\|\epsilon_h^k - \mathbf{e}_h\right\|_E. \tag{2.15}$$

2.5 Matrix Properties and Concepts

The definitions and theorems of this section are taken from [Var62].

Definition 2.13 *A matrix A is defined to be* **non-negative**, *denoted $A \geq 0$, provided $a_{i,j} \geq 0$ for each element $(a_{i,j})$ of A.*

Definition 2.14 *A directed graph is* **strongly connected** *if, for any ordered pair of nodes P_i and P_j there exists a directed path $\overrightarrow{P_iP_{l_1}}, \overrightarrow{P_{l_1}P_{l_2}}, \cdots, \overrightarrow{P_{l_{r-1}}P_{l_r}}$ connecting P_i to P_j.*

Theorem 2.1 *An $\mathrm{n} \times \mathrm{n}$ complex matrix A is* **irreducible** *if and only if its directed graph $G(A)$ is strongly connected.*

Definition 2.15 *An* n × n *complex matrix* $A = (a_{i,j})$ *is* **diagonally dominant** *if*

$$|a_{i,i}| \geq \sum_{\substack{j=1;\\ j\neq i}}^{n} |a_{i,j}|, \qquad 1 \leq i \leq n. \tag{2.16}$$

Definition 2.16 *An* n × n *complex matrix* $A = (a_{i,j})$ *is* **irreducibly diagonally dominant** *if it is both irreducible and diagonally dominant.*

Definition 2.17 *An* n × n *matrix* $A = (a_{i,j})$ *is* **strictly diagonally dominant** *if strict inequality in (2.16) is valid for all* $1 \leq i \leq n$.

Theorem 2.2 *Let* $A = (a_{i,j})$ *be an* n × n *strictly or irreducibly diagonally dominant complex matrix. Then the matrix* A *is nonsingular. If all the diagonal entries of* A *are in addition positive real numbers, then the eigenvalues* λ_i *of* A *satisfy*

$$Re\,\lambda_i > 0, \qquad 1 \leq i \leq n.$$

Corollary. *If* A *is a Hermitian* n×n *strictly diagonally dominant or irreducibly diagonally dominant matrix with positive diagonal entries, then* A *is* **positive definite**.

Definition 2.18 *Consider expressing the* n × n *matrix* A *in the form* $A = M - N$, *where* M *and* N *are also* n × n *matrices. If* M *is nonsingular, we say that this expression represents a* **splitting** *of the matrix* A.

Definition 2.19 *For* n × n *real matrices,* A, M, *and* N, $A = M - N$ *is a* **regular splitting** *of the matrix* A *if* M *is nonsingular with* $M^{-1} \geq 0$, *and* $N \geq 0$.

Definition 2.20 *A real* n × n *matrix* $A = (a_{i,j})$ *with* $a_{i,j} \leq 0$ *for all* $i \neq j$ *is an* **M-matrix** *if* A *is nonsingular, and* $A^{-1} \geq 0$.

Many matrices arising from the discretization of elliptic partial differential equations are known to be M-matrices, including those arising from the discretization of Laplace's equation on a rectangle.

Theorem 2.3 *Let* A *be an* n × n M*-matrix, and let* C *be any matrix obtained from* A *by setting certain off-diagonal entries of the matrix* A *to zero. Then,* C *is also an* M*-matrix.*

2.6 Iterative Methods

A variety of methods exists for solving linear systems such as $L_h u_h = f_h$. To solve this linear system where L_h is a given nonsingular $n \times n$ matrix, we consider expressing the matrix L_h in the form $L_h = M - N$, where M and N are also $n \times n$ matrices. Associated with this splitting is an iterative method. Substituting the matrix splitting for L_h, adding Nu_h to each side and then superscripting u_h with the iteration index k results in the following general form for an iterative equation:

$$Mu_h^{k+1} = Nu_h^k + f_h, \qquad k \geq 0, \tag{2.17}$$

where u_h^0 is the initial guess. If M and N are chosen properly, at each iteration the approximate solution u_h^{k+1} approaches a limiting value. When

$$\lim_{k \to \infty} \left\| u_h^k - \hat{u} \right\| = 0$$

for some vector norm the corresponding iterative method is called convergent and $\hat{u}$ is a fixed point of $L_h u_h = f_h$. One iteration of the method is commonly called a relaxation sweep. However, in the context of the multigrid algorithm (the inner iteration procedure of choice for our numerical experiments), one refers to the relaxation process as *smoothing* and from here on we use this term.

We arrive at an alternative form of the general iterative equation (2.17) by multiplying both sides by M^{-1} giving

$$u_h^{k+1} = M^{-1}Nu_h^k + M^{-1}f_h, \tag{2.18}$$

provided M is invertible. One is assured M is nonsingular by considering only positive definite M. In this form of an iterative equation the multiplier of u_h^k, in this case $M^{-1}N$, is named the *amplification matrix* (of the error) or the *iteration matrix.* In the case where M and N are derived from a simple splitting of L_h, we also refer to $M^{-1}N$ as the smoothing matrix S. In more complicated iterative methods the amplification matrix may be the product of a number of different matrices, for example, the amplification matrix described in equation 3.7.

Theorem 2.4 *If $A = M - N$ is a regular splitting of the matrix A and $A^{-1} \geq 0$, then*

$$\rho(M^{-1}N) = \frac{\rho(A^{-1}N)}{1+\rho(A^{-1}N)} < 1.$$

Thus, the matrix $M^{-1}N$ is convergent, and the iterative method

$$Mu_h^{k+1} = Nu_h^k + f_h, \qquad k \geq 0,$$

converges for any initial vector u^0. (Theorem 3.13 in [Var62]).

Generally the splitting used, $L_h = M - N$, is a regular splitting. One chooses a splitting such that M^{-1}, the *approximate inverse* of L_h, is easily invertible. One could solve $L_h u_h = f_h$ directly by computing the LU factorization of L_h and backsolving but the computation takes $O(n^3)$ operations, where n is the number of equations in the linear system. In addition to its large computational expense, a direct solve requires a prohibitive amount of memory for large systems. Generally, the difference operators that arise from discretization of partial differential equations result in block banded matrices. Even though the original matrix L_h is sparse, substantial fill-in occurs when computing the its LU factors. In practice, one rarely computes and stores M^{-1}, instead, one stores vectors corresponding to the bands of M and N.

The spectral norm of the amplification matrix $M^{-1}N$ is useful in describing the error behavior of iterative methods. The norm of the algebraic error at the k-th step of an iterative method written in the form of equation (2.18) satisfies

$$\left\|\epsilon^{k+1}\right\|_E \leq \left\|M^{-1}N\right\| \left\|\epsilon^k\right\|_E = \rho(M^{-1}N)\left\|\epsilon^k\right\|_E$$

provided $M^{-1}N$ is a symmetric matrix. $\rho(M^{-1}N)$ is called the *rate of convergence.* The spectral radius quantifies the amplification properties of the amplification matrix on the error. A maximum eigenvalue of less than one for the amplification matrix assures the iterative method reduces the norm of the error during each iteration. The smaller the spectral radius the greater the rate at which the error is reduced. For ordinary iterative methods, where the amplification matrix is the smoothing matrix, the spectral radius of the smoothing matrix must be less than one if the method is to converge.

Some common splittings arising from the discretization of partial differential equations are Jacobi, Gauss-Seidel and SOR, including their point, line and block variants. Theorems and knowledge about the spectral radius of different classes of matrices aid in choosing how to split the matrix L_h. Varga [Var62] gives a detailed analysis of these and other iterative methods. Elsewhere, [Bra77], [Mol81b] and [Ket82] have characterized these and other smoothing methods using local mode analysis. Appendix B, an excerpt from [Bra77], details some of this basic local mode analysis. Example illustrations of matrices that compose the splitting associated with an iterative method are found in Appendix D.

2.6.1 Defect Correction Formulation

If we make the substitution $N = M - L$ in (2.18) we arrive at yet another form of the general iterative method,

$$\begin{aligned} u_h^{k+1} &= M^{-1}(M-L)\,u_h^k + M^{-1}f_h, \\ &= u_h^k - M^{-1}L\,u_h^k + M^{-1}f_h, \end{aligned}$$

$$u_h^{k+1} = u_h^k + M^{-1}(f_h - Lu_h^k). \tag{2.19}$$

We call an iterative method written in this form (2.19) a *defect correction* method, since the latest iterate u_h^{k+1} is the previous iterate u_h^k corrected by the approximate inverse times the defect,

$$u_h^{k+1} = u_h^k + M^{-1}\mathbf{r}_h^k. \tag{2.20}$$

Recall, the defect is related to the algebraic error by the defect equation $L_h\epsilon_h^k = \mathbf{r}_h^k$. Solving the defect equation is equivalent to solving the original linear system $L_hu_h = f_h$, since $\mathbf{u}_h^* = u_h^k + \epsilon_h^k$. Therefore we may solve the original linear system (2.9) by working with the defect equation instead. The inner iteration procedure used in this thesis is a defect correction method.

Example. Let L_h be decomposed as $L_h = D - U - V$ where D is diagonal, U is strictly upper triangular, and V is strictly lower triangular. The Gauss-Seidel iteration is

$$(D - V)\,u_h^{k+1} = Uu_h^k + f,$$

or

$$u_h^{k+1} = (D-V)^{-1}Uu_h^k + (D-V)^{-1}f.$$

This is a defect correction process with $S = (D-V)^{-1}U$ and approximate inverse $(D-V)^{-1}$,

$$u_h^{k+1} = u_h^k + (D-V)^{-1}(f_h - L_h u_h^k).$$

2.7 Convergence

The end result of a direct method is a final, unique solution. In an iterative method however, each iterate offers another approximate solution. As an iterative method works its way to completion, u_h^k should be approaching $\mathbf{u}_h^*$, but when is the iteration process complete? We need to judge the accuracy of the iterative solution and determine a stopping point.

The general objective in applying an iterative method to a system of linear equations is to compute a vector u_h^k that approximates the exact solution $\mathbf{u}^*$ at the grid points (x_i, y_j) to within a prescribed accuracy $\varepsilon > 0$. More precisely, choosing the norm $\|\cdot\|_h$, for instance, we want to find u_h^k so that

$$\left\|u_h^k - \mathbf{w}_h\right\|_h \leq \varepsilon. \tag{2.21}$$

Since the immediate problem is to approximate $\mathbf{u}_h^*$, we will satisfy (2.21) if

$$\left\|u_h^k - \mathbf{u}_h^*\right\|_h + \|\mathbf{u}_h^* - \mathbf{w}_h\|_h \leq \varepsilon. \tag{2.22}$$

This says we want the sums of the norms of the algebraic error and the global error to be smaller than the prescribed tolerance. The first term, the norm of algebraic error in (2.22), is controlled by the number of iterations, becoming smaller as we perform more iterations. The second term, the norm of global erroris controlled by the size of h. "In general it seems best to roughly balance these errors: why go to extreme measures to reduce the algebraic error, when the global error is comparatively large; or conversely, why have a poor approximation to a very accurate discrete solution? Thus, we will attempt to satisfy (2.21) by way of the conditions" [BM84]

$$\left\|u_h^k - \mathbf{u}_h^*\right\|_h \leq \frac{\varepsilon}{2}, \quad \text{and} \quad \|\mathbf{u}_h^* - \mathbf{w}_h\|_h \leq \frac{\varepsilon}{2}. \tag{2.23}$$

2.7.1 Example Behavior of Norms

Table 2.1 demonstrates the typical behavior of four norms monitored while solving the model problem on a grid with 128 x 128 unknowns. The columns of Table 2.1 show, from left to right, the iteration index,[2] the maximum norm of the residual vector, the discrete L_2 norm of the residual, the RMS error, and the maximum norm of the vector $(u_h^k - \mathbf{w}_h)$.

k	$\left\|\mathbf{r}_h^k\right\|_\infty$	$\left\|\mathbf{r}_h^k\right\|_h$	$\left\|(u_h^k - \mathbf{w}_h)\right\|_E$	$\left\|(u_h^k - \mathbf{w}_h)\right\|_\infty$
0	$3.35 \times 10^{+2}$	$1.57 \times 10^{+1}$	8.35×10^{-2}	1.97×10^{-1}
1	$2.96 \times 10^{+1}$	$2.00 \times 10^{+0}$	3.40×10^{-3}	1.10×10^{-2}
2	$2.09 \times 10^{+0}$	1.25×10^{-1}	1.69×10^{-4}	6.10×10^{-4}
3	1.38×10^{-1}	7.48×10^{-3}	1.58×10^{-5}	4.19×10^{-5}
4	8.75×10^{-3}	4.56×10^{-4}	1.20×10^{-5}	2.32×10^{-5}
5	5.45×10^{-4}	2.77×10^{-5}	1.22×10^{-5}	2.34×10^{-5}
6	3.40×10^{-5}	1.72×10^{-6}	1.22×10^{-5}	2.34×10^{-5}

Table 2.1: Norms generated solving model problem.

k	$\left\|\mathbf{r}_h^{k+1}\right\|_\infty / \left\|\mathbf{r}_h^k\right\|_\infty$	$\left\|\mathbf{r}_h^{k+1}\right\|_h / \left\|\mathbf{r}_h^k\right\|_h$	$\dfrac{\left\|u_h^k - \mathbf{w}_h\right\|_E}{\left\|u_h^{k-1} - \mathbf{w}_h\right\|_E}$	$\dfrac{\left\|u_h^k - \mathbf{w}_h\right\|_\infty}{\left\|u_h^{k-1} - \mathbf{w}_h\right\|_\infty}$
1	0.0885	0.1269	0.0408	0.0559
2	0.0704	0.0627	0.0498	0.0553
3	0.0661	0.0597	0.0931	0.0688
4	0.0634	0.0610	0.7575	0.5519
5	0.0622	0.0607	1.0206	1.0095
6	0.0624	0.0622	1.0002	1.0004

Table 2.2: Convergence ratios for norms in Table 2.1.

[2]Each iteration corresponds to one multigrid V cycle (see page 33) done on the global domain.

Note, in columns four and five of Table 2.1 the values of the norms charted remain virtually constant from iterate 4 to iterate 6; whereas, the values of the norms charted in columns two and three of Table 2.1 keep decreasing at an approximately constant rate for iterates 4 through 6, see Table 2.2. The behavior of the norms in columns four and five indicates that the approximate solution is not getting any closer to $\mathbf{u}^*$ in the sense of those norms. Note, the exact discretization error for this problem is 2.92×10^{-4}, and the iterate at which the ratio of successive RMS error values exceeds one is the same iterate at which the discrete L_2 norm of the residual becomes less than the norm of the exact discretization error, iterate 5.

2.7.2 Stopping Criteria

Let $E_{i,j}$ be some pointwise measure of the error at the grid point $(x_i, y_j) \in \Omega_G$. Let E^k be the n-vector whose components are $E_{i,j}$ for iterate k. For instance, $E_{i,j}$ could be the value of

(1) $\mathbf{r}_h^k$, the residual,

(2) $u_h^k - u_h^{k-1}$, the correction,

(3) τ_h^k, the truncation error estimate, or

(4) $u_h^k - \mathbf{w}_h$, the error of the solution process,

at the grid point (x_i, y_j). The method of determining convergence is to track some measure(s) of the error and stop when that measure of the error is comparatively small. One takes some norm of the error(s) so that the error information is compressed into only one or possibly two numbers. Consider stopping criteria of the form

(1) $\left\|E^k\right\|_\square < \varepsilon$,

(2) $\left\|E^k\right\|_\square < \varepsilon \left\|E^0\right\|_\square$,

(3) $\left\|E_A^k\right\|_\square < \left\|E_B^k\right\|_\square$,

where $\|\cdot\|_\square$ is some norm of choice and ε is a constant.

In stopping criteria 1 and 2 above, ε determines when the error is relatively small. Stopping criteria 1 and 2 are a common convergence test. Method 1 is the typical method of iterating until some norm of E^k, frequently $\left\|\mathbf{r}_h^k\right\|_h$, is less than the arbitrarily chosen tolerance ε, say 10^{-6}. Method 2 is the equally common method of iterating until some norm of E^k, again frequently $\left\|\mathbf{r}_h^k\right\|_h$, has been decreased by a certain order of magnitude. For instance, if the initial value of the norm was 10^2 and $\varepsilon = 10^{-6}$, iterate until the value of the norm is six orders of magnitudes less, 10^{-4}. Unfortunately, in these methods no theoretical basis exists for choosing the value of ε, so its value is chosen based on empirical experience.

In another convergence test the stopping criterion is the time at which the correction meets the relationship $\left|u_h^k - u_h^{k-1}\right| \leq \varepsilon$ for every grid point. This is a stopping criterion of the form of method 1, with $\|\cdot\|_\square = \|\cdot\|_\infty$. Again, the choice of ε has no theoretical basis. A potential drawback of this method is that it may require storing an additional iterate. The logic behind this method is that when the change from one iteration to the next is small at every point there is no need to continue iterating.

In method 3 the subscripts A and B are meant to distinguish between two different measures of error. Let us restate the convergence conditions of (2.23) in the format of method 3,

$$\left\|u_h^k - \mathbf{u}_h^*\right\|_h < \|\mathbf{u}_h^* - \mathbf{w}_h\|_h . \tag{2.24}$$

with $E_A^k = u_h^k - \mathbf{u}_h^*$, the algebraic error, symbolized by ϵ_h^k and $E_B = \mathbf{u}_h^* - \mathbf{w}_h$, the global error, symbolized by $\mathbf{e}_h$. The iteration index of E_B is dropped because the global error doesn't change. This stopping criterion is useful for theoretical studies of convergence only; as it includes both the exact solution and the solution to the discrete problem.

Finally, we mention a stopping criterion in the form of method 3 which we can compute for problems that we do not know the analytical solution. The fact that this stopping criterion has a theoretical basis for its existence makes it desirable and more meaningful. The stopping criterion is

$$\left\|\mathbf{r}_h^k\right\|_h < \frac{4}{3}\left\|\tau_h^k\right\|_h . \tag{2.25}$$

Under reasonable smoothness assumptions, and a second order accurate difference approximation,

$$\frac{4}{3}\tau_h^k = \tau_h + O(h^2). \tag{2.26}$$

An estimate of the local truncation error, τ_h^k -- the relative truncation error, can be generated using Richardson's extrapolation [Hac85]. This stopping criterion states that the iteration process concludes when the norm of the residual error is less than the estimate of the local truncation error.

The RMS Ratio Stopping Criterion

We define one more stopping criterion which we use in Chapter 5. We base this stopping criterion on the RMS error, the Euclidean norm of the difference between the finite difference iterate and $\mathbf{u}^*$ evaluated at grid point locations, $\mathbf{w}_h$.

Definition 2.21 *We say the k-th iterate meets* **RMS ratio** *stopping criterion when*

$$\left\|u_h^{k-1} - \mathbf{w}_h\right\|_E \leq \left\|u_h^k - \mathbf{w}_h\right\|_E . \tag{2.27}$$

See Table 2.1 for an example of the behavior of the RMS error — second column from the right, and the same column of Table 2.2 for an example of the behavior of the ratio of successive RMS error values. The RMS error decreases to a minimum and then rises a small amount away, remaining almost constant from then on. When u_h^k is relatively far from $\mathbf{u}^*$ the RMS error experiences a decrease similar to the decrease experienced by the discrete L_2 norm of the residual and the maximum norm of the residual. When the iterative solution gets close to $\mathbf{u}_h^*$ the RMS error approaches a constant and the ratio of successive RMS error values becomes close to one.

This RMS ratio stopping criterion marks the time at which the ratio of the RMS error from one iteration to the next becomes greater or equal to 1.0. There is no guarantee the ratio will reach a value of 1.0. In a slowly converging problem the ratio might be greater than .99 but less then 1.0 for more iterations than we would care to monitor. For example, for some domain decompositions with small overlap, the ratio approached one very closely, but did not reach a maximum (local) in a reasonable time frame. When this stopping criterion is met, one might be led to believe that the iterative solution u_h^{k-1} was a more accurate than the iterative solution u_h^k. This is reasonable to believe since the nature of a convergent iterative method is to improve the solution with each succeeding iteration. However, this is not always the case. In the example problem, when we reach an inflection point of the ratio of solution norms we could say such that our solution has become as accurate as it is going to be. However, we might be mistaken. In chaotic variants of the SAP-MG method, described in a later chapter, the ratios of the RMS error from one iterate to the next are not even necessarily monotonic decreasing.

2.7.3 Comparing $\|\cdot\|_h$ and $\|\cdot\|_\infty$

For instance, suppose we have a 100×100 grid, 10000 unknowns. If the residual at each grid point is 1.0×10^{-5} except at ten points where the residual is 1.0×10^{-3}, then the sum of squares of the residuals is .000010999. The contribution from the set of 10 points is 10 times greater than that from the set of 9990 points. $\left\|\mathbf{r}_h^k\right\|_h$ in this case is 3.3×10^{-5}, assuming $h = .001$. If the residual at the ten points was 1.0×10^{-3} also, then $\left\|\mathbf{r}_h^k\right\|_h$ would be 1.0×10^{-5}. With $\left\|\mathbf{r}_h^k\right\|_h$, even though at ten points the residual is two orders of magnitude greater than the residuals at the other 9990 points, the value of the norm is of the same order as the norm of a problem where all 10000 grid point residuals had a value of 1.0×10^{-5}. With the $\|\cdot\|_\infty$ norm, this would not be the case. Those ten points alone would determine the convergence. The behavior of the $\|\cdot\|_h$ is such that if there are only a few problem points, overlook them. The behavior of $\|\cdot\|_\infty$ is such that if there is even one problem point, keep on solving.

2.8 Dynamic Computation of Stopping Criteria

Consider a domain decomposition of the n point grid Ω_G into m not necessarily disjoint subdomains $\Omega_1, \Omega_2, \cdots, \Omega_m$. Define the subdomain index q. Associated with every subdomain Ω_q is the set of its boundary points $\partial\Omega$. Let $E_{i,j}$ be some pointwise measure of the error at the grid point $(x_i, y_j) \in \Omega_G$. Let E_{Ω_G}, or E^k, be the n-vector whose components $E_{i,j}$ are some pointwise measure of the error at grid points in Ω_G. Let E_{Ω_q} be the n-vector whose components $E_{i,j}$ are some pointwise measure of the error at grid points $(x_i, y_j) \in \Omega_q$, and 0 for grid points $(x_i, y_j) \notin \Omega_q$. For instance, $E_{i,j}$ could be the value of

(1) $\mathbf{r}_h^k$, the residual,

(2) $u_h^k - u_h^{k-1}$, the correction,

(3) τ_h^k, the truncation error estimate, or

(4) $u_h^k - \mathbf{w}_h$, the error of the solution process,

at the grid point (x_i, y_j).

