UCRL-53918
Distribution Category UC-405

UCRL--53918

DE90 007929

A Parallel Multigrid Method for Solving Elliptic Partial Differential Equations

Theodore E. Ferretta

(M.S. Thesis)

Manuscript date: February 1989

LAWRENCE LIVERMORE NATIONAL LABORATORY
University of California • Livermore, California • 94550

Available from: National Technical Information Service • U.S. Department of Commerce
5285 Port Royal Road • Springfield, VA 22161 • A03 • (Microfiche A01)

A Parallel Multigrid Method For Solving Elliptic Partial Differential Equations

By

THEODORE E. FERRETTA
B.S. (California State University, Hayward) 1981
M.S. (California State University, Hayward) 1984

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Committee in Charge

1989

Abstract

This paper introduces a parallel multigrid method for solving elliptic partial differential equations. This method combines two other methods, both of which are popular methods under research today. One of the methods is a multigrid method, essentially a sequential method. The other is a parallel domain decomposition method, a variation on the Schwarz alternating procedure.

Each method is explained individually, before the combined method is explained. The combined method is then compared to each of the individual methods, demonstrating the superiority of the combined method over each of its parent methods.

As a model problem Poisson's equation is used.

The computer on which the various methods were tested is an Alliant FX/8, a shared memory multiprocessor machine having 8 processors which can be run simultaneously in executing parallel code.

Acknowledgements

I am sincerely grateful to my thesis advisor, Garry Rodrigue, for his guidance and encouragement throughout this investigation. In addition, a special thanks goes to both Pat Fitch and Peter Linz for serving on my thesis committee. Finally, the members of the Applied Mathematics Group at Lawrence Livermore National Laboratory deserve a thank you for providing me the time and encouragement to complete this endeavor.

Contents

Chapter 1

Introduction

Boundary value problems for elliptic partial differential equations are of major importance in computational physics and engineering. They occur, among other places, in the areas of fluid dynamics, electrodynamics, stationary heat and mass transport (diffusion), statics, and reactor physics (neutron transport) [1].

Most numerical methods for solving such problems involve a discretization of the problem using a finite-dimensional approximation space, followed by a numerical procedure to solve (or approximate) the resulting very large system of linear equations.

Many of the methods that have been used to solve such large systems have been in existence even longer than computer technology (eg. Jacobi, Gauss-Seidel, and successive over relaxation methods [2, 3]).

Since the inception of the computer, there have been enormous gains in the computational speed of large system solvers. These increases are due mainly to technological advances in electronic circuitry and component fabrication. Hardware performance increases, however, are physically limited by the speed of light, a factor which has caused a leveling off of gains in performance in recent years. Furthermore, despite the development of faster circuits, even on today's fastest computers, solving very large systems can require hundreds of hours of CPU time.

Two directions are currently being explored for potential solutions to this problem. The first involves the development of significantly new sequential algorithms running on traditional von Neumann single processor architectures. The second involves the development of parallel numerical algorithms to be run on multiprocessor architectures.

I have investigated both of these directions as part of the background research for my thesis. I have designed and implemented an algorithm that contains elements from both of these areas. I have restricted my attention to one principle algorithm from each area. The sequential algorithm I have been working with is a multigrid method. The parallel algorithm is a parallel domain decomposition method, a variation on the Schwarz alternating procedure.

Both of these algorithms, as well as the combined algorithm, requires the use of one of the more basic iterative methods for solving a matrix equation. The basic

method I have used in each of the three algorithms is the Gauss-Seidel method. I also use the Gauss-Seidel method in comparing the relative efficiency of the three algorithms.

All algorithms implemented for this thesis were developed and run on an Alliant FX/8 computer.

1.1 The Alliant FX/8

The Alliant FX/8 computer is a shared memory multiprocessor. The architecture of this system is represented in figure 1.1 [4]. The computational power of this system resides in the 8 computational elements (CEs). Each CE is a CMOS gate array implementation of a full scalar architecture, with additional hardware for IEEE floating point operations, vector operations, concurrency control, and virtual memory support. In vector mode, each CE executes floating point instructions at the peak rate of 11.8 million floating point operations per second (MFLOPS) for both single and double precision. All 8 CEs can be used concurrently to solve a single problem. Such a complex of 8 CEs can approach a performance level equal to 8 times the performance of a single CE for certain problems. Dedicated hardware on each CE is used to control the scheduling and synchronization of multiple CEs when used in a concurrent mode. There is also an expandable pool of interactive processors (IPs) that execute interactive user jobs and the operating

system. The IPs maintain system responsiveness and allow the CEs to concentrate on the computational-intensive portions of user applications.