When using a dynamically computable stopping criterion, after the solution of any subdomain is advanced the stopping criterion for Ω_G can be computed. Dynamically computing the stopping criterion as a Schwarz algorithm visits each of its subdomain minimizes the locality of reference a program exhibits. It does so by avoiding the sweep through the iterate and its discrete operator or the previous iterate that would have been necessary to compute the stopping criterion on Ω_G after completing the outer iteration.

Consider stopping criteria of the form

(1) $\left\|E^k\right\|_\square < \varepsilon$,

(2) $\left\|E^k\right\|_\square < \varepsilon \left\|E^0\right\|_\square$,

(3) $\left\|E_A^k\right\|_\square < \left\|E_B^k\right\|_\square$,

where $\|\cdot\|_\square$ is some norm of choice. If $\|\cdot\|_\square$ is dynamically computable we say these stopping criteria are dynamically computable also. In the next two sections we demonstrate how to dynamically compute the $\|\cdot\|_h$ and $\|\cdot\|_\infty$ norms.

2.8.1 Dynamic/Distributed Computation of $\|\cdot\|_h$

Recall that the $\|\cdot\|_h$ of an n-vector v on Ω_G is given by

$$\|v\|_h \equiv h\sqrt{\sum_{i=1}^{n} |v_i|^2}.$$

We define two quantities which allow us to support the dynamic computation of $\|E_{\Omega_G}\|_h$ during an outer iteration of the SAP. Define the m-vector $\overline{S}_{\Omega_q}$ whose components S_{Ω_q} are the sums of squares of components of E_{Ω_q}:

$$\overline{S}_{\Omega_q} \equiv \sum_{\substack{i,j \\ (x_i,y_j)\in\Omega_q}} |E_{i,j}|^2.$$

Similarly, define S_{Ω_G}, a scalar, for the global domain:

$$S_{\Omega_G} \equiv \sum_{\substack{i,j \\ (x_i,y_j)\in\Omega_G}} |E_{i,j}|^2.$$

Given S_{Ω_G}, $\|E_{\Omega_G}\|_h$ immediately follows as $\|E_{\Omega_G}\|_h = h\sqrt{S_{\Omega_G}}$. The starting value of the quantity S_{Ω_G} is computed after setting the solution vector of Ω_G to the initial guess (unless E_{Ω_G} is the correction).

In our method for dynamically computing the stopping criterion for Ω_G, we do not compute $\left\|E^k\right\|_h$ for the global domain Ω_G after each outer iteration as iterative methods typically do. Instead, we use the following procedure for subdomains Ω_1, Ω_2, $\cdots$, Ω_m as each subdomain is visited during the SAP. On entering Ω_q, we compute S_{Ω_q}, naming it $S_{\Omega_q}^{enter}$; then we complete the inner iteration procedure on Ω_q; but, before exiting Ω_q, we again compute S_{Ω_q}, naming it $S_{\Omega_q}^{exit}$. Finally, we modify S_{Ω_G} by the difference between $S_{\Omega_q}^{exit}$, the new sum of squares of the components of E_{Ω_q} and $S_{\Omega_q}^{enter}$, the old sum of squares of the components of E_{Ω_q}; i.e., $S_{\Omega_G}^{new} = S_{\Omega_G}^{old} + (S_{\Omega_q}^{exit} - S_{\Omega_q}^{enter})$. Therefore, after advancing the solution on any subdomain, $\|E_{\Omega_G}\|_h$ is known and any test for convergence on Ω_G involving $\|E_{\Omega_G}\|_h$ can be computed.

Note, let R_G be the set $\{|E_{i,j}|^2 : \forall\, i,j \text{ such that } (x_i,y_j) \in \Omega_G\}$ and R_q be the set $\{|E_{i,j}|^2 : \forall\, i,j \text{ such that } (x_i,y_j) \in \Omega_q\}$, then $R_q \subseteq R_G\ \forall\, q$. If for any (q,q') pair $\Omega_q \bigcap \Omega_{q'} \neq \emptyset$ then $R_q \bigcap R_{q'} \neq \emptyset$.

2.8.2 Dynamic/Distributed Computation of $\|\cdot\|_\infty$

Recall that the $\|\cdot\|_\infty$ of a n-vector v on Ω_G with grid spacing h is given by

$$\|v\|_\infty \equiv \max(|v_i|), \quad 1 \leq i \leq n.$$

Again we define two quantities which allow us to support the dynamic computation of $\|E_{\Omega_G}\|_\infty$ during an outer iteration of the SAP. Define the m-vector $\overline{C}_q$ whose components C_{Ω_q} are the maximum absolute values of components of E_{Ω_q},

$$\overline{C}_q \equiv \{\left\|E_{\Omega_q}\right\|_\infty, \ 1 \leq q \leq m\}.$$

Similarly, define C_{Ω_G}, a scalar, for the global domain, $C_{\Omega_G} \equiv \|E_{\Omega_G}\|_\infty$. The starting value of the quantity C_{Ω_G} is computed after setting the solution vector of Ω_G to the initial guess (unless E_{Ω_G} is the correction which requires u_h^0 and u_h^1).

In our method for dynamically computing the stopping criterion for Ω_G, we do not compute $\left\|E^k\right\|_\infty$ for the global domain Ω_G after each outer iteration as iterative methods typically do. Instead, we use the following procedure for subdomains Ω_1, Ω_2, $\cdots$, Ω_m as each subdomain is visited during the SAP. During the visit to each subdomain Ω_q, C_{Ω_q} is computed. After the inner iteration procedure on Ω_q is complete, set $C_{\Omega_G} = \left\|C_{\Omega_G}, C_{\Omega_q}\right\|_\infty$. Therefore, after advancing the solution on any subdomain, $\|E_{\Omega_G}\|_\infty$ is known and any test for convergence on Ω_G involving $\|E_{\Omega_G}\|_\infty$ can be computed.

Chapter 3

The Multigrid Method

In §2.6 we mentioned several common iterative methods for solving elliptic equations. Fourier analysis of these relaxation or smoothing methods, shows they efficiently remove the high frequency error components in the approximate solution. The notion of iterative methods as smoothers is crucial in the discussion of multigrid. See Appendix B, an excerpt from [Bra77], for details. Two to three iterations eliminate most of the high frequency error components [Bra77]. Nonetheless the asymptotic behavior of these methods, quantified by their convergence rates, was shown to be slow because they ineffectively smoothed the low frequency error components [Fed62], [Bak66], [Bra77]. The multigrid method addresses the problem of managing these low frequency errors by attacking them on coarser grid levels. In the Schwarz method presented in this thesis we will use the multigrid method for the inner iteration procedure. Notes about the developers and history of multigrid follow at the end of this chapter.

The multigrid method uses a multilevel structure to remove the low frequency components in the error that a smoothing routine is inefficient at removing. It does this by smoothing the low frequency error components on coarser grids. The relatively high frequencies on the coarser grid levels which are effectively smoothed correspond to low error frequencies on the finer grid levels.

Multigrid is an iterative, multilevel defect-correction method which can be used for solving symmetric, positive definite, linear systems. In this thesis we refer to the finest grid as the uppermost grid level. *Cycle* is a synonym for a multigrid iteration. During one cycle the multigrid method travels from the finest grid down through coarser grid levels to a coarsest grid and then back up to the finest grid, visiting each of the intermediate grids along the way. The manner in which the method travels from one grid level to the next defines the type of multigrid cycle. Figure 3.1 shows the two most common multigrid cycling procedures, V and W, along with the full multigrid iteration. Full multigrid, FMG, also called nested iteration, consists of one of the other types of cycles, prefaced by the bootstrapping of an initial guess from the coarsest grid to the finest grid.

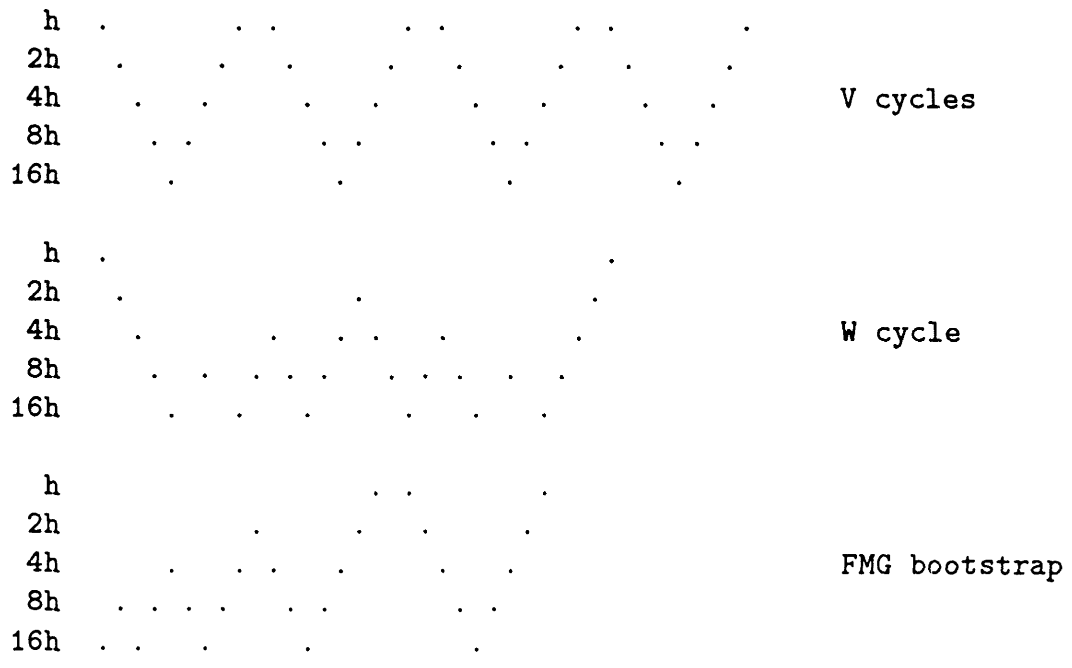


Figure 3.1: Grid Schedules for Common Multigrid Cycles

Smoothing is typically done at each grid level. The smoothing iteration that occurs before moving to a coarser grid is called *pre*-smoothing. The smoothing iteration that occurs before moving to a finer grid is called *post*-smoothing. The purpose of pre-smoothing is to smooth (damp) the relative high frequency error on the respective grid levels. The purpose of post-smoothing is to eliminate high frequency error components introduced during the interpolation and addition of the defect to the solution residing at the next higher level.

3.1 Two-grid Algorithm

We begin explaining the details of the multigrid method by considering a two-level algorithm. Our explanation follows the discussion given by Mol [Mol81a]. The two-grid method is a non-stationary defect correction process in which two different approximate inverses are used:

- Some smoothing method (e.g. Jacobi, Gauss-Seidel) on the fine grid which damps short wavelength fluctuations in the residual.
- A coarse grid correction which damps the long wavelength fluctuations in the residual.

Recall, for the discrete analog (2.3) of the linear boundary value problem we defined a computational grid Ω_h and a corresponding set of grid functions U_h

$$\Omega_h = \{(x_1, x_2) \mid x_i = mh,\ 0 \leq m \leq n_h + 1,\ i = 1,2\ \} \tag{3.1}$$

$$U_h = \{u_h : \Omega_h \rightarrow \Re\} \tag{3.2}$$

where $h = 1/(n_h + 1)$, n_h is an integer, and $n \equiv n_h \times n_h$. The discretized form of the differential equation resulted in the the linear system of equations denoted by

$$L_h u_h = f_h, \tag{3.3}$$

with L_h an $n \times n$ non-singular matrix and with u_h and f_h n-vectors.

The two-grid method uses an analog of (3.1) and (3.2) on a coarser grid $\Omega_{h'}$, that is, $n' < n$, with mesh size $h' > h$:

To avoid confusion about which grid level we are referring to we assign the index ℓ to the finer grid level and $\ell - 1$ to the coarser grid level. (For future reference $\ell = 1$ will index the coarsest grid and $\ell = M$ will index the finest grid.) Using this notation, system (3.3) is now denoted by

$$L^\ell u^\ell = f^\ell, \tag{3.4}$$

with $L^\ell : U^\ell \to U^\ell$. Its coarse grid analog $\Omega_{\ell-1}$ is now denoted

$$L^{\ell-1}u^{\ell-1} = f^{\ell-1}. \tag{3.5}$$

with $L^{\ell-1} : U^{\ell-1} \to U^{\ell-1}$.

Next we must define a function which maps the fine grid functions to the coarse grid functions and visa versa. We call the act of transferring the defect equation from the finer grid Ω_h to the coarser grid $\Omega_{h'}$ *restriction.* We denote the *restriction matrix* or *restriction operator* by $I_\ell^{\ell-1}$,

$$I_\ell^{\ell-1} : U^\ell \to U^{\ell-1}.$$

The solution to the defect equation on grid level $\ell - 1$ is a correction which we will interpolate and add to the grid solution of the problem on grid level ℓ. We call the act of transferring the correction from the coarser grid $\Omega_{h'}$ to the finer grid Ω_h *prolongation.* The *prolongation matrix* or *prolongation operator* denoted $I_{\ell-1}^\ell$ interpolates the correction formed on the coarser grid level $\ell - 1$ to grid level ℓ,

$$I_{\ell-1}^\ell : U^{\ell-1} \to U^\ell.$$

We can now define the coarse grid correction step of the two-grid method:

$$u^\ell \equiv u^\ell + I_{\ell-1}^\ell \,(L^{(l-1)})^{-1}\, I_\ell^{\ell-1}\,(f^\ell - L^\ell u^\ell).$$

In the two-grid method the coarse grid problem (3.5) is solved directly.

Finally, combining the smoothing sweeps with the coarse grid correction, we see one step in the two-grid method consists of ν_{pre} sweeps with the smoothing method, a coarse grid correction step and ν_{post} sweeps with the smoothing method. A pseudocode description of the two-grid method appears in Table 3.1. On level ℓ denote the pre-smoothing operator by $\hat{S}^\ell$ and the post-smoothing operator by S^ℓ. Denote the approximate inverse of the smoother on level ℓ, M^{-1} of (2.17), by B^ℓ. Take $B^\ell = (I - S^\ell)(L^\ell)^{-1}$.

```
BeginProc Two-grid Defect Correction Method
      // Do $\nu_{pre}$ pre-smoothing operations.
      do $\nu_{pre}$ times
            $u^\ell := \hat{S}^\ell\, u^\ell + B^\ell f^\ell$;
      enddo
      // Form the defect and restrict to the coarser grid.
      $f^{\ell-1} := I_\ell^{\ell-1}\, (f^\ell - L^\ell\, u^\ell)$;
      // Solve the defect equation on the coarser grid directly.
      $u^{\ell-1} := (L^{\ell-1})^{-1}\, f^{\ell-1}$;
      // Prolong the coarser grid solution to the finer grid.
      $u^\ell := u^\ell + I_{\ell-1}^\ell\, u^{\ell-1}$;
      // Do $\nu_{post}$ smoothing operations.
      do $\nu_{post}$ times
            $u^\ell := S^\ell\, u^\ell + B^\ell f^\ell$;
      enddo
EndProc
```

Table 3.1: Pseudocode for the two-grid method.

Note that $f^{\ell-1}$ is a coarse grid approximation to the residual $f^\ell - L^\ell u^\ell$, not to f^ℓ.

The amplification matrix $\mathbf{S}_{\text{two grid}}$ of one step of the two-grid method is

$$\mathbf{S}_{\text{two grid}} = (S^\ell)^{\nu_{post}}\, (\, (L^\ell)^{-1} - I_{\ell-1}^\ell\, (L^{\ell-1})^{-1}\, I_\ell^{\ell-1})\, L^\ell (\hat{S}^\ell)^{\nu_{pre}}, \tag{3.6}$$

with $S^\ell = I^\ell - B^\ell\, L^\ell$ and $\hat{S}^\ell = I^\ell - L^\ell\, B^\ell$ the amplification matrices of the smoothing processes.

$$(L^\ell)^{-1} - I_{\ell-1}^\ell\, (L^{\ell-1})^{-1}\, I_\ell^{\ell-1}. \tag{3.7}$$

is called the *relative convergence matrix.* Hackbusch [Hac85] defines the "*approximation property*" in terms of this matrix.

Several authors, e.g. Hackbusch [Hac80a] and Wesseling [Wes80], have shown under certain assumptions $||\mathbf{S}_{\text{two grid}}|| \le c < 1$ with $||\cdot||$ a suitable norm and c independent of mesh size $2^{-\ell}$. The two-grid method is completely determined by the discretizations L^ℓ and $L^{\ell-1}$, the restriction matrix $I_\ell^{\ell-1}$, the prolongation matrix $I_{\ell-1}^\ell$, a smoothing method S_ℓ and the number of pre-smoothing steps - ν_{pre} and post-smoothing steps - ν_{post}.

3.2 The Multigrid Algorithm

The multigrid method makes use of a hierarchy of computational grids Ω_{h_l} and corresponding sets of grid functions U_{h_l}, $l = M, M-1, M-2, \cdots, 1$ defined by (3.1) and (3.2) with h replaced by h_l. Here, h_l is the grid spacing on grid level l. As l decreases h_l increases, that is, the grids Ω_{h_l} become coarser as l gets smaller.

In the two-grid method we solved the coarse grid problem (3.5) directly. The multigrid method approximates the solution $u^{\ell-1}$ of the coarse grid by application of the same two-grid method on the coarse level. So, for $\ell > 2$, the error equation is recursively solved on increasingly coarser grids. When grid level 1 is reached, one usually solves directly for the defect as the time to do the direct solve is inexpensive since u^1 will contain few unknowns. See Table 3.2 for a pseudocode description.

```
BeginProc multigrid method $(\ell, L^\ell, u^\ell, f^\ell, \nu_{pre}, \nu_{post}, I_{\ell-1}^{\ell}, I_{\ell}^{\ell-1})$
    if $\ell = 1$ then
        $u^1 := (L^1)^{-1} f^1;$
    else
        // Do $\nu_{pre}$ pre-smoothing operations.
        do $\nu_{pre}$ times
            $u^\ell := \hat{S}^\ell \, u^\ell + B^\ell f^\ell;$
        enddo
        // Form the defect and restrict to the next coarser grid.
        $f^{\ell-1} := I_\ell^{\ell-1} \, (f^\ell - L^\ell \, u^\ell);$
        $u^{\ell-1} := 0;$
        // Recursively solve on grid $\ell - 1$.
        do multigrid method $(\ell - 1, L^{\ell-1}, u^{\ell-1}, f^{\ell-1}, \nu_{pre}, \nu_{post}, I_{\ell-2}^{\ell-1}, I_{\ell-1}^{\ell-2})$
    endif
    // Prolong the coarse grid correction to the next finer grid.
    $u^\ell := u^\ell + I_{\ell-1}^{\ell} \, u^{\ell-1};$
    // Do $\nu_{post}$ smoothing operations.
    do $\nu_{post}$ times
        $u^\ell := S^\ell \, u^\ell + B^\ell f^\ell;$
    enddo
EndProc
```

Table 3.2: Pseudocode for the multigrid method.

Up to this point we have said nothing about the details of the multigrid parameters we chose. We start by constructing our initial guess by applying the difference equation to the boundary points. We use V cycles, a direct solve on the coarsest grid and red-black Gauss-Seidel relaxation for the smoothing routine. We discuss the issues of grid specification and grid transfer operators in §3.3 and §3.4.

The way we use multigrid in the Schwarz process changes the notion of the cycling procedure slightly. To better explain when smoothing takes place we introduce four cycling parameters, *pbeg, pdown, pup, ptop*,

- pbeg $\equiv$ number of smoothing iterations on the finest level before cycling begins,
- pdown $\equiv$ number of smoothing iterations on level ℓ after restriction to level ℓ from $\ell + 1$,
- pup $\equiv$ number of smoothing iterations on level ℓ after prolongation (from $\ell - 1$ to ℓ) on all levels, with the exception of smoothing after prolongation to the finest level which is a separate case,
- ptop $\equiv$ number of smoothing iterations on the finest level M after prolongation from level $M - 1$.

3.3 Determining Coarse Grid Approximations

Generally, one considers two strategies for determining coarse grid approximations: coarse grid finite difference approximations (CFA) and coarse grid Galerkin approximations (CGA). The discussion in this section is derived from [Den82], [Wes84], and [ABDP81]. The simplest and most straightforward method of grid coarsening is *standard coarsening.* Standard coarsening falls under the category of CFA. For example, in the simplest case, Ω_ℓ and $\Omega_{\ell-1}$ are rectangular grids, the grid points of $\Omega_{\ell-1}$ are a subset of the grid points of Ω_ℓ, and the grid spacing $h_{\ell-1}$ of $\Omega_{\ell-1}$ is twice the grid spacing h_ℓ of Ω_ℓ. Recall that M indicates the highest level grid. If the above relationships are to hold true, then the number of grid points on each side of Ω_M, n_M, is constrained to $n_M = (n_1 - 1)\, 2^{M-1} + 1$, where n_1 is the number of grid points on each side of Ω_1 [Den82]. We see standard coarsening leads to grids whose size is always some 'magic' number. In a real problem if one's grid specification is not a 'magic' number, or is an irregular discretization, then an interpolation or transformation is necessary to use multigrid with standard coarsening. The advantages of CFA include:

- The definition of $L^{\ell-1}$ is independent of the fact that we solve an equation at level ℓ.
- One needs no additional computations for defining $L^{\ell-1}, L^{\ell-2}, \cdots, L^1$.
- Min'mal storage requirements.

Nonetheless, the ability to maintain arbitrarily sized grids without resorting to a messy interpolation phase is of utmost importance when studying the Schwarz method. Any computational study of the method invariably includes measuring the computational cost as a function of the overlap. A combination of these reasons and others to be mentioned below led us to abandon the CFA approach.

We built our Schwarz-multigrid program from the core subroutines of Dendy's blackbox multigrid code, BOXMG [Den82]. The BOXMG code implements multigrid using the "correction scheme" as opposed to the "full approximation scheme". Instead of using standard coarsening, BOXMG implements a more complicated coarsening scheme based on CGA. BOXMG allows mixed derivatives and nonselfadjoint equations. BOXMG works well in the selfadjoint case for smooth or strongly discontinuous coefficients. The user provides a finite difference approximation on the finest grid, which is uniform and rectangular. Dendy's primary interest was solving the neutron diffusion equation,

$$-\nabla \cdot (D(x,y)\nabla U(x,y)) + \sigma(x,y)U(x,y) = F(x,y), \quad (x,y) \in \Omega, \tag{3.8}$$

in which D, σ and F are piecewise constant. With $\sigma(x,y) = 0$ and $D(x,y) = 1$, equation (3.8) reverts to the simpler case of Poisson's equation. Appendix C gives details about the memory requirements for BOXMG.

In CGA a coarser grid $\Omega_{\ell-1}$ contains grid points which may not be a subset of the mesh points of the immediately finer grid Ω_ℓ. We obtain a coarse discrete approximation on $\Omega_{\ell-1}$ using the Galerkin formula

$$L^{\ell-1} = (I^{\ell}_{\ell-1})^* \, L^\ell \, I^\ell_{\ell-1}.$$

Some disadvantages of the Galerkin approach are:

- A preprocessing phase is needed to compute $L^{M-1}, L^{M-2}, \cdots, L^1$.
- Only works on linear problems.
- Usually the five- or seven-point grid stencils on grid level M will not be preserved on the coarser grid levels but become nine-point formulae.
- The definition of $L^{\ell-1}$ depends on L^ℓ. If the solution of $L^\ell u^\ell = f^\ell$ is followed by a multi-grid iteration for solving $L^{\ell+1}u^{\ell+1} = f^{\ell+1}$ we need matrices $L'^{\ell} \equiv rL^{\ell+1}p$, $L'^{\ell-1} \equiv rL'^{\ell}p$ which possibly differ from L^ℓ and $L^{\ell-1}$ used before. The matrices r and p are restriction and prolongation operators defined appropriately. (Pertains to FMG cycle.)

On the other hand the advantages of CGA are:

- Unrestricted number of points in the x and y directions allowed on any grid.
- Automatic generation of possibly irregular difference formula near any boundary.
- Automatic homogenization of rapidly varying coefficients.
- If L^ℓ is symmetric (positive definite), and if $I^{\ell-1}_{\ell} = (I^\ell_{\ell-1})^*$ then $L^{\ell-1} = I^\ell_{\ell-1} L^\ell I^\ell_{\ell-1}$ is also symmetric (positive definite).

Thus, CGA allows one to solve a larger class of problems.

3.4 Specification of Grid Transfer Operators

Finally, how should we choose prolongation and restriction operators? Brandt [Bra77] gives the following general result. We say the operator $I^{\ell}_{\ell-1}$ is of order m_p if polynomials of degree $m_p - 1$ are interpolated exactly, and that $I^{\ell-1}_{\ell}$ is of order m_r if $(I^{\ell-1}_{\ell})^T$ interpolates polynomials of degree $m_r - 1$ exactly. If the order of the differential equation is $2m$ then we should have

$$m_p + m_r > 2m.$$

The coding of interpolation can be further complicated by whether the points on the boundary represent knowns (Dirichlet boundary conditions) or unknowns (Neumann boundary conditions).

If the finite difference approximation of Eq. (3.8) is a vertex-centered one as in [ABDP81], then the *classic* multigrid method of [Bra77] ($I^{\ell}_{\ell-1}$ = bilinear interpolation, $I^{\ell-1}_{\ell}$ = a fixed nine point weighting operator, and the coefficients of $L^{\ell-1}$ a fixed weighting of the coefficients of L^{ℓ}) performs well as long as the discontinuities in D are not too severe and as long as the internal interfaces do not consist of too many line segments; otherwise, it performs badly; indeed it can fail to converge in the fixed mode described above.

Alcouffe et al. [ABDP81] dealt with the situation in which D, σ, and f jump by orders of magnitude across internal interfaces. They considered many possible choices of $I^{\ell-1}_{\ell}$ and $I^{\ell}_{\ell-1}$. Only one of these choices was found to be robust, that of

$$I^{\ell-1}_{\ell} = (I^{\ell}_{\ell-1})^*.$$

The solution they use is as follows: Suppose that at (x_i, y_j), L_ℓ has the pointwise template

$$\begin{bmatrix} -T^{\ell}_{i,j+1} & -W^{\ell}_{i,j+1} & -R^{\ell}_{i+1,j+1} \\ -Q^{\ell}_{i,j} & S^{\ell}_{i,j} & -Q^{\ell}_{i+1,j} \\ -R^{\ell}_{i,j} & -W^{\ell}_{i,j} & -T^{\ell}_{i+1,j} \end{bmatrix}. \qquad (3.9)$$

Form the "vertical sums"

$$\begin{aligned}
\tilde{Q}_{i+1,j} &= T_{i+1,j+1} \quad +Q_{i+1,j} \quad +R_{i+1,j}, \\
\tilde{S}_{i+1,j} &= -W_{i+1,j} \quad +S_{i+1,j} \quad -W_{i+1,j+1}, \\
\bar{Q}_{i+2,j} &= T_{i+2,j} \quad +Q_{i+2,j} \quad +R_{i+2,j+1}.
\end{aligned}$$

Then for horizontal lines embedded in the coarse grid, the interpolation $I^{\ell}_{\ell-1}$ is given by

$$(I^{\ell}_{\ell-1}\ u^{\ell}_{i+1,j}) = (\tilde{Q}_{i+1,j}\ u^{\ell-1}_{i,j} + \bar{Q}_{i+2,j}\ u^{\ell-1}_{i+1,j})/\tilde{S}_{i+1,j}. \tag{3.10}$$

(We have just summed Eq. (3.9) vertically to average out its y-dependence.) A similar formula can be used for vertical lines embedded in the coarse grid rectangles. Then, at fine grid points centered in coarse grid squares, $u^{\ell}_{i+1,j+1}$ may be obtained from the difference formula; i.e.,

$$\begin{aligned}
u^{\ell}_{i+1,j+1} = (\ & Q_{i+1,j+1}\ u^{\ell}_{i,j+1} + Q_{i+2,j+1}\ u^{\ell}_{i+2,j+1} \\
+\ & W_{i+1,j+1}\ u^{\ell}_{i+1,j} + W_{i+1,j+2}\ u^{\ell}_{i+1,j+2} \\
+\ & R_{i+1,j+1}\ u^{\ell}_{i,j} + R_{i+2,j+2}\ u^{\ell}_{i+2,j+2} \\
+\ & T_{i+1,j+2}\ u^{\ell}_{i,j+2} + T_{i+2,j+1}\ u^{\ell}_{i+2,j} \quad)\ /\ S_{i+1,j+1}
\end{aligned} \tag{3.11}$$

The vertical analog of (3.9)-(3.10) completes the definition of $I^{\ell}_{\ell-1}$.

Prolongation by linear interpolation is inaccurate when u in not locally linear between coarse grid points. This inaccuracy is severe when $D(x,y)$ in (3.8) is discontinuous between coarse grid lines. BOXMG can handle strongly discontinuous coefficients because it uses the matrix-dependent prolongation described above.

Many variations of the multigrid algorithm have been constructed by choosing different initialization procedures, cycle types, storage schemes, coarsest grid solvers, smoothing operators, restriction operators, prolongation operators, and pre-smoothing and post-smoothing iteration counts [ST82a].

3.5 Computational Efficiency

Unlike most iterative methods, when solving symmetric positive definite linear systems every multigrid cycle reduces the error at each step by a constant factor until the roundoff level is reached. This property is very useful when deciding when to stop solving. Many scientists choose to decrease the residuals by a factor and thus can explicitly program their code to do the necessary cycle count. Another nice property of multigrid is that the number of multigrid cycles needed to attain convergence is independent of the mesh spacing when FMG is used.

Another of the keys to the speed of multigrid is not as many points exist on levels $M-1, M-2, \cdots, 1$. The total work on all coarser levels for two space dimensions is about 1/3 of the work on the finest level.

Recall, for iterative methods as expressed in equation (2.17), the spectral radius of the smoothing matrix S_h must be less than one for the method to achieve convergence because the S_h is the amplification matrix of the iterative method. It is interesting to note that for multigrid this is not the case, because the smoothing matrix is only part of the total amplification matrix. In multigrid the purpose of the smoothing matrix S_h is to smooth the high frequency error on the grid of interest. An asymptotically divergent smoother may prove acceptable if the low frequency error components aren't amplified in the small number of smoothing iterations done on each grid level.

3.6 Historical Comments

Here we present a chronology of the development of multigrid, taken for the most part from Hackbusch's book [Hac85]. His book includes a comprehensive 20 page listing of multigird publications.

The first correct two-grid iteration was described by Fedorenko [Fed62]. He emphasized the complementary roles of the Jacobi iteration and of the coarse-grid correction.

It was also Fedorenko [Fed64] who formulated the first multigrid algorithm in 1964 and proved the typical convergence behavior. He described a multigrid method for the Poisson equation in a square, and he proved that the number of operations is $O(n)$, where n is the number of grid points. The much more complex situation of a difference scheme for a second order elliptic equation with variable coefficients in a rectangle was considered by Bakhvalov [Bak66] in 1966. His main focus was the optimal order of complexity achieved by the multigrid solution process rather than its practical efficiency.

In 1972 Brandt [Bra72] following the papers mentioned above discovered the efficiency of the multigrid algorithm. He laid emphasis on the combination of the multigrid process with additional techniques which should yield a "multi-level adaptive" method. A precise definition of his multigrid algorithm was given in 1975/1976 (cf. Brandt[Bra76, Bra77]). However, in these and later papers the considerations about convergence remain very vague. Brandt [Bra77] described a multigrid method similar to that of Fedorenko and Bakhvalov, and demonstrated its practical usefulness. Furthermore, he proposed ideas for adaptive discretization in certain parts of the region, e.g., in the neighborhood of singularities.

An important step towards convergence analysis was made by Nicolaides [Nic75], [Nic77]. While his first paper described a two-grid iteration, the second one from 1977 studied the convergence of a finite element discretization. Astrachancev [Ast71] and Nicolaides applied a multigrid method on finite element problems and gave convergence proofs. Hackbush published an early survey of convergence proofs in [Hac80b]. Other convergence proofs and experiments are given in Chapters 6 and 7 of Hackbush [Hac85], by Wesseling in [Wes80], by Greenbaum in [Gre84] and by Mol [Mol81a]. Bank and Dupont gave two different convergence proofs in a report from 1977, which has since been divided. The second part of the report is published in a journal (cf. Bank-Dupont [BD81]), whereas the first one, containing a new approach is available only as a report (cf. Bank-Dupont [BD80]). Chan and Tuminaro survey parallel multigrid algorithms in [CT87].

Chapter 4

Outer Iterations vs Inner Iterations

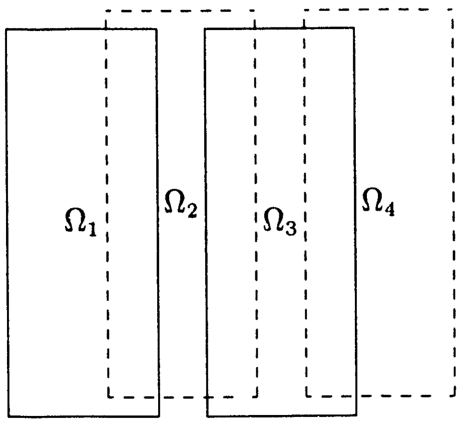
Having discussed the Schwarz alternating procedure in Chapter 1 and the multigrid method in Chapter 3 we now turn to the combination of the two algorithms. In our combined algorithm, SAP-MG, the Schwarz alternating procedure is the outer iteration procedure and the inner iteration procedure is some type of multigrid cycle executed some number of times. To establish useful variants of SAP-MG methods we consider the issues of the amount of overlap between subdomains, subdomain shape and the choice of inner iteration procedure.

Model Problem

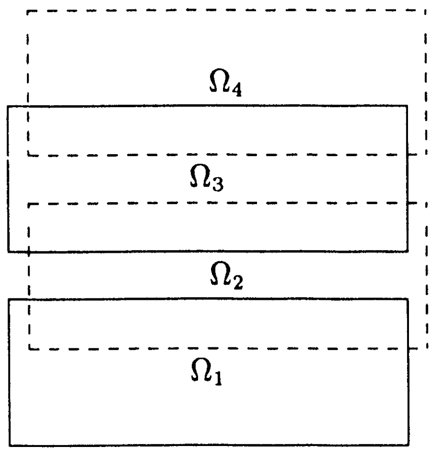
All experiments use the boundary conditions and right hand side of the model problem described in §2.1. We use a five point grid stencil on the finest grid and nine point grid stencil on the coarser grids. The smoothing method is colored point relaxation and we use a uniform $n_h \times n_h$ grid at the finest level.

4.1 Overlap

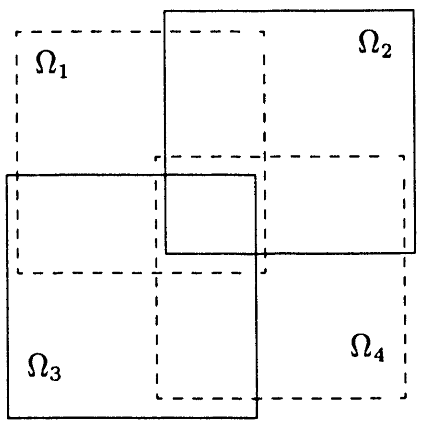
We need to know the effects and tradeoffs of overlapping subdomains to understand the SAP-MG algorithm. The overlap affects the convergence rate of the



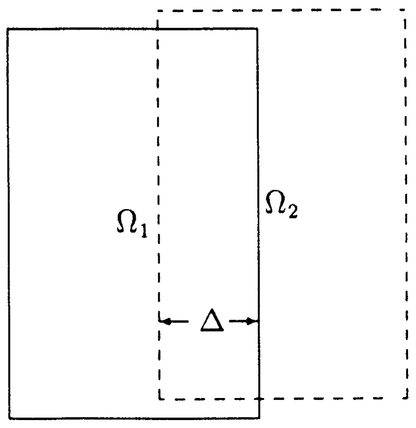
SD4a



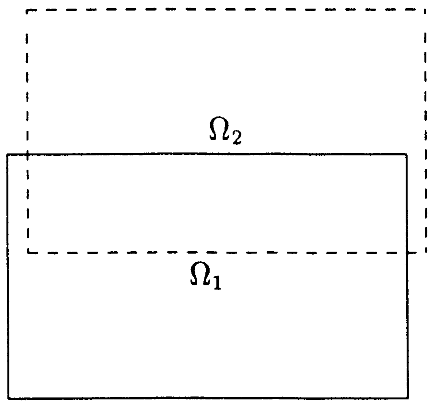
SD4b



SD4c



SD2a



SD2b

Figure 4.1: Schwarz Splittings

algorithm, communication cost, computational cost and memory use. The extent to which subdomains overlap affects the rate at which information travels across pseudo-boundaries. Analysis in the continuous domain shows the greater the overlap the better the convergence rate.

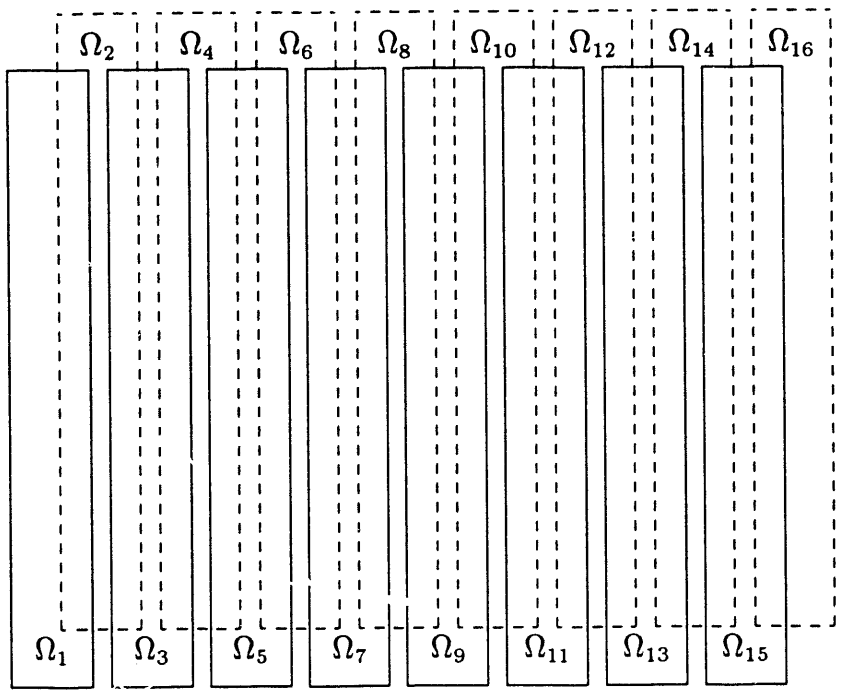
Both the communication and computational costs are proportional to the overlap. Obviously, communicating a smaller amount of information leads to lower communication costs. The computational overhead of SAP-MG per outer iteration roughly equals the percentage overlap; i.e., with two subdomains, 50% overlap implies 50% more floating point operations. We show in §4.6 and §4.7 how increasing the overlap, and thus communication cost and computational cost per outer iteration, decreases the time to solution because of improved convergence rates. We show in Appendix C how subdomain overlap affects memory costs.

4.2 Choice of Inner Iteration Procedure

In the continuous domain theoretical model of the Schwarz Alternating Procedure the convergence analysis assumes obtaining the analytical solution on the subdomain problem before updating the interior pseudo-boundaries. We could simulate this modus operandi in the discrete domain by solving on a subdomain until the discrete L_2 norm of the residual was as accurate as the discretization error or a truncation error estimate. However, in the initial outer iteration of the SAP the information passed at the pseudo-boundary areas does not approximate the global domain solution with a high degree of accuracy. We hypothesize spending computational time to solve an inaccurate subdomain problem accurately is a senseless proposition. Note, in the discrete domain not only do we update the pseudo-boundary values, but we update the entire overlapping boundary regions with the latest values.

4.3 Subdomain Shape

The inherent partitioning of the SAP limits the number of grid levels visited. In the SAP-MG method the subdomains are obviously smaller than the global domain.



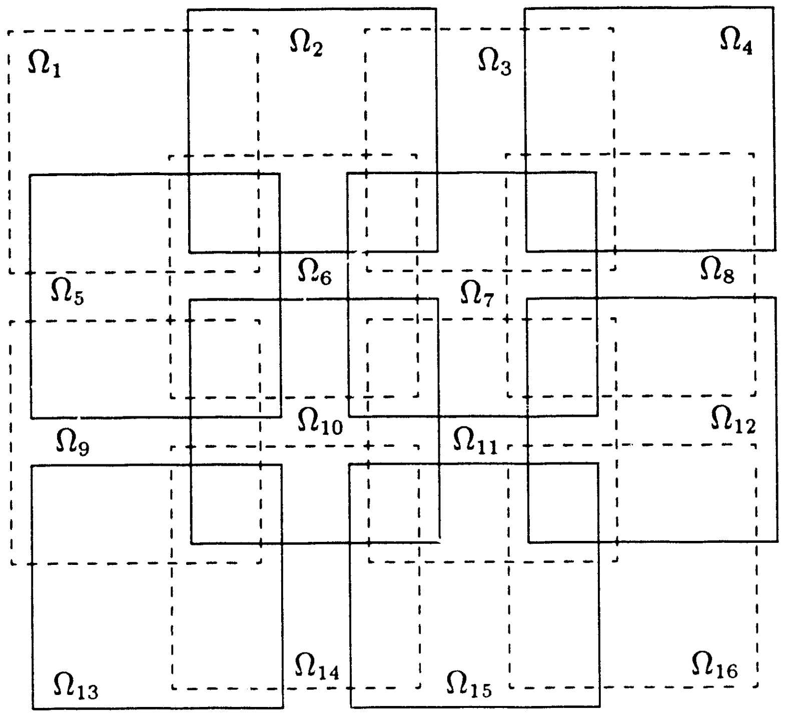


Figure 4.2: Sixteen Subdomain Schwarz Splittings (bottom $q = 4$).

Recall multigrid derives its speed from its ability to knock out the low frequency components of the error on the coarser grids. However, on the subdomains we cannot eliminate the lowest frequency wavelengths available to multigrid applied to Ω_G in at least one of the coordinate directions. Since, the convergence rate of multigrid is dependent only on the convergence rate of the lowest frequency wavelengths, and the decomposition of the global domain into subdomains eliminates the possibility of attacking the lowest frequency wavelengths, we expect the convergence rate of an SAP-MG method to be slower than that of a multigrid method. Furthermore, as the number of subdomains increases, and as a result the subdomains get smaller, we expect that the convergence behavior of SAP-MG becomes less like multigrid and more like that of the relaxation method.

Tang [Tan87] found for a solver other than multigrid on an equal number of subdomains, when the global domain was partitioned into a checkerboard of squares the problem converged much faster than when partitioned into a series of thin strips. We had hypothesized long thin regions may preserve some of the effect of getting rid of the lowest frequency components of the error in one direction when using multigrid as the inner iteration procedure. However, a simple set of initial experiments showed this not to be the case. Alternating between horizontal and vertical rectangular regions may conceivably offer advantages over other Schwarz splittings because we can access the lowest frequencies in both directions. We explore this idea in more detail in §4.8.

The subdomain shape also affects M, the number of grid levels visited. As M decreases a greater percentage of the total work is done on the coarsest grid(s) during the direct solves. The operation count of a direct solve is greater than that of a relaxation so we gain computationally by making n_h on the coarsest grid as small as possible. To maximize the number of grid levels one chooses an equal number of points in all coordinate directions. Using a semi-coarsening procedure would allow us to increase the number of grid levels visited for rectangular regions. Semi-coarsening continues coarsening in the wider direction after coarsening in the narrower direction stops. However, our code does not include this feature.

We illustrate the number of grid levels visited by SAP-MG in Table 4.1, assuming the decomposition of a 128×128 problem into $q \times q$ square subdomains, each of which overlaps its nearest neighbors by 50% in the direction of overlap. M is the number of grid levels visited. n_h is the number of grid points on the boundary. Figure 4.2 illustrates the domain decomposition for $q = 4$.

q	1	2	4	8	16	32
M	6	6	5	4	3	2
n_h	128	58	51	28	15	7

Table 4.1: Grid information for $q \times q$ domain decompositions.