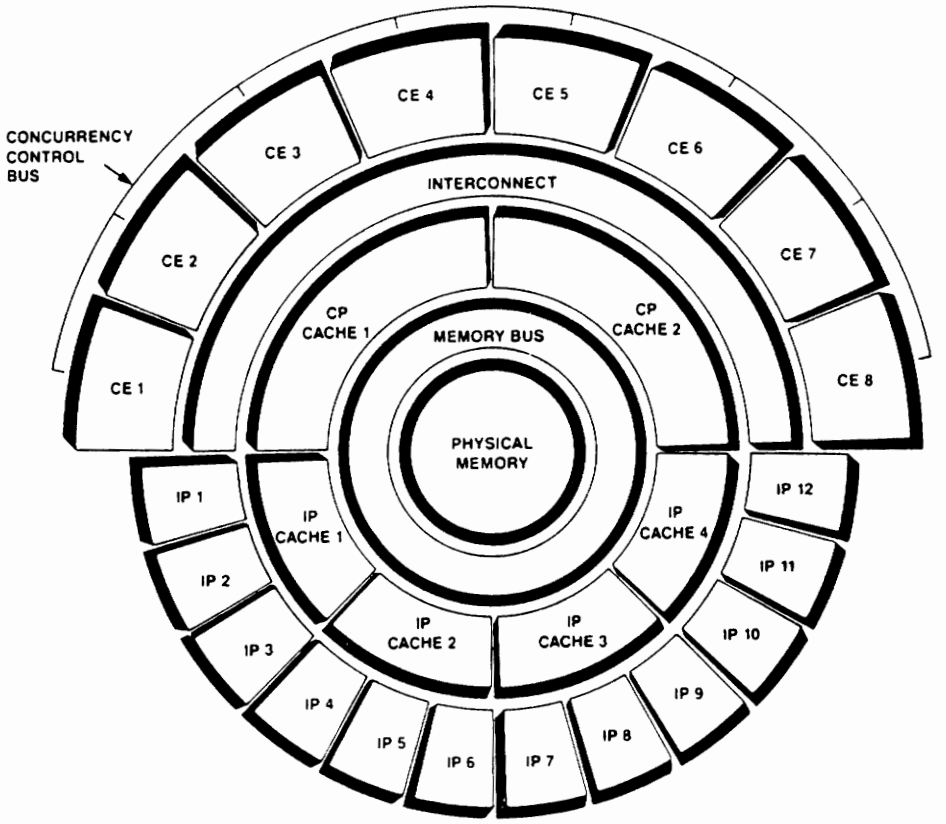

Figure 1.1: The Alliant FX/8

Chapter 2

Elliptic Partial Differential Equations

2.1 Classification of Partial Differential Equations

A *partial differential equation (pde)* is an equation

$$F(x, y, ..., u, u_x, u_y, ..., u_{xx}, u_{xy}, ...) = 0 \tag{2.1}$$

involving more than one independent variable x, y,..., a function u of these variables, and the partial derivatives u_x, u_y, ..., u_{xx}, u_{xy}, ... of the function. The equation is is called *linear* if the unknown function and its partial derivatives appear, at most, to the first degree. The *order* of the equation is the order of the partial derivative of highest order appearing in the equation.

For example, the most general second-order linear partial differential equation defined on a two-dimensional region Ω, with boundary Γ is

$$au_{xx} + 2bu_{xy} + cu_{yy} + hu_x + ku_y + eu + f = 0 \tag{2.2}$$

where a, b, c, h, k, e, f are functions of x and y. This equation is called

- *elliptic* if $ac - b^2 > 0$,
- *hyperbolic* if $ac - b^2 < 0$, and
- *parabolic* if $ac - b^2 = 0$.