On machines with vector hardware the subdomain shape affects vector length. The length is proportional to the number of grid points in the direction of vectorization. It may pay to maintain some degree of vector length by avoiding regions that are relatively thin in the direction of vectorization.

4.4 Comparing Convergence Criteria

To begin our experiments, we ran SAP-MG on two subdomains using one multigrid V cycle for the inner iteration procedure. We varied the overlap between 3 and 123 in increments of 5. An overlap of 128 corresponds to the ordinary multigrid algorithm. In this experiment we studied the convergence issue, and we show the effects of varying the overlap. Table 4.2 charts the number of outer iterations to convergence for four stopping criteria. The columns labeled ARB correspond to the stopping criterion $\left\|\mathbf{r}_h^k\right\|_h < 10^{-6}$. The columns labeled ANS correspond to the stopping criterion $\left\|\mathbf{r}_h^k\right\|_h < \|\mathbf{e}_h\|_E$. The columns labeled RMS refer to the RMS ratio in the stopping criterion of §2.7.2. The columns labeled TAU refer to the stopping criterion of $\left\|\mathbf{r}_h^k\right\|_h < \left\|\tau_h^k\right\|_h$. Recall, $\mathbf{r}_h^k$ is the residual of iterate k, $\mathbf{e}_h$ is the exact discretization error and τ_h^k is a local truncation error estimate. The criteria of columns ANS, RMS

Stopping Criterion	Sequential				Chaotic Parallel*			
	ARB	ANS	RMS	TAU	ARB	ANS	RMS	TAU
overlap								
3	82	52	58	58	***	***	***	***
8	34	21	26	23	71	47	48	49
13	22	14	18	15	49	34	33	35
18	17	11	14	11	40	27	27	28
23	14	9	12	9	31	21	21	22
28	11	6	9	7	27	19	4	19
33	10	7	6	7	23	16	3	16
38	9	6	6	6	21	15	6	13
43	8	6	6	6	18	12	13	12
48	7	5	5	5	18	13	8	13
53	8	5	5	5	14	10	10	10
58	8	5	5	5	14	10	8	10
63	7	5	5	5	13	10	9	9
68	8	5	5	5	12	8	6	8
73	8	5	5	5	14	9	6	9
78	7	5	5	5	13	8	6	8
83	7	5	5	5	12	9	6	9
88	7	5	4	5	12	9	6	9
93	7	5	4	5	13	9	6	9
98	7	5	5	5	14	10	8	10
103	7	5	4	5	12	9	7	9
108	7	5	4	5	12	9	7	8
113	7	5	4	5	11	8	6	8
118	7	5	4	5	13	10	7	8
123	7	5	4	4	10	7	5	7
128	7	5	5	5	–	–	–	–

Table 4.2: Outer iteration count versus overlap for the different convergence criteria. Inner iteration procedure of 1 V cycle. *Representative data since non-reproducible. *** $>$ 150.

and TAU have theoretical support. The criterion of the column ARB is a guess, possibly based on experience. The criteria of columns ANS and RMS depend on knowing the analytical solution and thus are useful for convergence studies only. The outer iteration counts for the columns labeled ANS, RMS and TAU agree with one another well for the sequential SAP. Also, the outer iteration counts for the columns labeled ANS, and TAU agree with one another well for the parallel SAP. In the chaotic parallel method, the column labeled RMS displays odd results. The reason the outer iteration count is smaller than it ought to be is because the RMS error was not monotonically decreasing. The ARB criterion obviously leads to more outer iterations than are necessary for this size problem. These convergence tests are applied to Ω_G.

4.5 Parallel Method

In the parallel SAP, at the beginning of each outer iteration, each of the subdomains is parceled out to a processor. The individual processors work independently to advance their respective subdomain solutions. When the processors have completed performing the inner iteration procedure on the their respective subdomains they update the solution vector of Ω_G. The value of the solution at a grid point after all the inner iterations are complete is the value that was computed by the last processor to update that grid point. Since the update process is non-deterministic for grid points that reside in multiple regions we name this type of update procedure *chaotic updating.* We define a synchronization point between the subdomain processes and the convergence check. After all processors arrive at the synchronization point they compute the convergence test for Ω_G in parallel by microtasking the loops in the convergence test. We define yet another synchronization point after the convergence test; processors reaching this point wait until the convergence test is complete. If the problem hasn't converged, the processors begin again working on subdomains in parallel. We present results for two and four processors/subdomains.

SAP-MG: Two Subdomains								
Inner Iterations	Sequential				Chaotic Parallel			
	1	2	3	τ	1	2	3	τ
overlap								
3	58	52	52	53	90	90	83	87
8	23	23	23	23	52	53	51	51
13	15	15	15	15	34	33	32	31
18	11	11	10	11	28	27	26	25
23	9	8	8	8	20	19	20	19
28	7	6	6	6	18	16	18	17
33	7	5	5	5	18	16	16	15
38	6	5	5	5	15	15	13	14
43	6	5	5	5	12	13	12	12
48	5	5	5	5	10	12	11	11
53	5	4	4	4	10	11	10	10
58	5	4	4	4	8	10	10	9
63	5	4	4	4	9	9	9	9
68	5	4	4	4	9	8	8	8
73	5	4	3	4	9	7	8	7
78	5	4	3	3	9	7	7	7
83	5	4	3	3	9	7	7	7
88	5	3	3	3	8	6	7	7
93	5	3	3	3	9	6	5	5
98	5	3	3	3	9	6	5	5
103	5	3	3	3	7	6	5	5
108	5	3	3	3	9	6	5	5
113	5	3	3	2	8	5	5	5
118	5	3	2	2	7	5	5	4
123	4	3	2	2	7	5	4	4
128	5	—	—	1	—	—	—	—

Table 4.3: Experimental results for a study of outer iteration count versus the inner iteration procedure.

4.6 Two Subdomain Results

The domain decomposition studied in this section is that labeled SD2a in Figure 4.1. Table 4.3 charts the experimental results for a study of outer iteration count versus the inner iteration procedure. Again, we present results for both the sequential and chaotic parallel SAP. At the top of the columns of data is the indicator of the inner iteration procedure used. The columns labeled 1, 2, and 3 correspond to doing 1, 2, and 3 multigrid V cycles for the inner iteration procedure. The results for the simulated direct solve are in the column labeled τ. Its outer iteration count was minimal, except for the aberration for overlap equal to 18 in the sequential SAP with a 3 cycle inner iteration procedure. For most of the overlaps increasing the number of V cycles done during the inner iteration procedure did not decrease the outer iteration count more than one or two cycles. The decrease is barely noticeable in most cases. As the overlap increased the outer iteration count decreased. For the smallest overlap, the outer iteration count is an order of magnitude greater than the outer iteration count for most overlaps. The outer iteration count for the chaotic parallel SAP is approximately double that of the sequential SAP.

4.7 Four Subdomain Results

The domain decomposition studied in this section is that labeled SD4c in Figure 4.1. Preliminary experiments showed domain decomposition SD4c to converge faster than domain decompositions SD4a or SD4b. Since these results mirrored the assertion of Tang, we decided to only consider square subdomains.

The four subdomain results found in Table 4.4 and Table 4.5 are similar to those of the two subdomain results. Again the tables chart the experimental results for a study of outer iteration count versus the inner iteration procedure. The results for the sequential SAP-MG and chaotic parallel SAP-MG are presented in separate tables. At the top of the columns of data is the indicator of the inner iteration procedure used. The columns labeled 1, 2, and 3 correspond to doing 1, 2, and 3 multigrid V cycles for the inner iteration procedure. The results for the simulated direct solve

Sequential SAP - 4 Subdomains								
	Outer Iterations				Floating Point Operations ($\times 10^6$)			
Inner Iterations	1	2	3	τ	1	2	3	τ
overlap 3	***	***	***	***	***	***	***	***
8	46	46	44	44	80.7	119.	161.	143.
13	27	28	27	27	51.6	79.5	107.	101.
18	16	17	17	17	34.3	55.5	74.7	69.6
23	13	13	13	13	30.3	46.1	61.9	57.1
28	8	9	9	9	22.1	36.6	49.0	44.9
33	5	7	5	5	16.2	24.0	31.8	28.3
38	7	6	7	7	22.2	33.3	44.4	38.7
43	7	6	6	6	22.4	30.9	41.0	36.9
48	6	5	5	5	23.3	29.0	38.5	36.0
53	5	5	5	5	20.5	30.4	40.3	38.3
58	5	5	5	5	21.7	32.1	42.6	38.5
63	5	5	5	5	22.6	33.6	44.5	38.9
68	5	5	4	4	24.4	30.7	40.5	39.1
73	5	4	4	4	25.4	31.9	42.1	36.9
78	5	4	4	4	26.7	33.5	44.3	38.8
83	4	4	3	3	23.7	34.8	37.5	35.4
88	4	4	3	3	25.3	30.6	40.1	35.2
93	4	4	3	3	25.3	31.7	41.6	39.3
98	4	4	3	3	26.3	33.2	43.5	39.2
103	4	4	3	3	28.5	34.4	45.1	37.7
108	4	3	2	2	30.3	36.5	37.1	34.9
113	4	3	2	2	31.3	37.7	38.4	33.8
118	4	3	2	2	32.7	39.4	40.0	35.3
123	3	3	2	2	28.0	31.8	41.3	34.1
128	5	—	—	1	9.0	—	—	—

Table 4.4: Outer iterations and flops versus overlap for 4 square subdomains.

are in the column labeled τ. These tables also include information about the number of floating point operations. The charting of the floating point operation count allows us to determine the most computationally efficient inner iteration procedure.

Comparing these four subdomain results to the two subdomain results, we see in most cases for a given overlap, the four subdomain method took more iterations to converge. The operation count data shows that in all cases an inner iteration procedure of one cycle is the most efficient. These results lend credence to our statement about not bothering to solve a subdomain problem with inaccurate boundary conditions to a high degree of accuracy.

In the sequential experiment, Table 4.4, the work necessary for the SAP-MG is over twice that of the ordinary multigrid algorithm for all overlaps except one. This does not bode well for obtaining high multiprocessing efficiencies. Assuming a parallel SAP-MG variant has the same outer iteration count as the sequential SAP-MG variant, (and it probably won't because the sequential SAP-MG always is using the newest information) computational efficiency for a four processor system is likely to be less than 50%.

In the chaotic parallel experiment, Table 4.5, we see the outer iteration count in the chaotic update variant of the parallel SAP-MG is 2 to 4 times that of the sequential SAP-MG. For a shared memory machine with enough memory to fit the ordinary multigrid problem, the chaotic update variant of parallel SAP-MG offers no benefits. The minimum computational cost divided by number of processors (4) is greater than the cost of solving the global domain using ordinary multigrid. A simple scheme of averaging solutions (or weighted averaging) in the overlapping regions did nothing to improve the convergence of the parallel method. In fact, it always made it worse. We hypothesis the chaotic method of the update is in some sense optimal for the multigrid method because of the way the error is distributed. The residual error in the chaotic update is most likely not smooth.

PSAP - 4 Subdomains - 4 CEs with Chaotic Updating								
	Outer Iterations				Floating Point Operations ($\times 10e6$)			
Inner Iterations	1	2	3	τ	1	2	3	τ
overlap 3	***	***	***	***	***	***	***	***
8	112	107	108	107	192.	284.	483.	442.
13	77	72	72	72	141.	204.	389.	321.
18	51	50	49	49	101.	153.	204.	252.
23	43	42	41	42	91.0	137.	181.	226.
28	36	34	34	32	84.2	123.	166.	191.
33	31	30	30	28	76.8	114.	155.	180.
38	27	24	25	25	72.0	107.	139.	175.
43	24	21	22	22	67.6	104.	129.	164.
48	22	20	21	18	67.6	99.4	134.	139.
53	19	16	19	18	62.0	99.6	128.	150.
58	16	16	17	16	45.1	85.7	122.	144.
63	12	15	17	15	45.5	89.5	127.	137.
68	17	13	15	11	66.5	90.8	122.	111.
73	16	13	13	14	65.6	83.1	111.	142.
78	16	14	14	10	69.0	87.4	125.	121.
83	15	13	14	10	67.6	97.0	130.	125.
88	15	12	13	9	72.2	97.0	139.	117.
93	15	14	14	8	75.0	107.	125.	112.
98	13	12	12	8	69.2	98.1	141.	120.
103	12	11	13	8	67.0	94.3	126.	123.
108	12	11	11	7	71.1	100.	155.	115.
113	12	10	13	10	73.4	95.2	149.	151.
118	9	7	12	8	60.1	73.7	156.	128.
123	9	8	11	8	62.1	84.9	149.	133.
128	5	—	—	1	9.0	—	—	—

Table 4.5: Outer iterations and flops versus overlap for 4 square subdomains.

4.8 Alternating Vertical and Horizontal Subdomains

For the smallest overlaps the outer iteration count increases dramatically as the overlap goes to zero, because only a small amount of information from one subdomain transfers over to the other subdomain. Zero overlap implies that the subdomains share no unknowns. Information is passed only at the boundaries. That is, in the case of two vertical subdomains, the rightmost unknown of the leftmost subdomain would be the boundary of the rightmost subdomain and visa versa. As the overlap increases the computation work per outer iteration increases. Our desire is to minimize the time to convergence. As an attempt to provide greater communication between opposite sides of Ω_G we resort to alternating between vertical and horizontal subdomains. Using this trick on every other outer iteration information travels across an internal interface that it wasn't traveling across the previous outer iteration. For example, the domain decomposition alternates between those illustrated in Figure 4.1, SD2a and Figure 4.1, SD2b.

	Sequential SAP-MG		Parallel SAP-MG	
overlap	alternating	vertical	alternating	vertical
0	83	36	**	47
1	**	25	81	37
2	28	20	58	32
3	22	17	45	29
4	17	14	35	27
5	14	13	28	25
6	12	11	27	23
7	11	10	23	22
8	9	9	23	21

Table 4.6: Comparison of SAP-MG with vertical subdomains against SAP-MG with alternating vertical and horizontal subdomains for small overlaps.

Table 4.6 shows a marked improvement for the smallest overlaps. However, the algorithm still converges in an unfavorable outer iteration count. A ** in the field of the table means more than 150 outer iterations. The results are for the model problem of §2.1 solved on a 64×64 grid of unknowns with an inner iteration procedure of one multigrid V cycle, with the cycling parameters of §3.2 equal to one. Convergence was determined by comparison of the discrete L_2 norm of the residual to the $\|\cdot\|_E$ norm of the global discretization error. For overlaps exceeding those tabulated this alternating method showed little if any improvement.

4.9 Conclusion

For the model problem considered, we conclude the inner iteration procedure should be one multigrid V cycle. We conclude using q square subdomains is better than using q vertical or horizontal subdomains. We conclude an overlap of in the neighborhood of 50% results in a more efficient SAP-MG algorithm than one with a relatively small or large overlap. In exploring chaotic updating we found the convergence rate to be substantially slower for the parallel updating of shared areas. We saw evidence that supports others' statements that applying parallelism at the subdomain level in conjunction with multigrid does not result in a faster algorithm.

Chapter 5

Domain Decomposition for Large Systems

With the current version of the multigrid program and a particular instance of the Alliant hardware, running the model problem with over 730×730 unknowns uses all the physical memory and incurs heavy overhead because of page faults. As the multigrid problem size reaches 800×800 the computational efficiency decreases dramatically because of page faulting but the problem does run to completion. When we ran a 1024×1024 multigrid problem the computation starts but the operating system terminates it, presumably because of insufficient memory for system tables and/or swap space.

In this chapter we test the hypothesis that by limiting the spatial locality of the data operated on, through decomposition of the global domain into subdomains, the page fault rate drastically decreases resulting in a more efficient computation. The idea is analogous to that of strip-mining vector loops for optimizing cache performance, but the optimization occurs at a different level of the memory hierarchy.

Given an elliptic problem, increasing its size will eventually cause its memory requirements to exceed the size of physical memory. The exact size of the specific problem for which this happens is not the relevant consideration, since a more complicated problem, e.g., nonconstant coefficients and/or large discontinuities, may have different storage requirements and as a result have a different memory size threshold.

Also, the amount of physical memory varies from one computer system to another. Our concern here is characterizing the general behavior.

We ran four experiments to show the usefulness of the SAP in this context. All experiments use the boundary conditions and right hand side of the model problem described in §2.1. The first experiment charts the page fault behavior of ordinary multigrid as the problem size increases, revealing the need for an improved method. The second experiment shows how applying additional processors to a multigrid problem incurring page faults has little effect in decreasing wall clock time to solution. The third experiment detailed in §5.2 shows the superiority of the SAP-MG method over multigrid for a specific problem. Finally, §5.3 presents results for SAP-MG with an extended set of domain decompositions over a range of problem sizes.

Throughout this process we employ all the Alliant CEs using the do loop, microtasking parallelism facilities provided by the Alliant FX Series hardware and compiler.[1] In the Alliant FX Series Architecture 'CE' refers to the enhanced computational element [FXm86]. The 'CE' moniker distinguishes the vector processing CPU from the interactive processor CPU.

Although we solve using multigrid cycles during the inner iteration procedure in this thesis, the Schwarz theory states for elliptic problems using any convergent solver during the inner iteration procedure suffices to reach a global solution [KK58]. We point this out because multigrid is only one of many solvers used.

The outer iteration procedure is specified as follows: sequentially visit each subdomain, while visiting the subdomain, advance the subdomain solution by performing the inner iteration procedure, update convergence data structure, and update the global solution, then proceed to the next subdomain until all of the subdomains have been visited. After all subdomains have been visited check the convergence data structure to see if the stopping criteria is met, if it is not met, repeat the outer iteration procedure. The inner iteration procedure is 1 V cycle with cycling parameters *pbeg, pdown, pup, ptop* all set to 1. (Parameters are explained in §3.2.) We construct the initial guess by applying the difference operator to the boundary points. Timing begins after initializing the arrays U and F; as a result, the system time to set up

[1]See Chapter 6.

		Major	Time (seconds)			
		Page				VSS
Unknowns	Cycle	Faults	user	system	wall clock	(MB)
700x700	first	0	10.2	1.4	78.6	54.1
	avg	0	4.6	.2	5.9	
720x720	first	10	10.8	1.0	81.7	57.0
	avg	0	4.9	.1	5.7	
730x730	first	3	11.0	1.0	81.2	58.5
	avg	0	5.0	.2	6.7	
740x740	first	223	11.9	4.4	158.9	59.9
	avg	225	5.4	1.2	38.0	
760x760	first	675	13.0	9.1	229.5	63.3
	avg	538	5.9	2.5	73.0	
780x780	first	1458	14.0	8.9	250.5	66.5
	avg	1277	6.4	4.3	88.0	
800x800	first	4635	15.8	22.5	437.0	70.0
	avg	2434	7.0	6.4	215.2	

Table 5.1: Page Fault Characterization of Multigrid for Large Systems.
Alliant FX/8 – 58.5MB Main Memory.

the pages in which U and F are located is not included. All timed experiments ran with no other users on the system. The page fault statistics were monitored with the UNIX system call **getrusage**. In part, we gathered timing information using the **etime** system call, which returns the elapsed user time and elasped system time with a resolution of .01 seconds. We monitored the wall clock time with the high-resolution timing routines provided in the FX/FORTRAN library, **hrcget** and **hrcdelta**, whose resolution is ten microseconds.

5.1 The Page Fault Behavior of Multigrid

First, we show how the computational efficiency degrades as the problem size grows when using the multigrid method[2]. The experimental results in Table 5.1 chart the page fault behavior as a function of the problem size. The problem size increases from 700 $\times$ 700 unknowns to 800 $\times$ 800 unknowns in steps of 10 to 20 unknowns in both the x and y directions. Each experiment consists of running six consecutive cycles. The data labeled *first* corresponds to the first cycle. We list it separately for two reasons: 1) in the multigrid method we compute the operators in the first cycle only; 2) during this cycle the operating system must set up any uninitialized page tables entries. The data labeled *avg* represents the average of the last five cycles. (We do not concern ourselves with convergence in the table, the purpose of the table is to display timing statistics only. The number five isn't meant to relate to convergence. Choosing six cycles for this experiment is not meant to imply anything about the convergence rate of multigrid.) The rightmost column lists the virtual set size (VSS) in units of megabytes (MB) for each process. The data presented reflects the outcome of a typical trial. The number of major page faults, and therefore the wall clock time, varies from one trial to the next. A major page fault happens when replacing a page results in a write to disk. One calculates the operator construction time by subtracting the average cycle user time from the first cycle user time.

A problem size that executes without major page faulting incurs nearly all its system time and extra wall clock time during the first cycle of the multigrid method. We incur the overhead of setting up the page tables on a once per process basis. For instance, the 700$\times$700 problem uses about 68 seconds of wall clock time beyond the computational time in the first cycle. This amounts to over 200% of the wall clock time to execute the last five cycles. We see the effect of Amdahl's Law; setting up page tables is highly sequential. Any competing method with similar memory requirements would incur a similar page table setup overhead.

On this instance of the hardware, for problems with less than 740$\times$740 unknowns

[2]We ran this set of experiments on the Alliant FX/8 at Argonne National Laboratory with 58.5 MB of memory partitioned for user space, running version 4.0 of the Alliant Concentrix operating system and version 4.0.24 of the Alliant FORTRAN compiler.

no page faults occur. Once major page faults start to occur, the wall clock time increases sharply and dramatically. This reveals the large spatial locality of the multigrid algorithm; with each cycle the computation sweeps through the whole virtual address space, not spending any prolonged period of time accessing any localized subset of the data. The time spent accessing the data associated with a particular grid point on any level is approximately the same throughout a cycle. Even though in a conventional multigrid algorithm 3/4 of the time is spent on the finest grid one cannot say the algorithm is exhibiting good spatial locality because during a visit to the finest grid, the majority of memory is accessed. The use of multicolor relaxation schemes, typically for vectorization and parallel processing purposes, makes the locality of reference even worse. The 740×740 problem exceeds the physical memory size by 1.4 MB, which exceeds the 14,600 physical pages set aside for users by around 350 pages. This 2.4% excess of address space causes the wall clock time per cycle to slow down by a factor of $5\frac{2}{3}$. A 19.6% excess causes the wall clock time per cycle to slow down by a factor of 32. The number of page faults per cycle almost doubles every time the multigrid problem size grows by 20 grid points in each direction beginning with 740×740 grid points on up to 800×800 grid points. Clearly the virtual machine is not so virtuous for this implementation the multigrid algorithm in this instance. Nonetheless, it does let us run the problem.

Inefficient Multiprocessing When Page Faulting

Table 5.2 compares page fault behavior for the 800×800 problem for different degrees of multiprocessing[3]. As in the previous experiments, we present the data from a representative trial. Using 4 CEs instead of 1 CE only decreases the trial's wall clock time by 16% overall and an average 25% per cycle. These results presented here should be compared to the quite favorable parallel processing performance results (for problem sizes where no page faults occurs) presented in Chapter 6. The wall clock time to complete the first cycle was nearly the same in all four cases. The greatest

[3]We ran this set of experiments on an Alliant FX/80, with 50.0 MB of memory partitioned for user space and four computational elements, running version 4.1.0 of the Alliant Concentrix operating system. We compiled the code with version 4.0.28 of the FORTRAN compiler.

deviation from the average wall clock time was 6% for the 3 CE trial. The system time for the first cycle decreased when using more than 1 CE, which gives a strong indication of the presence of some parallel work being done by the operating system itself. From this experiment we conclude that when major page faults are occurring little if any benefit is realized from having multiple processors available.

CEs	Outer Iteration	Major Page Faults	Time (seconds)		
			user	system	wall clock
1	first	1271	46.8	30.6	223.8
	avg(2-8)	734	26.8	8.2	93.8
	total	5144	234.8	88.5	882.3
2	first	1344	25.7	15.7	219.7
	avg(2-8)	929	15.1	10.2	93.9
	total	6506	131.5	87.7	880.4
3	first	1414	18.9	18.5	252.8
	avg(2-8)	753	10.9	6.2	78.8
	total	5259	95.7	61.9	806.3
4	first	1656	15.7	13.7	224.9
	avg(2-8)	717	9.1	5.1	70.4
	total	6677	79.5	50.0	739.7

Table 5.2: Ordinary Multigrid Method
Alliant FX/80 - 50 MB Main Memory.
800×800 Unknowns (VSS - 69.5 MB).

5.2 SAP-MG Outperforms Multigrid

Having seen the performance of multigrid, we now consider the SAP-MG method. We show the superiority of SAP-MG over multigrid for problems whose multigrid memory requirements exceed the size of physical memory. The computational cost of SAP-MG, measured in units of floating point operations exceeds multigrid's because of the combined effects of SAP-MG's slower convergence rate and SAP-MG doing more work per outer iteration. We hypothesize that once we load the data associated with a subdomain into memory, page faulting will temporarily cease and the computation will proceed efficiently.

To verify our hypothesis we solved the model problem on an evenly spaced, 800×800, grid with the SAP-MG method with four square subdomains that overlap each other by 50% as in Figure 4.1, SD4c. That is, the center square of SD4c, the intersection of all four subdomains is equal to 1/4 the size of one of the subdomains. Our inner iteration procedure was one multigrid V cycle with cycling parameters set to unity. The multigrid method had a VSS of 69.5 MB and therefore used all 50 MB of main memory[4]. The memory required for this SAP-MG algorithm was 36 MB, leaving 14 MB of main memory for other jobs. Already we see an advantage, this SAP-MG variant needed only 51% the memory multigrid needed.

Let us consider three variants of the SAP-MG algorithm and denote them 'A', 'B' and 'C'. In SAP-MG variant 'A' we assume our fine grid operator is the same at all grid points. Because of this property, if all the subdomains are of the same size and shape, then they can all share the same data for the difference and grid transfer operators; that is, these operators need to be computed and stored only for a set equal to the size of the subdomains. This only works if the differential equation and boundary conditions have constant coefficients.

In SAP-MG variant 'B' the operators are recomputed on entry to successive subdomains during each inner iteration throughout the whole computation. In variant 'B' we trade off the additional computational cost of reconstructing operators for

[4]We ran this set of experiments on an Alliant FX/80, with 50.0 MB of memory partitioned for user space and four computational elements, running version 4.1.0 of the Alliant Concentrix operating system. We compiled the code with version 4.0.28 of the FORTRAN compiler.

decreased memory needs. An application of this variant would be the case of non-constant coefficients, unevenly spaced meshes or irregularly shaped regions.

Finally, variant 'C', works under the assumption one has enough memory to store each of the subdomain operators completely. Variant 'C' maps well to smaller problems and distributed memory architectures. When considering our SAP-MG method for a distributed memory architecture, variant 'C', is the only method that has practical value. However, it is not the method of choice for parallelizing multigrid on distributed memory architectures because of its slow convergence rate as seen in §5.3. In Chan's and Tuminaro's survey of parallel multigrid algorithms [CT87], they describe the common methods for parallelizing on hypercube architectures. We hypothesize variant 'C' may be of some practical value. For instance, a Cray 2 computer has 64k double words of local memory. This is enough to divide up a 64×64 problem into four square subdomains, each of which could reside in the local memory of a processor for the duration of the problem. It is conceivable that due to the faster access time of the local memory, in spite of the slower convergence rate of the SAP-MG method — its larger outer iteration count, the outer iterations would be computed much faster and a total improvement may be realized. However, it was not in our best interest to pursue this idea and we consider variant 'C' no further.

We determine convergence of the SAP-MG method on the model problem by comparing the discrete L_2 norm of the residual to the norm of exact discretization error. We used the norm of exact discretization error instead of some arbitrary tolerance. The method stops doing the outer iteration procedure when the residual norm is less than the exact discretization error. We discussed the computational aspects of determining convergence in §2.8. Note, when we used the truncation error estimate method for the stopping criterion the problem converged in the same number of cycles. However, our computer code uses five point grid stencils on the finest grid level and nine point grid stencils on the coarser grid levels and as a result the truncation error estimate is not theoretically sound. The ideal determinator of convergence would be a method that dynamically computes the truncation error estimate and discrete L_2 norm of the residual.

Outer Iteration	Major Page Faults	Time (seconds)			Flops Per Unknown
		usr	sys	wall clock	
Ordinary Multigrid Method (VSS - 69.5 MB)					
first	1656	15.7	13.7	224.9	151
2	689	9.1	5.1	71.9	65
3	894	9.1	4.9	63.3	65
4	600	9.1	4.9	65.0	65
5	919	9.4	7.4	90.7	65
6	608	9.0	3.9	61.7	65
7	762	9.1	4.8	78.2	65
8	549	9.0	4.7	61.7	65
avg(2-8)	717	9.1	5.1	70.4	65
total	6677	79.5	50.0	739.7	605
SAP-MG — Four Subdomains (VSS — 36MB)					
Variant A					
first	0	17.0	0.9	24.9	152
avg(2-9)	0	14.3	0.0	14.3	115
total	0	131.5	1.0	139.5	1070
Variant B					
first	7	25.6	0.9	32.2	264
avg(2-9)	0	25.8	0.0	25.8	264
total	7	232.0	1.0	239.0	2384

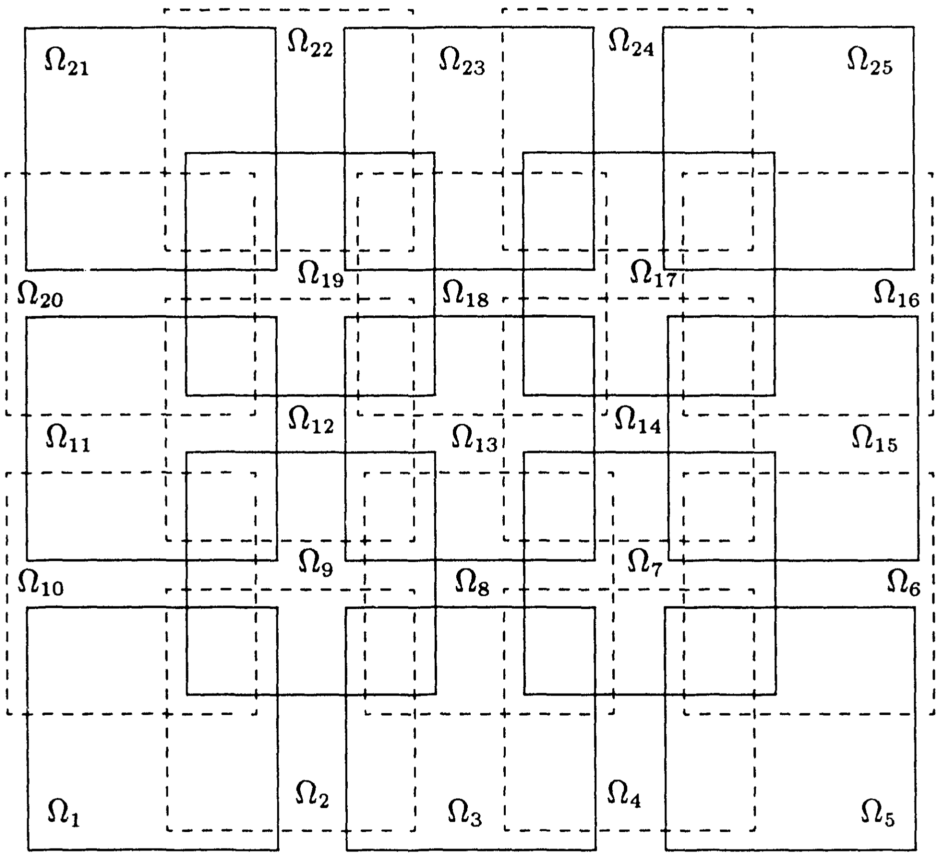
Table 5.3: Detailed Comparison of MG to SAP-MG.
Alliant FX/80 - 4 CEs - 50 MB Main Memory.
Inner Iteration Procedure is $V_{1,1,1,1}$. 800 × 800 Unknowns.

Table 5.3 reports results for SAP-MG variants 'A' and 'B'. Recall both these variants only allocate storage for the operators on one subdomain. SAP-MG variant 'A' is the method of choice when the finest grid difference operator is the same at all grid points. Thus, we compute the difference and grid transfer operators only once, before we enter the first subdomain. SAP-MG variant 'B' recomputes the operators on entry to every subdomain throughout the solving process. Variant 'B' is a more expensive procedure as constructing the operators on a subdomain requires slightly more time than doing a multigrid V cycle (*pbeg*, etc. all 1) on that subdomain. However, variant 'B' solves a wider class of problems.

Table 5.3 compares SAP-MG variant 'A' and SAP-MG variant 'B' to the ordinary multigrid method. The rows labeled *first, avg* and *total* are as described previously except for the rightmost column which now indicates the number of floating point operations per unknown required in the associated time frame. Table 5.3 includes the data for each multigrid cycle for ordinary multigrid to show the deviation in number of page faults from cycle to cycle. The time per cycle for the SAP-MG variants was constant to the tenth of a second.

For this problem the page fault behavior exceeds our expectations since the physical address space exceeds the SAP-MG method's virtual address space. Few, if any, major page faults occur in this case. The bottom line is a wall clock time of 140 seconds for SAP-MG variant 'A' versus 740 seconds for multigrid. SAP-MG variant 'A' obtains a "speedup" of 5.3 over multigrid for this particular problem even though the SAP-MG method requires 1.7 times more floating point operations per grid point to converge. SAP-MG variant 'B' obtains a "speedup" of 3.1 over multigrid for this particular problem even though the SAP-MG method requires 3.9 times more floating point operations per grid point to converge.

In conclusion, a SAP-MG algorithm is more efficient than ordinary multigrid for a class of large systems of equations. With SAP-MG the gains made from minimizing the main memory, disk and I/O resources used overshadow its increased operation count.



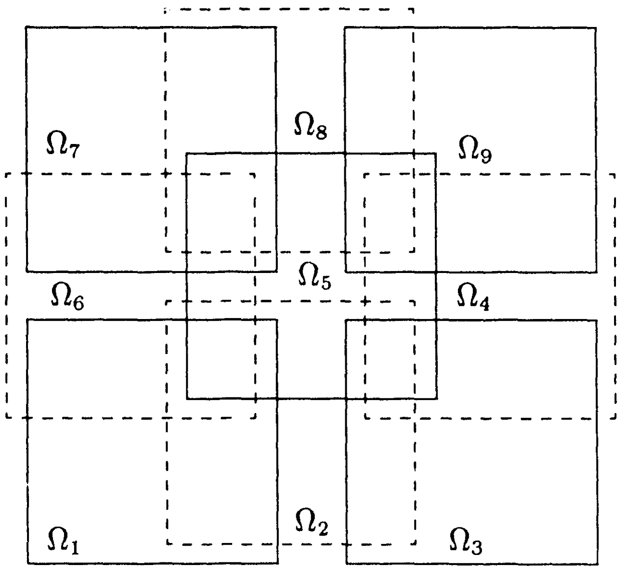


Figure 5.1: Schwarz Splittings for q = 3, 5.

5.3 Increasing the Limits of Linear System Size

In this section we show how domain decomposition allows us to solve problem sizes that the operating system cannot set up using multigrid[5]. In this last set of experiments we study the outer iteration count of the sequential SAP-MG for four different domain decompositions and for increasingly larger problems. We divide the global domain of the model problem into $q \times q$ square subdomains with $q = 2$, 3, 5 and 7. Figure 5.1 illustrates the domain decomposition for $q = 3$ and $q = 5$. The domains are numbered in the order they are visited during the outer iteration. Each subdomain overlaps its neighbors by 50%. We stop iterating when we meet the RMS ratio convergence criterion described in §2.7.2.

The results appear in Table 5.4. A *** in a field of the table means we could not solve that problem run using that domain decomposition on the given computer system. We could not solve it due to insufficient memory or insufficient swap space (or combination thereof). The combination of swap space on disk and incore memory was insufficient. We made no attempt to solve the problems with blank entries in the 5×5 and 7×7 column because of time constraints.

These experiments verify the assertion made in §4.3 that as the number of subdomains increases, and therefore the size of each subdomain decreases, the convergence rate per outer iteration becomes slower. Also, as the number of subdomains increases the convergence rate becomes less and less independent of the number of unknowns.

Using 4 subdomains, $q = 2$, with 50% overlap the largest problem that we solved with a high degree of computational efficiency (over 90%) was 800 × 800 on the given system. Solving the larger systems necessitated an increase in q. Even though the SAP-MG methods presented here seem to be computationally inefficient when compared to ordinary multigrid in the sense of needing greater outer iteration counts to converge, the SAP-MG method allows us to solve a problem that multigrid is unable to solve.

[5]Again we acknowledge the use of the Alliant FX/8 in the ACRF computing facility at Argonne National Laboratory with 58.5 MB of memory partitioned for user space, running Alliant Concentrix 4.0, using the Alliant FORTRAN compiler version 4.0.24.)

Sequential SAP–MG				
Domain Decomposition Squares	2x2	3x3	5x5	7x7
Global Unknowns	Outer Iteration Count			
200x200	7	11	22	26
300x300	7	12	23	
400x400	7	12	25	34
500x500	8	12		
600x600	8	13	26	38
700x700	8	13		
800x800	8	14	27	41
900x900	??	14	28	
1200x1200	***	14	29	45
1500x1500	***	***	29	

Table 5.4: Outer Iterations for Various Domain Decompositions versus # of Unknowns.

5.4 Poor Potential for Multiprocessing

In this section we use the results in Table 5.4 to show why parallel processing through the domain decomposition of the global domain into large grain subdomain tasks has been discounted. This sequential SAP-MG data gives us a gross indication of what types of parallel processing efficiencies might be expected. Here we quantify computational efficiency using operation counts. We make the following overly optimistic assumptions:

(1) The outer iteration count to convergence for sequential SAP-MG is a lower bound for any parallel SAP-MG variant.

(2) The work per outer iteration is constant for all SAP-MG domain decompositions and multigrid.

(3) Communication cost between processors is negligible.

(4) The data associated with each subdomain resides in the memory level closest to the processor(s) operating on it. That is, no read or write to disk occurs during the solving process.

Given that multigiid meets the RMS ratio convergence criterion of §2.7.2 in five cycles, then under these optimistic assumptions for $q = 7$ we obtain a dismal parallel processing efficiency for a 200×200 problem of about 20%. For larger problems the outer iteration count is even higher and the parallel processing efficiency is even lower. For instance, in the 1200×1200 unknowns problem the parallel processing efficiency diminishes to around 10%. The neglected computational overhead of 50% overlap would decrease these efficiencies by at least a factor of two. Thus, multiprocessing in this manner does not make sense.

Chapter 6

Parallelizing Multigrid With Microtasking

In this chapter we introduce parallelism into the multigrid algorithm through the use of microtasking. Microtasking is a fine grain parallel construct. Virtually all of the work perform[illegible] our multigrid code occurs in doubly nested, vectorizable loops. We review the major loops in §6.8. One can easily take advantage of do loop/microtasking parallelism in the multigrid algorithm. Coding such algorithms can be as easy as putting in a compiler directive that says, microtask here. In the future as compilers become more intelligent or new programming languages are adopted the procedure will simplify even more; programmers will need to make no alterations to their codes. Most compilers will detect the parallelism for the user and put in the microtasking code itself. In fact, the Alliant FORTRAN compiler does just that. One of the major advantages of this form of parallelism is the user doesn't worry about synchronization. We intend to show how effective the Alliant FORTRAN compiler and Alliant FX/8 Series hardware is at automatically parallelizing the **do loops** in the Boxmg multigrid code. Boxmg previously ran on a vector processor.

6.1 Alliant FX Series Computer Parallelism

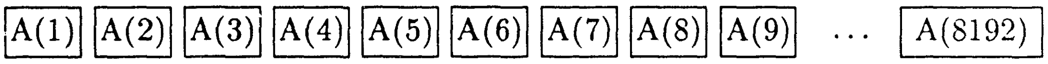
The four modes of **do loop** parallelism the Alliant FX/8 implements are **vector, concurrent, vector-concurrent** (VC) and **concurrent-outer-vector-inner** (COVI). *Vector* parallelism is the normal vector technique for a single processor. The Alliant FX/8 has a vector length of 32 elements. Processing 32 elements with a vector instruction is two to four times faster than processing the elements in scalar mode, depending on the instruction and degree of pipelining achieved in vector mode. Recall, Alliant refers to their enhanced vector CPU as a CE. An Alliant system can devote up to eight CEs (on an FX/8 or FX/80 model computer) to the execution of a single program. The FX/80 model has faster CEs and a larger cache than an FX/8 model. See the following page for an illustration of the modes of execution. They are discussed in more detail below.

In the *concurrent* mode of parallelism each of the p available processors takes the next available loop index and executes the code inside the loop in scalar mode in parallel with the other processors. Concurrent parallelism also makes it possible to call subroutines and functions in parallel.

The *vector-concurrent* mode of parallelism operates by dividing up the vector(s) to be processed across the p CEs available. For instance, CE 0 would process element 1 and every p-th element from then on of a given vector up to 32 elements on the first iteration. CE 1 would process element 2 and every p-th element from then on of a given vector up to 32 elements on the first iteration. In the case of an outer loop and an inner loop, only one of the outer loop indices is worked on at a time.

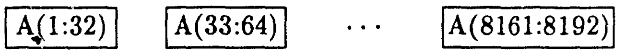
The *concurrent-outer-vector-inner* mode of parallelism is the method used to parallelize multiply nested loops with innermost loops that vectorize. Vector instructions from the inner loop are executed by each of the p available CEs in parallel. For example, an outer loop with an iteration index of four can put four CEs to work in parallel, each performing the work to be done in the whole inner loop corresponding to the particular outer iteration index. Let p be the number of processors available. Up to p indices of outer loop can be worked on at a time.

Scalar



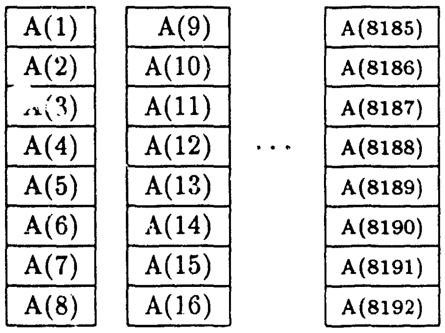
122472 cycles

Vector



24184 cycles

Concurrent



15970 cycles

Code can be written as a loop:

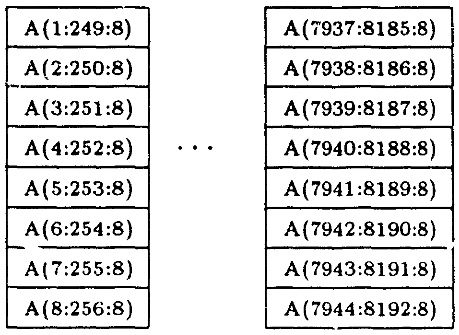
```
do I = 1, N
   A(I) = A(I) + S
enddo
```

Or as an array operation:

$$A(1:N) = A(1:N) + S$$

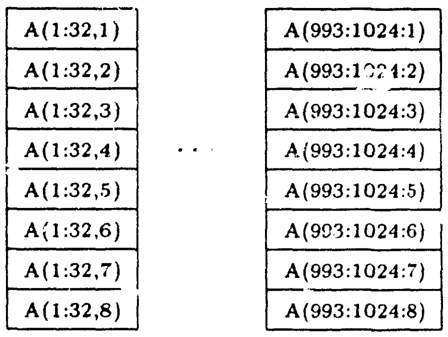
where $N = 8192$

Vector-concurrent



3848 cycles

Concurrent-outer-vector-inner



4825 cycles

Code can be written as a loop:

```
do J = 1, N
   do I = 1, N
      A(I,J) = A(I,J) + S
   enddo
enddo
```

Or as an array operation:

$$A(1:N,1:L) = A(1:N,1:L) + S$$

where $N = 8192$ and $L = 8$

6.2 Comments About Speedup

Many methods have evolved by which algorithms are compared. Currently it is popular to report the "speedup" achieved using multiple processors. Too much emphasis has been placed on how close an algorithm comes to achieving the optimal theoretical speedup in practice, usually with the number of processors equating to the optimal speedup.

An important factor often not taken into consideration is the cost of the CPUs involved. If a vendor sells a system a user at relatively low cost then one shouldn't care if they achieve parallel efficiencies of 90% to 95%. Vendors price their hardware to be cost competitive in their perceived market. The parallel efficiency trade-offs vary from computer to computer. On a relatively low cost computer a parallel inefficiency of 30% may be an acceptable loss. However, a smaller 10% parallel inefficiency on an expensize CPU such as those produced by Cray Research Inc. may be completely unacceptable. The bottom line is how big of a problem can be run in the time you are willing to wait for it to complete on the hardware you are willing to pay for.