If a, b, c, h, k, and e are all constants we have a second order linear partial differential equation with constant coefficients. Such an equation, by a linear change in coordinates, can always be reduced into one of three normal forms which correspond to the three types of equations referred to above:

- *elliptic* if the form is $u_{xx} + u_{yy} + \gamma u = -f$,
- *hyperbolic* if the form is $u_{xx} - u_{yy} + \gamma u = -f$, and
- *parabolic* if the form is $u_{xx} + u_y = -f$,

where γ is a constant with one of the values -1, 0, or 1. In the elliptic form, with $\gamma = 0$, we have what is known as Poisson's equation,

$$u_{xx} + u_{yy} = -f, \tag{2.3}$$

which is also written as

$$\Delta u = -f. \tag{2.4}$$

If we also have $f = 0$, we have Laplace's equation,

$$u_{xx} + u_{yy} = 0, \tag{2.5}$$

which is also written as

$$\Delta u = 0. \tag{2.6}$$

As model problems we shall use both Laplace's equation and Poisson's equation with different functions f.

2.2 Boundary Value Problems

Any equation, other than the differential equation itself, which a solution of the differential equation is required to satisfy is called an *auxiliary condition* for the equation. Auxiliary conditions may involve undetermined constants or functions. A set of auxiliary conditions is called *appropriate* for the differential equation if there exists one and only one function which satisfies both the differential equation and the auxiliary conditions.

Because of the physical interpretation associated with an elliptic partial differential equation the most natural auxiliary conditions associated with such an equation

are *boundary conditions*. There are three basic types of boundary value conditions: the *Dirichlet*, *Neumann*, and *Robins* conditions.

Using Laplace's equation as an example, we give three representative boundary value problems. First of all, the Dirichlet problem is given by

$$\begin{aligned} \Delta u &= 0 \quad \text{on} \quad \Omega \\ u &= \phi \quad \text{on} \quad \Gamma = \text{boundary}(\Omega) \end{aligned} \tag{2.7}$$

where ϕ is a known function defined on the boundary Γ of Ω. Next, we have the Neumann problem,

$$\begin{aligned} \Delta u &= 0 \quad \text{on} \quad \Omega \\ \frac{\partial u}{\partial v} &= \phi \quad \text{on} \quad \Gamma = \text{boundary}(\Omega) \end{aligned} \tag{2.8}$$

where $\frac{\partial u}{\partial v}$ denotes the outward normal derivative on the boundary. Lastly, we have the Robins problem in which the auxiliary condition is a linear combination of the previous two,

$$\begin{aligned} \Delta u &= 0 \quad \text{on} \quad \Omega \\ \frac{\partial u}{\partial v} + hu &= \phi \quad \text{on} \quad \Gamma = \text{boundary}(\Omega) \end{aligned} \tag{2.9}$$

where h is a constant greater than zero.

We will be working exclusively with Dirichlet conditions on our model problems.

2.3 A Physical Interpretation

As an illustration of the physical interpretation associated with an elliptic boundary value problem consider the Dirichlet problem, equation (2.7). We may interpret the solution u as the equilibrium temperature distribution in a uniform heat-conducting body occupying the domain Ω, when the temperature distribution on the boundary Γ of the body is kept fixed. On the basis of this physical model, it is reasonable to assume that the problem has a solution if Ω is a finite *reasonably shaped* domain and if the function ϕ is *reasonably smooth*. That the solution is unique follows, for if u_1 and u_2 were both solutions of (2.7), then $v = u_1 - u_2$ would be a solution to

$$\begin{aligned} \Delta v = 0 \quad &\text{on} \quad \Omega \\ v = 0 \quad &\text{on} \quad \Gamma = \text{boundary}(\Omega). \end{aligned} \tag{2.10}$$

So v could be interpreted as the equilibrium temperature in Ω when the boundary Γ is kept at a temperature of zero. From the physical model it is easily seen that v must be identically zero in Ω. Therefore $u_1 = u_2$ and the solution is unique. Hence, the auxiliary conditions for this partial differential equation are appropriate.

As a second illustration consider Poisson's equation with Dirichlet conditions:

$$\begin{aligned} \Delta u = -f \quad &\text{on} \quad \Omega \\ u = \phi \quad &\text{on} \quad \Gamma = \text{boundary}(\Omega). \end{aligned} \tag{2.11}$$

In this case, we again interpert u as the equilibrium temperature distribution in Ω

when there is a known distribution of sources in Ω.