Amdahl's Law states that for a program with serial work fraction s, the maximum parallel speedup obtainable is bounded by $1/s$. This law has led to the assertion that the serial fraction will dominate execution time for any large parallel ensemble of processors, limiting the advantage of the parallel approach. If P is the number of processors, s is the amount of time spent (by a serial processor) on serial parts of the program, and p is the amount of work spent (by a serial processor) on parts of the program that can be done in parallel, then Amdahl's law states

$$\text{Speedup } = \frac{(s+p)}{s+\frac{p}{P}} = \frac{(1)}{s+\frac{p}{P}} \tag{6.1}$$

where we have normalized total time $s+p=1$. For $P=1024$ this is a steep function of s near $s=0$ (slope of approximately $-P^2$). This expression is based on the implicit assumption that p is independent of P. However, one does not generally take a fixed sized problem and run it on various numbers of processors. In practice, a scientific computing problem scales with the available processing power. The fixed quantity is not the problem size but rather the amount of time a user is willing to wait for an answer. When given more computing power the user expands the problem to use the available hardware.

6.3 Ease of Parallel Implementation

First, the algorithm itself didn't need to be changed. All the same routines are called, in the same order. The cycling procedure stays the same, the test for convergence remains the same and no new data structures are needed. No new common areas are needed and no synchronization primitives need to be added or thought about by the programmer. Multigrid is quite often run on vector machines and code optimized for vectorization most likely compiles into microtasked, parallel code without modification. Obviously vector code always maps to VC mode on an Alliant FX Series computer. If the compiler doesn't automatically implement the **do loop** parallelism, it identifies the difficulty by putting a comment in the listing file. Generally inserting a simple compiler direction, in the form of a FORTRAN comment, allows the parallel mode to be used.

We made a minimal number of changes to our code to get all the parallelism available; the compiler does the majority of the work. We made two types of changes to the code. The first was due to a scalar reduction variable used in calculating the error. The simple and standard solution makes the reduction variable a vector of length corresponding to the outer loop and reduces the vector to a scalar after finishing the **do loops**. This needed to be done in three places and the compiler pointed out all three.

The other code alteration dealt with the way the code accesses the colored sets of points. The original code had four nested loops with two of the loops having length two, including the one concurrency would have been used on. As written the compiler couldn't figure out how it was possible to parallelize the code. One expected this loop to parallelize since it is the key loop of the whole program. By taking the color control logic cut of the loop and straightforwardly implementing the loop as at the end of this chapter the code parallelized into COVI mode.

We observed the following interesting phenomena. COVI mode is the compiler method of choice for parallelizing doubly nested loops. Even if the number of outer iteration indices is less than the number of potentially available processors the compiler still implements COVI mode. The compiler doesn't know at compile time the number of CEs to be used. If the vector length of the inner loop is long, VC mode would be more appropriate as all available processors could be put to work. In multigrid, COVI mode of execution is preferable to VC mode of execution because during COVI mode of execution the program displays better spatial locality for the doubly nested **do loops** as written, resulting in better cache performance.

Results presented in sections to follow show a moderate to high degree of success for the compiler-generated parallelism.

6.4 Microtasking Limitations

A number of algorithm idiosyncracies in multigrid can hinder the use of and efficiency of microtasking in speeding up multigrid. If the smoother isn't vectorizable VC parallelism obviously is not applicable. On a machine that only vectorizes stride one loops such as the NEC SX computer, circa 1987, the code is reduced to running in scalar mode unless a major overhaul of the data structures is undertaken. Furthermore, in addition to not being vectorizable, the smoother may not be parallelizable. As the number of processors grows the tendency is for the coupling between processors to decrease. In this case both the communication time to reach the shared variables and the contention for shared variables increases. Additionally, assuming a fixed problem size as the number of processors grows the work per processor on the inner loop decreases. This situation is mitigated by the fact that in many scientific applications the problem size scales with the computational power available.

6.5 Do Loop Parallelism Experiments

The multigrid algorithm consists of two logical phases. The first phase consists of setting up of the grid and interpolation operators. The second phase consists of using these operators to do the multigrid iteration. To test the parallel efficiency of the

Alliant architecture and its parallelizing compiler on the multigrid algorithm we ran the following set of experiments. We tested each logical phase separately. The setup phase testing was nearly the same as the conditions for testing the solving procedure listed in Table 6.1 with the exception that the type of solving cycle is irrelevant. One relaxation was done on each grid level.

Experimental Constants
Poisson's Equation
Two Color Relaxation On Finest Grid
Four Color Relaxation On Coarse Grids
Single User on System

Experimental Parameters

Types of Cycles
Setup
V
W

of Unknowns
64x64
128x128
256x256
512x512

Modes of Execution	
scalar	1 CE
vector	1 CE
concurrent	1-8 CEs
COVI/VC	1-8 CEs

Output Statistics
User, System and Wall Clock Time
Memory Use
of Page Faults

Table 6.1: Do Loop Parallelism Experiment

The Alliant FX/8 is a virtual memory machine. To avoid skew in the results we precede all timings by a cycle to assure the software and hardware page tables and segment table entries are set up.

CEs	Time (seconds)				Speedup			
	64x64	128x128	256x256	512x512	64x64	128x128	256x256	512x512
Scalar Mode of Execution								
1	35.4	162.3	161.6	160.9	—	—	—	—
Concurrent Mode of Execution								
1	59.7	157.4	157.5	157.4	1.00	1.00	1.00	1.00
2	30.1	79.3	79.5	79.4	1.98	1.98	1.98	1.98
3	21.4	55.2	54.3	53.8	2.79	2.85	2.90	2.92
4	15.5	40.6	40.7	40.7	3.86	3.87	3.86	3.87
5	13.9	34.2	33.6	33.3	4.30	4.60	4.68	4.77
6	11.8	29.2	28.8	28.2	5.04	5.39	5.46	5.58
7	10.7	26.3	25.5	24.7	5.60	5.98	6.17	6.37
8	8.3	21.5	21.8	21.7	7.18	7.32	7.19	7.25
Vector Mode of Execution								
1	13.67	53.8	51.8	50.4	2.59	3.02	3.12	3.19
COVI/VC Mode of Execution								
1	13.6	52.2	50.1	49.3	1.00	1.00	1.00	1.00
2	7.2	26.7	25.7	25.4	1.90	1.96	1.95	1.94
3	5.1	18.9	18.0	17.5	2.66	2.76	2.78	2.82
4	4.5	14.2	13.7	13.6	2.99	3.68	3.65	3.62
5	3.4	12.3	11.5	11.3	3.97	4.24	4.36	4.36
6	3.0	10.7	10.1	9.8	4.59	4.88	4.96	5.03
7	2.6	9.8	9.0	8.9	5.24	5.32	5.57	5.54
8	2.1	8.3	8.1	8.1	6.44	6.29	6.18	6.09
(R)	(64)	(64)	(16)	(4)				

Table 6.2: Do loop parallelism results for setup phase of multigrid.

6.6 Do Loop Parallelism Results

Tables 6.2, 6.3 and 6.4 show the experimental results for testing the degree of **do loop** parallelism inherent in the multigrid algorithm. We used two-color Gauss-Seidel point relaxation on the finest grid and four-color Gauss-Seidel, point relaxation on the coarse grids. On the coarsest grid we solved directly. Column one gives the number of CEs applied to the problem. Listed in columns two through five are the times in seconds for an experiment. The bottom line of the tables, labeled (R), gives the number of times the cycle was repeated during the experiment. Columns six through nine report the observed speedup. The speedup for vector mode is the ratio of vector-only to scalar (1 CE). The speedups for multiple CE's are the ratio of the execution times of n CEs over 1 CE — executing in the same parallelism mode. The scalar times for the cycling procedures are significantly better than the concurrent times when executing on 1 processor, especially for the 64×64 problem. For example, we list the percentage extra time spent running concurrent mode for four experiments,

$$W_{64} : 42\% \quad W_{128} : 14\% \quad V_{64} : 55\% \quad V_{128} : 20\%.$$

We attribute the differences to multiprocessing overhead. The percentage overhead is greatest for the smallest problem as expected. The concurrency overhead is amortized as the amount of computation work becomes greater.

Interestingly enough the same phenomena doesn't appear in the COVI/VC mode of execution compared to the vector only mode. We speculate that the COVI/VC mode of execution might be exhibiting bette spatial locality and has better cache performance which offsets the overhead due to multiprocessing synchronization.

Comparing the vector-only results to the scalar results for both V and W cycles we see healthy speedups. The V cycles vectorize 10% better than the W cycles for the smaller problems, 64×64, 128×128 and 2-3% better for the 256×256 and 512×512 problems.

CEs	Time (seconds)				Speedup			
	64x64	128x128	256x256	512x512	64x64	128x128	256x256	512x512
Scalar Mode of Execution								
1	32.7	144.4	144.2	142.9	—	—	—	—
Concurrent Mode of Execution								
1	51.1	173.9	154.1	157.4	1.00	1.00	1.00	1.00
2	26.6	88.5	78.2	79.2	1.92	1.96	1.97	1.99
3	19.3	61.5	53.5	53.2	2.68	2.83	2.88	2.96
4	14.4	45.6	40.3	40.4	3.54	3.81	3.82	3.90
5	13.0	38.6	33.4	33.0	3.92	4.50	4.61	4.77
6	11.5	33.5	28.6	28.0	4.45	5.19	5.39	5.62
7	10.5	29.7	25.3	24.6	4.84	5.85	6.09	6.40
8	8.8	24.8	21.8	22.1	5.77	7.01	7.07	7.12
Vector Mode of Execution								
1	12.0	44.9	44.6	44.2	2.72	3.22	3.22	3.23
COVI/VC Mode of Execution								
1	11.9	46.0	44.3	43.6	1.00	1.00	1.00	1.00
2	6.4	24.6	23.4	23.3	1.87	1.87	1.89	1.87
3	4.8	17.8	16.7	16.4	2.46	2.58	2.65	2.66
4	3.7	13.5	13.1	12.9	3.18	3.41	3.38	3.38
5	3.3	12.0	11.3	11.1	3.62	3.83	3.92	3.93
6	2.9	10.5	10.1	9.9	4.14	4.38	4.38	4.40
7	2.7	10.0	9.4	9.2	4.40	4.60	4.71	4.74
8	2.3	8.9	8.6	8.7	5.23	5.17	5.15	5.01
(R)	(64)	(64)	(16)	(4)				

Table 6.3: Do loop parallelism results for multigrid V cycle.

	Time (seconds)				Speedup			
CEs	64x64	128x128	256x256	512x512	64x64	128x128	256x256	512x512
Scalar Mode of Execution								
1	48.3	212.2	214.0	213.3	—	—	—	—
Concurrent Mode of Execution								
1	70.0	242.8	225.3	227.0	1.00	1.00	1.00	1.00
2	37.0	123.7	114.8	115.3	1.62	1.96	1.96	1.97
3	27.4	88.8	80.6	79.6	2.19	2.73	2.80	2.85
4	20.5	65.0	59.7	60.0	2.90	3.74	3.77	3.78
5	19.1	57.2	50.8	49.9	3.13	4.24	4.43	4.55
6	17.1	50.3	43.8	42.7	3.50	4.83	5.14	5.32
7	16.0	45.6	39.5	38.3	3.72	5.32	5.70	5.93
8	13.3	37.0	33.2	33.1	4.50	6.56	6.79	6.85
Vector Mode of Execution								
1	19.4	72.4	68.3	67.4	2.48	2.93	3.13	3.16
COVI/VC Mode of Execution								
1	19.9	72.3	67.9	65.9	1.00	1.00	1.00	1.00
2	10.8	39.0	36.3	35.0	1.84	1.85	1.87	1.88
3	8.6	29.7	26.6	25.2	2.36	2.43	2.55	2.61
4	6.3	23.1	21.2	19.7	3.15	3.12	3.20	3.35
5	6.1	21.0	18.1	17.2	3.27	3.44	3.75	3.83
6	5.6	19.3	16.4	15.4	3.53	3.75	4.14	4.28
7	5.5	18.4	15.4	14.3	3.64	3.93	4.41	4.61
8	4.6	15.9	13.8	13.3	4.32	4.54	4.92	4.95
(R)	(64)	(64)	(16)	(4)				

Table 6.4: Do loop parallelism results for multigrid W cycle.

6.6.1 Comparison of Multiprocessing Efficiency of the W cycle and the V cycle

Table 6.5 compares the multiprocessing efficiency of the W cycle and the V cycle. When using W cycles the average vector length over a cycle is smaller because a greater Thus, there has been concern in the multigrid literature about the use of the W cycle on high performance vector supercomputers. In machines with a global main memory and no local cache beyond that of the vector registers, longer vector length results in improved better performance. Machines that fit into this class are the CDC Cyber computers and the Cray XMP.

	Concurrent				COVI/VC			
CEs	64	128	256	512	64	128	256	512
2	18.5	0	.5	0	0	1.0	1.0	-.5
3	22.4	3.7	2.8	3.8	4.2	6.2	3.9	1.9
4	22.1	1.9	1.3	3.2	1.0	9.3	5.6	.9
5	25.2	6.1	1.0	4.8	10.7	11.3	4.5	2.6
6	27.1	7.4	4.1	5.6	17.3	10.4	5.8	2.8
7	30.0	9.9	4.9	7.9	20.9	17.0	6.8	2.8
8	28.2	6.8	4.1	3.9	21.1	13.5	4.7	1.2

Table 6.5: $\left(\frac{V_{speedup}}{W_{speedup}} - 1\right) \times 100$, Comparison of V cycle and W cycle microtasking efficiency.

Memory Hierarchy

On a shared memory machine with an additional level of memory hierarchy between the registers and main memory, (a cache), longer vectors do not necessarily imply better performance. The code is said to be executing from memory when the majority of reads do not result in a cache hit. When the code is exhibiting good spatial locality and referencing from cache often it is said to be executing from cache. [ASM86] studied the performance of common math expressions as a function of vector length. They found that on the Alliant FX/8 executing from memory was up to 6 times slower than executing from cache. Long vectors tend to overflow the cache and no longer reside in the cache the next time they are referenced.

The results of Table 6.5 do not include any set up work, only the time spent during the cycling procedure is included. Table 6.5 shows a significant difference in microtasking efficiency for very small problems. As problems get larger the difference fades. Note, we are comparing the multiprocessing efficiency only. We are not stating W cycles are less efficient, and therefore should not be used. The benefits of W cycles go beyond the computational efficiency as W cycles reduce the error in a different fashion than V cycles. For those who prefer using W cycles, the results show for all but the smallest problems they can do so without contemplating the tradeoff of computational efficiency on this architecture.

For a 512×512 problem the vector length in the relaxation scheme on increasingly coarser grids is 256, 128, 64, 32, 16 and 8 assuming one CE per inner loop (COVI). If the inner loops where coded for VC mode then the vector lengths are 32, 16, 8, 4, 2 and 1. If the whole relaxation can take place in cache it makes sense to avoid VC mode because of the shorter vectors. The average computation efficiency in vector mode is 12% worse for W cycles in the 128×128 problem but the difference diminishes to 2% for the larger 512×512 problem. The W cycle efficiency approaches that of the V cycle efficiency as the problem size increases to the largest that can fit in memory without incurring page faults.

6.6.2 Do Loop Parallelism Conclusions

We conclude that multigrid with colored relaxation parallelizes well on the Alliant FX Series computer. Some may argue a speedup of five out of a possible eight is poor. However, the cost of adding processors seven and eight in an eight processor system is relatively small compared to the cost of purchasing the first few processors. Also, the loops can be rewritten to utilize the cache more efficiently, resulting in better code performance. We did not do this here because we wanted to demonstrate the ease with which we can introduce parallelism into the multigrid algorithm.

6.7 Reasons for Imperfect Speedup

Programs don't show perfect speedup for many reasons. If we know the potential causes of slow-down, then we can analyze the program and attempt to minimize their effects. We see the effects of each of the following in the experimental results presented in Tables 6.2, 6.3 and 6.4.

Load Imbalance

This situation occurs in the COVI mode when the number of loop iterations, m, is not an even multiple of the number of CPUs, p. In the worst case $\mathrm{mod}(m,p) = 1$ and when the end of the outer loop is reached, $p-1$ CPUs wait while a single CPU works on the last of the loop indices. COVI loops with iteration indices less than the number of processors are a special case of the above. Executing sequential code can also be classified as load imbalance since only one processor executes the sequential code, and the others wait. Another type of imbalance occurs when the vector length is not long enough to amortize the vector startup costs.

Contention for Memory

Another factor is memory bandwidth. As the number of processors increases the data paths to/from memory are more likely to become saturated. For instance, less than perfect speedups while multiprocessing with all four CPUs of a Cray XMP/4 have been attributed to bank conflict and memory path saturation. The concurrent mode of execution shows greater speedup than the COVI/VC mode of execution because in COVI/VC mode data is being accessed at a rate about 2.5 times faster than in concurrent mode, possibly running into bus limitations.

Synchronization Overhead

Another factor is the cost for synchronizing the contention for available loop indices. Some type of synchronization method is required. The loop indices can be scheduled statically or dynamically. When a concurrency loop is reached the processors could dynamically request the next available iteration index. The advantage of this method is potentially better load balancing. The disadvantage of this method is that the index counter must be encased in a critical section increasing synchronization costs. For eight CPUs and a shared memory architecture this works reasonably well for the amount of work contained in the multigrid loops. In the VC mode on the Alliant FX/8, when the loop is reached each processor is assigned a *vpn*, virtual process number. Each processor then proceeds to do the computation for every *p*th vector element. Thus, VC mode statically allocates iterates.

6.8 Code Listing For Major Multigrid Loops

Finally, we list the loops in which the majority of the floating point operations in our multigrid algorithm occur.

```
C     9 point, 4 color relaxation
       do 900 icolor = 1 , 4
          do 900 j = jbeg(icolor) , j1 , 2
             do 900 i = ibeg(icolor) , i1 , 2
          q(i,j) = (  w(  i,   j) * q(i-1,j  ) + qf(i,j)
     $              + w(i+1,   j) * q(i+1,j  )
     $              + s(  i,   j) * q(  i,j-1)
     $              + s(  i,j+1) * q(  i,j+1)
     $              +sw(  i,   j) * q(i-1,j-1)
     $              +sw(i+1,j+1) * q(i+1,j+1)
     $              +nw(i+1,   j) * q(i+1,j-1)
     $              +nw(  i,j+1) * q(i-1,j+1) )*msor(i,j)
  900    continue
C    Compute residuals.
       do 10 icolor = 1 , 3
          do 10 j = jbeg(icolor) , j1 , 2
             do 10 i = ibeg(icolor) , i1 , 2
        mtot(i,j)  = qf(  i,j   ) - q(  i,   j) * o(i,j)
     $             +  w(  i,   j) * q(i-1,j  )
     $             +  w(i+1,   j) * q(i+1,j  )
     $             +  s(  i,   j) * q(  i,j-1)
     $             +  s(  i,j+1) * q(  i,j+1)
     $             + sw(  i,   j) * q(i-1,j-1)
     $             + sw(i+1,j+1) * q(i+1,j+1)
     $             + nw(i+1,   j) * q(i+1,j-1)
     $             + nw(  i,j+1) * q(i-1,j+1)
   10        aerr(j-1) = aerr(j-1) + mtot(i,j)**2
```

Relaxation loops: turning the scalar reduction variable into a vector allows the loop to compile in COVI mode.

```
      do 10 ic = 3 , iicf1
         if = if+2
         q(if,jf) = q(if,jf)+qc(ic,2)
         a = ci(ic,2,lr)*qc(ic,2)+ci(ic,2,ll)*qc(ic-1,2)
         q(if-1,jf) = q(if-1,jf)+a+sor(if-1,jf,mtot)
   10 continue
      .... code for some edge
      do 20 jc =3 , jjcf1
         jf = jf+2
         if = 2
         q(2,jf) = q(2,jf)+qc(2,jc)
         aq = ci(2,jc,la)*qc(2,jc)+ci(2,jc,lb)*qc(2,jc-1)
         q(2,jf-1) = q(2,jf-1)+aq+mtot(2,jf-1)
         do 20 ic = 3 , iicf1
            if = if+2
            q(if,jf) = q(if,jf) + qc(ic,jc)
            a = ci(ic,jc,lr)*qc(ic,jc) + ci(ic,jc,ll)*qc(ic-1,jc)
            q(if-1,jf) = q(if-1,jf) + a + mtot(if-1,jf)
            aq = ci(ic,jc,la)*qc(ic,jc) + ci(ic,jc,lb)*qc(ic,jc-1)
            q(if,jf-1) = q(if,jf-1) + aq + mtot(if,jf-1)
            a = ci(ic-1,jc-1,lsw) * qc(ic-1,jc-1)
     1        + ci(ic-1,jc-1,lnw) * qc(ic-1,jc)
     1        + ci(ic-1,jc-1,lne) * qc(ic,jc)
     1        + ci(ic-1,jc-1,lse) * qc(ic,jc-1)
           q(if-1,jf-1) = q(if-1,jf-1) + a + mtot(if-1,jf-1)
   20 continue
```

Interpolate and add loops: Cray version of code compiled without modification. Loops compiled to COVI mode.

```
      jf = 0
      do 80 jc = 2,jjc1
         jf = jf+2
         if = 0
         do 80 ic = 2,iic1
            if = if+2
      qfc(ic,jc) =  lne(ic-1,jc-1) * mtot(if-1,jf-1)
     $           + la(ic  ,jc  ) * mtot(if  ,jf-1)
     $           +lnw(ic  ,jc-1) * mtot(if+1,jf-1)
     $           + lr(ic  ,jc  ) * mtot(if-1,jf  )
     $           +                 mtot(if  ,jf  )
     $           + ll(ic+1,jc  ) * mtot(if+1,jf  )
     $           +lse(ic-1,jc  ) * mtot(if-1,jf+1)
     $           + lb(ic  ,jc+1) * mtot(if  ,jf+1)
     $           +lsw(ic  ,jc  ) * mtot(if+1,jf+1)

   80 continue
```

Relative truncation error loops: Cray version of code compiled with one minor modification. The compiler pointed out a reduction variable and the standard change allowed the loop to compile into COVI mode.

Operator construction loops: Cray version of code compiled **without modification.** All loops compiled to COVI mode. The code for operator construction loops is many pages long so we summarize the vector content of the loops in the following table. The iteration order relates to the number of grid points the computation takes place for, iif is the number of grid points on the finest grid level; iic is the number of grid points on coarser grid levels. Listed in the "vector instructions" column is the number of vector instructions that the compiler generated for the loop, excluding vector reads and writes.

loop number	vector* instructions	iteration order	purpose
10	2	iif	msor, mtot
240	16	iic	rhs
5-pt			
160	20	iic	lr, ll
190	20	iic	la, lb
210	39	iic	lsw,lse,lnw,lne
220	165	iic	w,sw,s,nw,or,msor
9-pt			
40	24	iic	lr, ll
80	24	iic	la, lb
90	54	iic	lsw,lse,lnw,lne
110	133	iic	w,sw,s,nw
120	107	iic	or, msor

Table 6.6: Operator construction. * doesn't include memory references.

Chapter 7

Conclusion

In this thesis we solved an elliptic model problem using variants of the Schwarz Alternating Procedure. We chose the multigrid method for the inner iteration procedure. Also, we ran experiments to test the multiprocessing efficiency of the multigrid multigrid method on Alliant FX Series computers.

The standard rationalization for using the SAP is solving problems on irregularly shaped domains. But we showed that the SAP can also be useful for solving problems with a large number of unknowns when combined with a multigrid inner iteration procedure. We used domain decomposition to limit the spatial locality to an area that can fit into main memory, thus avoiding page faults. The same principle applies for any multilevel memory hierarchy. Thus, even though SAP-MG executes more floating point operations, it can execute faster than multigrid in some multilevel memory hierarchies..

We presented results that show why parallel processing the SAP-MG method across subdomains generally will not result in an algorithm that runs any faster that the multigrid method itself run on a single processor. Then, we showed how the Alliant compiler and FX Series computers efficiently parallelize the multigrid method (multigrid with colored point relaxation). Furthermore, we showed the task of parallelizing in this manner is trivial.

Appendix A

Locality of Reference

The following comments about locality of reference are taken from [Dei84]. The concept of locality tells us that programs tend to reference storage in nonuniform, highly localized patterns. Locality manifests itself both in time and space. *Temporal locality* is locality over time. *Spatial locality* means that nearby items tend to be similar. It means a process will tend to concentrate its references in a time interval to a particular subset of its page, of its virtual data. Actually, locality is quite reasonable in computer systems, when one considers the way programs are written and data is organized. In particular,

1. Temporal locahty means storage locations referenced recently are likely to be referenced in the near future. Supporting this observation are

   a) looping,

   b) subroutines,

   c) stacks, and

   d) variables used for counting and totaling.

2. Spatial locality — means that storage references tend to be clustered so that once a location is referenced, it is highly likely that nearby locations will be referenced. Supporting this observation are

   a) array traversals,

   b) sequential code execution, and

   c) the tendency of programmers to place related definitions near one another.

Deitel states "Perhaps the most significant consequence of storage refence locality is that a program can run efficiently as long as its favored subset of pages is in primary storage."

Appendix B

Local Mode (Fourier) Analysis

The following discussion is taken directly from [Bra77]. Suppose we are interested in solving the partial differential equation

$$LU(x,y) \equiv a\frac{\partial^2 U(x,y)}{\partial x^2} + c\frac{\partial^2 U(x,y)}{\partial x^2} = F(x,y) \tag{2.1}$$

with some suitable boundary conditions. Denoting U^k and F^k approximations of U and F, respectively, on the grid G^k, the usual second-order discretization of (2.1) is

$$L^k U^k_{\alpha,\beta} \equiv a\frac{U^k_{\alpha+1,\beta} - 2U^k_{\alpha,\beta} + U^k_{\alpha-1,\beta}}{h_k^2} + c\frac{U^k_{\alpha,\beta+1} - 2U^k_{\alpha,\beta} + U^k_{\alpha,\beta-1}}{h_k^2} = F^k_{\alpha,\beta}, \tag{2.2}$$

where

$$U^k_{\alpha,\beta} = U(\alpha h_k, \beta h_k), \quad F^k_{\alpha,\beta} = F(\alpha h_k, \beta h_k); \quad \alpha, \beta \text{ integers.}$$

(In the context of multigrid it is important to define the difference equations in this divided form, without, for example, multiplying throughout by h_k^2, in order to get the proper relative scale at the different level.) Given an approximation u to U^k, a simple example of a relaxation scheme to improve it is the following.

Gauss-Seidel Relaxation. The points (α, β) of G^k are scanned one by one in some prescribed order; e.g., lexicographic order. At each point the value $u_{\alpha,\beta}$ is replaced by a new value $\bar{u}_{\alpha,\beta}$ such that (2.2) is satisfied. That is $\bar{u}_{\alpha,\beta}$ satisfies

$$a\frac{u^k_{\alpha+1,\beta} - 2\bar{u}^k_{\alpha,\beta} + \bar{u}^k_{\alpha-1,\beta}}{h_k^2} + c\frac{u^k_{\alpha,\beta+1} - 2\bar{u}^k_{\alpha,\beta} + \bar{u}^k_{\alpha,\beta-1}}{h_k^2} = F^k_{\alpha,\beta}, \tag{2.3}$$

where the new values $\bar{u}_{\alpha-1,\beta}$, $\bar{u}_{\alpha,\beta-1}$ are used since, in the lexicographical order, by the time (α, β) is scanned new values have already replaced old values at $(\alpha - 1, \beta)$, and $(\alpha, \beta - 1)$.

A complete pass, scanning in this manner all the points of G^k, is called a (Gauss-Seidel lexicographical) G^k *relaxation sweep.* The new approximation $\bar{u}$ does not necessarily satisfy (2.2), and further relaxation sweeps may be required to improved it. An important quantity therefore is the *convergence factor*, μ say, which may be defined by

$$\mu = \|\bar{v}\| \,/\, \|v\|, \quad \text{where } v = U^k - u, \quad \bar{v} = U^k - \bar{u}, \tag{2.4}$$

$\|\cdot\|$ being any suitable discrete norm.

The rate of convergence of the above relaxation scheme is asymptotically very slow. That is, except for the first few relaxation sweeps we have $\mu = 1 - O(h_k^2)$. This means that we have to perform $O(h_k^{-2})$ relaxation sweeps to reduce the error an order of magnitude.

In the multigrid method, however the role of relaxation is not to reduce the error but to smooth it out; i.e., to reduce the high-frequency components of the error (the lower frequencies being reduced by relaxation sweeps on coarser grids). In fact, since smoothing is basically a local process, (high frequencies have short coupling range), we can analyze it in the interior of G^k by (locally) expanding the error in Fourier series. This allows us to study separately the convergence rate of each Fourier component, and in particular, the convergence rate of high frequency components, which is the rate of smoothing.

Thus to study the $\theta = (\theta_1, \theta_2)$ Fourier component of the error functions v and $\bar{v}$ before and after the relaxation sweep, we put

$$v_{\alpha,\beta} = A_\theta e^{i(\theta_1\alpha+\theta_2\beta)} \quad \text{and} \quad \bar{v}_{\alpha,\beta} = \bar{A}_\theta e^{i(\theta_1\alpha+\theta_2\beta)}. \tag{2.5}$$

Subtracting (2.2) from (2.3), we get the relation

$$a(v_{\alpha+1,\beta} - 2\bar{v}_{\alpha,\beta} + \bar{v}_{\alpha-1,\beta}) + c(v_{\alpha,\beta+1} - 2\bar{v}_{\alpha,\beta} + \bar{v}_{\alpha,\beta-1}) = 0, \tag{2.6}$$

from which, by (2.5)

$$(ae^{i\theta_1} + ce^{i\theta_2})A_\theta + (ae^{-i\theta_1} + ce^{-i\theta_2} - 2a - 2c)\bar{A}_\theta = 0. \tag{2.7}$$

Hence the convergence factor of the θ component is

$$\mu(\theta) = \left|\frac{\bar{A}_\theta}{A_\theta}\right| = \left|\frac{ae^{i\theta_2} + ce^{i\theta_2}}{2a + 2c - ae^{-i\theta_1} - ce^{-i\theta_2}}\right| \tag{2.8}$$

Define $|\theta| = \max(|\theta_1|, |\theta_2|)$. In domains of diameter $O(1)$ the lowest Fourier components have $|\theta| = O(h_k)$, and their convergence rate therefore is $\mu(\theta) = 1 - O(h_k^2)$. Here, however, we are interested in the *smoothing factor*, which is defined by

$$\bar{\mu} = \max_{\rho\pi \le |\theta| \le \pi} \mu(\theta), \tag{2.9}$$

where $\hat{\rho}$ is the mesh-size ratio and the range $\rho\pi \leq |\theta| \leq \pi$ is the suitable range of high-frequency components, i.e., the range of components that cannot be approximated on the coarser grid, because its mesh-size is $h_{k-1} = h_k/\hat{\rho}$. We will assume here that $\hat{\rho} = \frac{1}{2}$, which is the usual ratio.

Consider first the case a = c (Poisson equation). A simple calculation shows that $\bar{\mu} = \mu(\pi/2, \arccos 4/5) = .5$. This is a very satisfactory rate; it implies that *three relaxations sweeps reduce the high-frequency error-components by almost an order of magnitude.* Similar rates are obtained for general a and c, provided a/c is of moderate size.

The rate of smoothing is less remarkable in the degenerate case $a \ll c$ (or $c \ll a$). For instance

$$\mu(\frac{\pi}{2}, 0) = \sqrt{\frac{a^2 + c^2}{a^2 + (c + 2a)^2}}, \tag{2.10}$$

which approaches 1 as $a \to 0$. Thus, for problems with such degeneracy, Gauss-Seidel relaxation is not a suitable smoothing scheme. But other schemes exist. For example,

Line Relaxation. instead of treating each point (α, β) of G^k separately, one takes simultaneously a line of points at a time, where a line is a set of all points (α, β) in G^k with the same α (a vertical line). All the values $u_{\alpha,\beta}$ on such a line are simultaneously replaced by new values $\bar{u}_{\alpha,\beta}$ which simultaneously satisfy all the Eqs. (2.2) on that line. (This is easy and inexpensive to do, since the system of equations to be solved for each such line is a tridiagonal, diagonally dominant system. See, e.g., in [Wac66].) As a result, we get the same relation as (2.3) above, except that $u_{\alpha,\beta+1}$ is replaced by $\bar{u}_{\alpha,\beta+1}$. Hence, instead of (2.8) we get:

$$\mu(\theta) = \left| \frac{a}{2(a + c - c\cos\theta_2) - ae^{-i\theta_1}} \right| \tag{2.11}$$

from which one can derive the smoothing factor

$$\bar{\mu} = \max\{5^{-\frac{1}{2}}, \frac{a}{a + 2c}\}, \tag{2.12}$$

which is very satisfactory, even in the degenerate case $a \ll c$.

Appendix C

Details of Memory Requirements

To simplify the discussion of memory considerations we logically split the memory requirements into operator space, problem space, and work space. We define some parameters associated with the memory requirements. NX and NY refer to the number of grid points inclusive of the boundaries in the x and y directions respectively. NXY refers to the number of grid points in the subdomain including the boundary. $NFMAX$ is the total number of grid points in the finest grid and its corresponding coarser grids for one logical storage entity. Let $NCMAX = NFMAX - NXY$.

Although BOXMG incurs a storage penalty because it uses nine-point coarse grid and grid transfer operators its storage requirements are less than or equivalent to that of its conjugate gradient competitors.

Operator space

Operator space contains the storage necessary for the difference operators on all M grid levels and the $M-1$ grid transfer operators (in the code, the $M-1$ prolongation and $M-1$ restriction operators are derived from the same data). A five stripe, symmetric difference operator requires $3 \times NFMAX + 2 \times NCMAX$ memory locations. A nine stripe, symmetric difference operator requires $5 \times NFMAX$ memory locations. The matrices storing the grid transfer operators require $8 \times NCMAX$ locations. All operators remain fixed until the problem meets the convergence criterion.

Problem space

The problem space consists of the data that defines the problem on the finest grid and the solution. The code solves equations of the form

$$-\nabla \cdot (D(x,y)\nabla U(x,y)) + \sigma(x,y)U(x,y) = F(x,y), \quad (x,y) \in \Omega. \qquad (3.1)$$

Each of the matrices D, U, σ and F require NXY storage locations. The matrices D, σ and F are given at the beginning of the problem and U is solved for. The code uses D and σ to build the fine grid difference operator which are then discarded provided we permanently store the fine grid operator. We only use σ to calculate the center point of the difference stencil. The center point of the difference star (ko) is only a function of the grid, D and σ. We completely avoid allocating storage for σ by storing it in ko initially. We then use σ to form ko simultaneously overwriting σ. We avoid allocating storage for D in a similar manner. After initialization the combined storage requirements for the problem space is $2 \times NXY$.

Also, we include the data structures that define the domains in the problem space.

Work Space

The code holds residuals and element by element inverses of various matrices in temporary storage during the multigrid cycle. The memory requirements for arrays of this type that span all M grid levels are $NFMAX$ for each of msor, mtot and msos. We only allocate the storage for msos when using a line relaxation smoothing procedure. The memory requirements for arrays that span the $M-1$ coarse grid levels are $NCMAX$ for corrections and $NCMAX$ for right hand sides. Using a direct solve on the coarsest grid requires the arrays bbd(NXC×NYC) and abd(NXC+2,NXC×NYC). The parameters NXC and NYC are the number of unknowns in their respective directions on the coarsest grid.

Having defined the logical sections of memory we present a chart of problem, operator and work space for various domain sizes.

n_h	5 stripe operator	9 stripe operator	problem	work	5-all	9-all	(9-line)
40	12.6	17.3	3.1	8.4	24.2	28.9	31.2
50	18.8	25.9	4.9	12.7	36.4	43.5	47.0
60	26.4	36.5	7.0	18.0	51.4	61.5	66.5
70	35.2	48.8	9.6	24.1	68.8	82.4	89.2
80	45.8	63.5	12.5	31.3	89.7	107.	116.
90	56.9	79.0	15.8	39.1	111.	134.	145.
100	70.1	97.3	19.5	48.1	137.	165.	178.
150	153.2	213.4	43.9	106.1	303.	363.	393.
200	270.2	376.8	78.1	187.4	535.	642.	695.
250	418.0	583.4	122.1	290.7	830.	996.	1078.
300	599.9	837.7	175.8	417.5	1193.	1431.	1549.
350	812.4	1135.	239.3	566.1	1617.	1940.	2101.
400	1061.	1482.	312.5	739.4	2113.	2534.	2745.

Table C.1: Subdomain memory requirement in kilowords where 1 word corresponds to the storage needed for one floating point array element.

	operator	problem	work	(msos)
5 stripe				
point	51 %	14 %	35%	
line	46 %	13 %	31%	10%
9 stripe				
point	59 %	12 %	29%	
line	54 %	11 %	27%	8%

Table C.2: Breakdown of memory allocation as percentage of total for multigrid.

pct. overlap	N	5 stripe operator	9 stripe operator	problem	work	5-all	9-all
	(4x 30)	29.5	40.3	7.0	19.5	56.1	66.9
10%	57	27.7	37.4	6.3	4.9	39.0	48.6
30%	52	26.1	34.7	5.3	4.9	36.3	44.9
50%	45	24.2	31.4	4.0	4.9	33.0	40.3
	(4x 40)	50.6	69.3	12.5	33.7	96.8	115.5
10%	76	47.8	64.7	11.3	8.4	67.5	84.4
30%	70	45.2	60.4	9.6	8.4	63.2	78.4
50%	60	41.4	54.0	7.0	8.4	56.9	69.5
	(4x 50)	75.1	103.6	19.5	50.8	145.4	173.9
10%	95	71.1	96.8	17.6	12.7	101.4	127.1
30%	87	66.8	89.7	14.8	12.7	94.3	117.2
50%	75	61.1	80.2	11.0	12.7	84.8	103.9
	(4x 60)	105.7	146.0	28.1	71.8	205.6	245.9
10%	114	100.1	136.8	25.4	18.0	143.5	180.1
30%	105	94.3	127.2	21.5	18.0	133.8	166.6
50%	90	85.8	112.9	15.8	18.0	119.5	146.7
	(4x 70)	140.8	195.1	38.3	96.2	275.4	329.6
10%	133	133.6	183.0	34.5	24.1	192.2	241.6
30%	122	125.4	169.3	29.1	24.1	178.5	222.4
50%	105	114.1	150.5	21.5	24.1	159.7	196.1
	(4x 80)	183.4	254.0	50.0	125.3	358.6	429.3
10%	152	174.2	238.6	45.1	31.3	250.6	315.1
30%	140	163.9	221.5	38.3	31.3	233.5	291.1
50%	120	148.7	196.1	28.1	31.3	208.1	255.6
	(4x 90)	227.7	316.1	63.3	156.4	447.4	535.8
10%	171	216.3	297.2	57.1	39.1	312.5	393.4
30%	157	202.8	274.7	48.1	39.1	290.1	362.0
50%	135	184.0	243.4	35.6	39.1	258.7	318.1
	(4x100)	280.2	389.1	78.1	192.6	550.9	659.8
10%	190	266.4	366.1	70.5	48.1	385.1	484.8
30%	175	250.4	339.4	59.8	48.1	358.3	447.4
50%	150	226.6	299.7	43.9	48.1	318.7	391.8

Table C.3: Memory requirement in kilowords where 1 word corresponds to a floating point array element for the 4 color problem. Work space and finest grid difference operator is shared.

Appendix D

A Matrix Convergence Proof

Efforts to accelerate the Schwarz process in the continuous domain show that using some type of derivative boundary condition at the pseudo-boundaries of the subdomains, as opposed to imposing a Dirichlet boundary condition is beneficial [RS89]. In this section we prove the convergence of matrix splittings in the discrete domain corresponding to the addition of some degree of derivative boundary conditions at the pseudo-boundaries in the continuous domain.

We consider on the two–dimensional Helmholtz equation with Dirichlet boundary conditions:

$$\begin{cases} -\Delta u + q^2 u = f & \text{in } \Omega = \{(x,y) \mid 0 < x < 1, \quad 0 < y < 1\} \\ u \mid_\Gamma = \phi \end{cases} \tag{4.1}$$

where $q \neq 0$, and f and ϕ are given. To simplify our proof, we assume q is constant.

Consider the Dirichlet problem (4.1) and choose a square grid with mesh size $h = 1/(n_h + 1)$ on the domain Ω. If we set $u_{i,j} = u(ih, jh)$, $i, j = 1, 2, \ldots, n_h$, and replace the partial derivatives u_{xx} and u_{yy} by central difference approximations, we arrive at the finite difference equations

$$\begin{array}{c} -u_{i,j-1} - u_{i-1,j} + (4 + h^2 q^2) u_{i,j} - u_{i+1,j} - u_{i,j+1} = h^2 f_{i,j}, \qquad 1 < i, j < n_h, \\ u_{i,0} = \phi_{i,0};\ u_{0,j} = \phi_{0,j};\ u_{i,n_h+1} = \phi_{i,n_h+1};\ u_{n_h+1,j} = \phi_{n_h+1,j}. \end{array} \tag{4.2}$$

We now formulate the matrix problem with each row of the matrix A corresponding to a difference equation at one of the grid points (x_i, x_j). Define $n = n_h \times n_h$. We write the matrix form of the $n \times n$ system determined by (4.2) as

$$Au_{i,j} = f_{i,j}. \tag{4.3}$$

The matrix A has the block tridiagonal matrix structure

$$A = \begin{pmatrix} \hat{D} & -I & & & & & \\ -I & \hat{D} & -I & & & & \\ & -I & \hat{D} & -I & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & -I & \hat{D} & -I \\ & & & & & -I & \hat{D} \end{pmatrix}.$$

Define $\Delta = h^2 \bar{q}^2$ and introduce the J matrix, (ones on the subdiagonal, zero elsewhere) and the L matrix below.

$$J = \begin{pmatrix} 0 & & & & & & \\ 1 & 0 & & & & & \\ & 1 & 0 & & & & \\ & & 1 & 0 & & & \\ & & & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \\ & & & & & 1 & 0 \\ & & & & & & 1 \quad 0 \end{pmatrix}_{n_h \times n_h} \qquad L = \begin{pmatrix} J & & & & & & \\ I & J & & & & & \\ & I & J & & & & \\ & & I & J & & & \\ & & & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \\ & & & & & I & J \\ & & & & & & I \quad J \end{pmatrix}_{n \times n}$$

Furthermore, let $\hat{D} = \bar{D} - J - J^T$ (tridiagonal) with $\bar{D} = (4 + \Delta)I$, so the $\hat{D}$ in A above looks like

$$\hat{D} = \begin{pmatrix} 4+\Delta & -1 & & & & & \\ -1 & 4+\Delta & -1 & & & & \\ & -1 & 4+\Delta & -1 & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & -1 & 4+\Delta & -1 \\ & & & & & -1 & 4+\Delta \end{pmatrix}.$$

The standard Gauss-Seidel iterative method can be viewed as having a Dirichlet boundary condition at each pseudo-boundary. Its splitting $A = M_{gs} - N_{gs}$, has $M_{gs} = \bar{D} - L$ and $N_{gs} = L^T$. The matrix M_{gs} of the Gauss-Seidel splitting satisfies the requirements for being an M-matrix [Var62]. [Var62] gives a proof showing the Gauss-Seidel splitting meets the conditions for regular splitting. The general form of the Gauss-Seidel matrix splitting for problem (4.1) is

$$M_{gs} = \begin{pmatrix} \bar{D}-J & & & & & \\ -I & \bar{D}-J & & & & \\ & -I & \bar{D}-J & & & \\ & & \ddots & \ddots & & \\ & & & -I & \bar{D}-J & \\ & & & & -I & \bar{D}-J \end{pmatrix}, \tag{4.4}$$

and

$$N_{gs} = \begin{pmatrix} J^T & I & & & & \\ & J^T & I & & & \\ & & J^T & I & & \\ & & & \ddots & \ddots & \\ & & & & J^T & I \\ & & & & & J^T \end{pmatrix}. \tag{4.5}$$

Suppose, for example, we wish to solve problem (4.1) when $h = 1/4$ implying a 3×3 square of unknowns. Then, the numerical operator for this system is

$$A_{ex} = \left(\begin{array}{ccc|ccc|ccc} 4+\Delta & -1 & & -1 & & & & & \\ -1 & 4+\Delta & -1 & & -1 & & & & \\ & -1 & 4+\Delta & & & -1 & & & \\ \hline -1 & & & 4+\Delta & -1 & & -1 & & \\ & -1 & & -1 & 4+\Delta & -1 & & -1 & \\ & & -1 & & -1 & 4+\Delta & & & -1 \\ \hline & & & -1 & & & 4+\Delta & -1 & \\ & & & & -1 & & -1 & 4+\Delta & -1 \\ & & & & & -1 & & -1 & 4+\Delta \end{array}\right).$$

We illustrate below the corresponding Gauss-Seidel regular splitting.

$$M_{ex} = \left(\begin{array}{ccc|ccc|ccc}
4+\Delta & & & & & & & & \\
-1 & 4+\Delta & & & & & & & \\
 & -1 & 4+\Delta & & & & & & \\
\hline
-1 & & & 4+\Delta & & & & & \\
 & -1 & & -1 & 4+\Delta & & & & \\
 & & -1 & & -1 & 4+\Delta & & & \\
\hline
 & & & -1 & & & 4+\Delta & & \\
 & & & & -1 & & -1 & 4+\Delta & \\
 & & & & & -1 & & -1 & 4+\Delta
\end{array}\right),$$

and

$$N_{ex} = \left(\begin{array}{ccc|ccc|ccc}
0 & 1 & & 1 & & & & & \\
 & 0 & 1 & & 1 & & & & \\
 & & 0 & & & 1 & & & \\
\hline
 & & & 0 & 1 & & 1 & & \\
 & & & & 0 & 1 & & 1 & \\
 & & & & & 0 & & & 1 \\
\hline
 & & & & & & 0 & 1 & \\
 & & & & & & & 0 & 1 \\
 & & & & & & & & 0
\end{array}\right).$$

We want to widen the class of matrix splittings for which we can prove convergence. To set up the matrix splitting for the Gauss-Seidel like method used by Rodrigue and Shah in [RS89], we subtract a portion, Σ, from the M-matrix's diagonal and place it on the diagonal of N. The matrix Σ is an $n \times n$ diagonal matrix with σ_i the degree of derivative conditions selected, $\{0 \le \sigma_i \le 2 : i = 1, \ldots, n\}$. We call the new matrices derived from M and N, P and Q respectively. To fully illustrate we define an example Σ matrix

$$\Sigma_{ex} = \left(\begin{array}{ccc|ccc|ccc}
0 & & & & & & & & \\
 & 1 & & & & & & & \\
 & & 0 & & & & & & \\
\hline
 & & & 1 & & & & & \\
 & & & & 2 & & & & \\
 & & & & & 1 & & & \\
\hline
 & & & & & & 0 & & \\
 & & & & & & & 1 & \\
 & & & & & & & & 0
\end{array}\right),$$

which in turn defines the splitting

$$P_{ex} = \left(\begin{array}{ccc|ccc|ccc}
4+\Delta & & & & & & & & \\
-1 & 3+\Delta & & & & & & & \\
 & -1 & 4+\Delta & & & & & & \\
\hline
-1 & & & 3+\Delta & & & & & \\
 & -1 & & -1 & 2+\Delta & & & & \\
 & & -1 & & -1 & 3+\Delta & & & \\
\hline
 & & & -1 & & & 4+\Delta & & \\
 & & & & -1 & & -1 & 3+\Delta & \\
 & & & & & -1 & & -1 & 4+\Delta
\end{array}\right),$$

$$Q_{ex} = \left(\begin{array}{ccc|ccc|ccc}
0 & 1 & & 1 & & & & & \\
 & -1 & 1 & & 1 & & & & \\
 & & 0 & & & 1 & & & \\
\hline
 & & & -1 & 1 & & 1 & & \\
 & & & & -2 & 1 & & 1 & \\
 & & & & & -1 & & & 1 \\
\hline
 & & & & & & 0 & 1 & \\
 & & & & & & & -1 & 1 \\
 & & & & & & & & 0
\end{array}\right).$$

We take the splitting $A = P - Q$ such that $P = D - (L - \Sigma)$ and $Q = L^T - \Sigma$. Since $\mathrm{diag}(A) = \mathrm{diag}(P) - \mathrm{diag}(Q)$, we have $D = 4 + \Delta - 2\Sigma$. The M and N of (4.4) and (4.5) can be compared and contrasted to P and Q which are of the form

$$P = \begin{pmatrix}
D-J+\Sigma & & & & & \\
-I & D-J+\Sigma & & & & \\
 & -I & D-J+\Sigma & & & \\
 & & \cdot & \cdot\;\cdot & & \\
 & & & \cdot\;\cdot & \cdot & \\
 & & & -I & D-J+\Sigma & \\
 & & & & -I & D-J+\Sigma
\end{pmatrix}, \qquad (4.6)$$

$$Q = \begin{pmatrix}
J^T-\Sigma & I & & \cdot & & \\
 & J^T-\Sigma & I & & & \\
 & & J^T-\Sigma & I & & \\
\cdot & & & \cdot\;\cdot & & \\
\cdot & & & \cdot & \cdot & \\
 & & & & J^T-\Sigma & I \\
 & & & & & J^T-\Sigma
\end{pmatrix}. \qquad (4.7)$$

Our goal is to prove the splitting $A = P - Q$ is convergent. Matrices arising from the discretization of the Helmholtz equation are known to be M-matrices. As a result A^{-1} is non-negative. If $P - Q$ were a regular splitting, then by Theorem 2.4 the iterative method associated with the splitting would be convergent. Unfortunately, by definition diag$(Q) < 0$, Q is not non-negative and we do not have a regular splitting.

If we write P as $D - \omega E$ and Q as ωE^T then we can apply the Ostrowski-Reich theorem provided D and P meet certain conditions. Let S_ω be the amplification matrix $P^{-1}Q$. Matrices (4.6) and (4.7) illustrate the case of $\omega = 1$. When $\omega = 1$ the splitting is similar to an Gauss-Seidel splitting. For $\omega = 1$ the off-diagonal elements of P and Q are the same as the off-diagonal elements of M and N of the Gauss-Seidel splitting. The difference is a portion of M, Σ, has been moved to N. When $\omega \neq 1$ the splitting is similar to an SOR splitting.

Now, the main result of this section. For the Dirichlet problem (4.1), we prove the following theorem:

Theorem D.1 *Let $A = P - Q$ be the described above, where A is the difference operator approximating the differential equation of problem (4.1) and P and Q are as specified in (4.6) and (4.7) respectively. If $\{0 \leq \sigma_i \leq 2 : i = 1, \ldots, n\}$ then $\rho(S_\omega) < 1$. Thus, the matrix S_ω is convergent, and the associated iterative method $Px^{k+1} = Qx^k + f$, $k \geq 0$, converges for any initial vector x^0.*

Proof:

We begin by recalling the Ostrowski-Reich theorem (as it appears in Theorem 3.6 of [Var62]).

Theorem D.2 *(Ostrowski-Reich) Let $A = D - E - E^*$ be an $n \times n$ Hermitian matrix, where D is Hermitian and positive definite, and $D - \omega E$ is nonsingular for $0 \leq \omega \leq 2$. Then $\rho(S_\omega) < 1$ if and only if A is positive definite and $0 < \omega < 2$.*

The matrix A is known to be positive definite. To use the theorem we must cast A in terms of the appropriate D and E. We take $E = L - \Sigma \Rightarrow Q = \omega E^T = \omega(L - \Sigma)^T$. Given A, and having chosen Q, to obtain the desired P, $D - \omega E$, we choose D to be the diagonal matrix $4 + \Delta - 2\Sigma$. Now, having chosen $A = D - E - E^T$, it remains to show

(1) D is Hermitian and positive definite,

(2) $D - \omega E$ is nonsingular for $0 \leq \omega \leq 2$.

Proving (1) is trivial: D is diagonal and has positive real elements – since $4 + \Delta > 2\sigma_i$ for all i. Therefore, D is Hermitian and positive definite.

To show $D - \omega E$ is nonsingular we prove it is strictly diagonally dominant. To prove strict diagonal dominance we show that our lower triangular matrix P satisfies the stricter condition

$$\min_i \operatorname{diag}(P) > \max_i \sum_{j=1}^{i-1} |p_{i,j}|, \qquad 1 \leq i \leq n. \tag{4.8}$$

First we consider the right side of inequality (4.8). For our matrix $P = D - \omega E$

$$\max_i \sum_{j=1}^{i-1} |p_{i,j}| = 2\omega, \qquad 1 \leq i \leq n \tag{4.9}$$

since the E matrix has at most two off-diagonal elements of unity in any one row and the diagonal matrix D makes no contribution to P's off-diagonal elements. Now considering the left hand side of inequality (4.8) we have

$$\operatorname{diag}(P) = 4 + \Delta - (2 - \omega)\Sigma. \tag{4.10}$$

Since the multiplier of Σ in (4.10) is ≥ 0 the minimum is attained for the element i at which σ_i is a maximum. Since, $\max_i \sigma_i \leq 2$, this yields

$$\min [\ \operatorname{diag}(P)\ ] = 4 + \Delta - 4 + 2\omega = \Delta + 2\omega \tag{4.11}$$

$$\min[\operatorname{diag}(P)] = 4 + \Delta - (2 - \omega)\max[\sigma_i] \geq \Delta + 2\omega$$

Substituting (4.9) and (4.11) into (4.8) gives

$$\Delta + 2\omega > 2\omega. \tag{4.12}$$

Obviously (4.12) holds true for any ω so the inequality of (4.8) holds true for $0 \leq \omega \leq 2$. Thus $D - \omega E$ and therefore P is strictly diagonally dominant and as a consequence nonsingular (Theorem 2.2). Finally, we may apply the Ostrowski-Riech theorem. The positive definiteness of A and limitation of ω to $0 < \omega < 2$ guarantees $\rho(S_\omega) < 1$.

Bibliography

[ABDP81] R. E. Alcouffe, A. Brandt, J. E. Dendy, and J.W. Painter. The Multigrid Method for the Diffusion Equation with Strongly Discontinuous Coefficients. *SIAM J. SISSC*, 2(4):430–454, 1981.

[ASM86] Walid Abu-Sufah and Allen D. Maloney. Experimental Results for Vector Processing on the Alliant FX/8. Technical Report CSRD Rpt. No. 539, Center for Supercomputing Research and Development, Univeristy of Illinois, Urbana, Illinois, 1986.

[Ast71] G. P. Astrakhantsev. An iterative method of solving elliptic net problems. *U.S.S.R Computational Math. and Math. Phys.*, 11(2):171–182, 1971.

[Bak66] N. S. Bakhvalov. On the convergence of a relaxation method with natural constraints on the elliptic operator. *U.S.S.R Computational Math. and Math. Phys.*, 6(5):101–135, 1966.

[BD80] R. E. Bank and T. F. Dupont. Analysis of a two-level scheme for solving finite element equations. Technical Report CNA-159, University of Texas at Austin, 1980.

[BD81] R. E. Bank and T. F. Dupont. An optimal order process for solving elliptic finite element equations. *Math. Comp.*, 36:35–51, 1981.

[BM84] W. Briggs and S. McCormick. Introduction. In Stephen F. McCormick, editor, *Multigrid Methods*, chapter 1, pages 1–30. Society for Industrial and Applied Mathematics, Philedelphia, Penn., 1984.

[Bra72] A. Brandt. Multi-level adaptive technique (MLAT) for fast numerical solutions to boundary-value problems. In *Cabannes - Temam*, chapter 1, pages 82–89. 1972.

[Bra76] A. Brandt. Multi-level adaptive techniques. Technical Report RC 6026, IBM TJ Research Center, Yorktown Heights, 1976.

[Bra77] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Math. Comp.*, 31:333–390, 1977.

[CH62] R. Courant and D. Hilbert. *Methods of Mathematical Physics*, volume 2. Willey, New York, 1962.

[CT87] Tony F. Chan and Ray S. Tuminaro. A Survey of Parallel Multigrid Algorithms. Technical Report RIACS 87.22, Research Institute for Advanced Computer Science, NASA Ames Research Center, Corso Tenuto a lspra, August 1987.

[Dei84] Harvey M. Deitel. *An Introduction to Operating Systems.* Addison-Wesley Publishing Company, Reading, Mass, 1984.

[Den82] J. E. Dendy. Black Box Multigrid. *Journal of Computational Physics*, 48:366–386, 1982.

[Den88] Joel E. Dendy. 1988. Personal communication.

[DGP80] R. V. Dinh, R. Glowinski, and J. Periaux. Applications of domain decomposition methods for Navier-Stokes equations. Technical report, INF-LAB, France, 1980.

[D'J62] E. G. D'Jakonov. A method for solving Poisson's equation. *Soviet Math*, 3:320–323, 1962.

[Dup67] T. Dupont. On the Existence of an Iterative Method for the Solution of Elliptic Difference Equation with an Improved work estimate. Technical report, Centro Tenuto Internationale Mathmatico Estivo, Corso Tenuto a lspra, 1967.

[Fed62] R. P. Fedorenko. A relaxation method for solving ellipitc difference equations. *U.S.S.R Computational Math. and Math. Phys.*, 1(5):1092–1096, 1962.

[Fed64] R. P. Fedorenko. The speed of convergence of an iterative process. *U.S.S.R Computational Math. and Math. Phys.*, 3(4):227–235, 1964.

[FM66] G. Fairweather and A. Mitchell. Some computational results of an improved ADI method for the Dirchlet problem. *Computer J.*, 9:298–307, 1966.

[FXm86] *FX/Series Architecture Manual.* Alliant Computer Systems Corporation, Acton, Mass., 1986. Part number 300-00001-B.

[GDP80] R. Glowinski, Q. V. Dinh, and J. Periaux. Domain decomposition methods for nonlinear problems in fluid dynamics. Technical report, INRIA, France, 1980.

[Gre84] A. Greenbaum. Analysis of a multigrid method as an iterative technique for solving linear systems. *SIAM Journal on Scientific and Statistical Computing*, 21(3):473–485, June 1984.

[Hac80a] W. Hackbusch. Convergence of multi-grid iterations applied to difference equations. *Math. Comp.*, 34:425–440, 1980.

[Hac80b] Wolfgang Hackbusch. Survey of Convergence Proofs for Multi-grid Iterations. In J. Frehse, D. Pallaschke, and U. Trottenberg, editors, *Special Topis of Applied Mathematics.* North-Holland, Amsterdam-New York-Oxford, 1980.

[Hac85] W. Hackbusch. *Multi-Grid Methods and Appplications.* Springer-Verlag, Berlin, 1985.

[Kan81] Lishan Kang. The Schwarz Algorithm. *Wuhan University Journal*, pages 88–88, 1981. Natural Science Edition, Special Issue of Mathematics, China.

[KCSQ85] L. Kang, Y. Chen, L. Sun, and H. Quan. A Class of New Asynchronous Parallel Algorithms for Solving Partial Differential Equations. Technical Report 19, Wuhan University, Wuhan, China, 1985.

[Ket82] Rob Kettler. Analysis and comparison of relaxation schemes in robust multigrid and preconditioned conjugate gradient methods. Technical Report 82-17, Delft Univerisity of Technology, 1982.

[KK58] L. V. Kantorovich and V. I. Krylov. *Appoximate Methods of Higher Analysis.* P. Noordhoff, Ltd., Groningen, Netherlands, 1958.

[KW59] L. S. Kang and D. R. Wang. *Lectures on the Finite Difference Method.* Wuhan Unversity Press., China, 1959.

[Lin81] J. Linden. Mehrgitterverfahren für die Poisson-Gleichung in Kreis und Ringgebiet unter Verwendung lokaler Koordinaten. Master's thesis, Institut f"ur Angewandte Mathematik, Universität Bonn, January 1981. Diplomarbeit.

[Lio78] P. L. Lions. Interpretation stochastique de la mêthode alterneè de schwarz. Technical Report 286, C. R. Acad. Sc, Paris, 1978.

[Mei86] U. Meier. Two parallel SOR variants of the Schwarz Alternating Procedure. Technical Report 5710, Zentralinstitut für Angewandte Mathematik, Julich, West Germany, 1986.

[Mik34] S. G. Mikhlin. The method of succesive approximations applied to the biharmonic problem. *Trudy Seism*, 34:1–14, 1934.

[Mik51] S. G. Mikhlin. On the Schwarz Algorithm. *Doklady Akademia Nauk U.S.S.R.*, LXXVII(4):569–571, 1951.

[Mik65] S. G. Mikhlin. The Problem of the Minimum of a Quadratic Functional. 1965.

[Mil65] K. Miller. Numerical analogs to the Schwarz Alternating Procedure. *Numerische Mathematik*, 7:91–103, 1965.

[Mol81a] W. J. A. Mol. On the choice of suitable operators and parameters in multigrid methods. Technical Report NW 107/81, Mathematisch Centrum, Amsterdam, 1981.

[Mol81b] W. J. A. Mol. Smoothing and coarse grid approximation properties of multigrid methods. Technical Report NW 110/81, Mathematisch Centrum, Amsterdam, 1981.

[Mys59] I. P. Mysovskih. The finite difference method for solving the Dirichlet problem in the rectangular region. *XUEBAO of Kirin University*, 1:39–45, 1959.

[Neu90] C. Neumann. Zur theorie des logrithmischen und newton'schen potentials. *Leipziger Berichte*, 22:264–321, 1890.

[Nic75] R. A. Nicolaides. On multiple grid and and related techniques for solving discrete elliptic systems. *J. Comput. Phys.*, 19:418–431, 1975.

[Nic77] R. A. Nicolaides. On the ℓ^2 convergence of an algorithm for solving finite element equation. *Math. Comp.*, 31:892–906, 1977.

[OST86] Joseph Oliger, William Skamarock, and Wei-Pai Tang. Schwarz Alternating Method and its SOR accelerations. Technical report, Stanford University, Computer Science Department, 1986.

[Pic90] E. Picard. Memoire sur la theorie des equations aux derivees partiells et la methode des approximations successives. *J. Math. Pures et Appl.*, 4(6):145–210, 1890.

[Poi90] H. Poincaré. Sur les equations aux derivees partielles de la physique mathematique. *Amer. J. Math.*, 12:211–294, 1890.

[PR89] A. Louise Perkins and G. Rodrigue. A Domain Decomposition Method for Solving A 2-Dimensional Viscous Burgers' Equation. *Journal of Applied Numerical Methods*, 1989.

[Rod86] Garry Rodrigue. Inner/outer Iterative Methods and Numerical Schwarz Algorithms. *Parallel Computing*, 2:205–218, 1986.

[RS84a] G. Rodrigue and J. Simon. A generalization of the numerical Schwarz algorithm. In R. Glowinski and J. Lions, editors, *Computing Methods in Applied Sciences and Engineering VI*, pages 273–283. North-Holland, Amsterdam-New York-Oxford, 1984.

[RS84b] G. Rodrigue and J. Simon. Jacobi Splittings and the Method of Overlapping Domains for Solving Elliptic PDEs. In R. Vichnevetsky and R. Stepleman, editors, *Advances in Computer Methods for Partial Differential Equations V*, pages 383–386. IMACS, June 1984.

[RS85] G. Podigue and P. Saylor. Inner/outer Iterative Methods and Numerical Schwarz Algorithms -II. In *Proceedings of the IBM Conference on Vector and Parallel Computations for Scientific Computing.* IBM, 1985.

[RS89] Garry Rodrigue and Shantilal Shah. Pseudo-Boundary conditions to accelerate parallel Schwarz methods. UCRL 100893, Lawrence Livermore National Laboratory, April 1989.

[Sch69] H. A. Schwarz. Ueber einige Abbildungsaufgaben. *Jour. f. dei reine und angew. Math.*, 70:105–120, 1869.

[Sim84] Jeff Herbert Simon. Domain Decomposition for Solving Elliptic Partial Differential Equations on Multiprocessors. Master's thesis, Univeristy of Califonia Davis, May 1984.

[Sob36] S. L. Sobolev. l'Algorithme de Schwarz dans la théoire de l'élasticité. *Comptes Rendus (Doklady) de l'academie des Sciences de l'URSS*, IV (VIII)(6 (110)):235–238, 1936.

[ST82a] Klaus Stüben and Ulrich Trottenberg. Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications. In A. Dold and B. Eckmann, editors, *Multigrid Methods*, pages 1–150, 1982. Proceedings of the Conference Held at Köln-Porz, November 23-27, 1981.

[ST82b] Klaus Stüben and Ulrich Trottenberg. Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications. In *Multigrid Methods, Lecture Notes im Mathematics*, volume 960, Berlin, 1982. Springer-Verlag.

[Sta77] G. Starius. Composite mesh difference methods for elliptic boundary value problems. *Numerische Mathematik*, 28:243–258, 1977.

[Sto72] D. Stoutemyer. *Numerical Implementation of the Schwarz Alternating Procedure for Elliptic Partial Differential Equations.* PhD thesis, Stanford University, Stanford, CA 94305, 1972.

[Sto73] D. Stoutemyer. Numerical Implementation of the Schwarz Alternating Procedure for Elliptic Partial Differential Equations. *SIAM J. Numer. Anal.*, 10(2):308–326, 1973.

[Tan87] Wei Pai Tang. *Schwarz Splitting and Template Operators.* PhD thesis, Stanford University, 1987.

[Var62] R. S. Varga. *Matrix Iterative Analysis.* Prentice Hall, Inc., Englewood Cliffs, NJ, 1962.

[Vol68] E. Volkov. The Method of Composite Meshes for Finite and Infinite Region with Piecewise Smooth boundary. Technical Report 96, Steklov Institue of Mathematics, 1968.

[Wac66] E. L. Wachspress. *Iterative Solution of Elliptic Systems and Applications to the Nuetron Diffusion Equations of Reactor Physics.* Prentice-Hall, Englewood Cliffs, N. J., 1966.

[Wer60] H. Werner. Schwarz's alternating method for boundary value problems of the third kind. Technical report, University of Southern California, 1960.

[Wer63] H. Werner. Anwendungen und Fehlerabschaetzungen fuer das alternierende Verfahren von H.A. Schwarz. *ZAMM*, 43:55–61, 1963.

[Wes80] P. Wesseling. The rate of convergence in the multiple grid method. In G. A. Watson, editor, *Lecture Notes in Math 773*, pages 164–184. Springer, Berlin, 1980. Numerical analysis. Proceedings, Dundee, June 1979.

[Wes84] P. Wesseling. Linear Multigrid Methods. In Stephen F. McCormick, editor, *Multigrid Methods*, chapter 2, pages 57–72. Society for Industrial and Applied Mathematics, Philedelphia, Penn., 1984.

END

DATE
FILMED

10/2/91