Chapter 3

Discretization of an Elliptic PDE

3.1 Obtaining a Difference Equation

As an illustrative example we will now consider the numerical solution for Poisson's equation with Dirichlet boundary conditions on the unit square. That is

$$-\Delta u = f \quad \text{on} \quad \Omega = (0,1) \times (0,1) \tag{3.1}$$

$$u = \phi \quad \text{on} \quad \Gamma = \text{boundary}(\Omega).$$

We first superimpose a grid G of horizontal and vertical lines over Ω with uniform spacing $h = \frac{1}{n}$, where n is an integer. That is,

$$G = \{(x_i, y_j) | x_i = ih, y_j = jh, h = \frac{1}{n}, 0 < i, j < n, \ i \text{ and } j \text{ integers}\}. \tag{3.2}$$

We wish to determine approximate values of $u(x_i, y_j)$ at each of the grid points (x_i, y_j) of G. For notational convenience we will use $u_{i,j} \equiv u(x_i, y_j)$. One standard

approach to accomplish this end begins with the use of Taylor expansions in two variables. Expanding in the x direction we obtain

$$u_{i+1,j} = u_{i,j} + h\frac{\partial u_{i,j}}{\partial x} + \frac{h^2}{2}\frac{\partial^2 u_{i,j}}{\partial x^2} + \frac{h^3}{6}\frac{\partial^3 u_{i,j}}{\partial x^3} + \frac{h^4}{24}\frac{\partial^4 u_{i,j}}{\partial x^4} + \cdots \tag{3.3}$$

and

$$u_{i-1,j} = u_{i,j} - h\frac{\partial u_{i,j}}{\partial x} + \frac{h^2}{2}\frac{\partial^2 u_{i,j}}{\partial x^2} - \frac{h^3}{6}\frac{\partial^3 u_{i,j}}{\partial x^3} + \frac{h^4}{24}\frac{\partial^4 u_{i,j}}{\partial x^4} - \cdots. \tag{3.4}$$

Summing (3.3) and (3.4) and rearranging we obtain

$$-\frac{\partial^2 u_{i,j}}{\partial x^2} = \frac{1}{h^2}\left[2u_{i,j} - u_{i+1,j} - u_{i-1,j} + \frac{h^4}{24}\frac{\partial^4 u_{i,j}}{\partial x^4} + \cdots\right]. \tag{3.5}$$

Then, expanding in the y direction we obtain

$$u_{i,j+1} = u_{i,j} + h\frac{\partial u_{i,j}}{\partial y} + \frac{h^2}{2}\frac{\partial^2 u_{i,j}}{\partial y^2} + \frac{h^3}{6}\frac{\partial^3 u_{i,j}}{\partial y^3} + \frac{h^4}{24}\frac{\partial^4 u_{i,j}}{\partial y^4} + \cdots \tag{3.6}$$

and

$$u_{i,j-1} = u_{i,j} - h\frac{\partial u_{i,j}}{\partial y} + \frac{h^2}{2}\frac{\partial^2 u_{i,j}}{\partial y^2} - \frac{h^3}{6}\frac{\partial^3 u_{i,j}}{\partial y^3} + \frac{h^4}{24}\frac{\partial^4 u_{i,j}}{\partial y^4} - \cdots. \tag{3.7}$$

Summing (3.6) and (3.7) and rearranging we obtain

$$-\frac{\partial^2 u_{i,j}}{\partial y^2} = \frac{1}{h^2}\left[2u_{i,j} - u_{i,j+1} - u_{i,j-1} + \frac{h^4}{24}\frac{\partial^4 u_{i,j}}{\partial y^4} + \cdots\right]. \tag{3.8}$$

Finally, summing (3.5) and (3.8), we obtain

$$\begin{aligned} -\Delta u|_{(x_i,y_j)} &= -\left[\frac{\partial^2 u_{i,j}}{\partial y^2} + \frac{\partial^2 u_{i,j}}{\partial x^2}\right] \\ &= \frac{1}{h^2}\left[4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1}\right] \\ &\quad + \frac{h^2}{24}\left[\frac{\partial^4 u_{i,j}}{\partial x^4} + \frac{\partial^4 u_{i,j}}{\partial y^4}\right] + \cdots. \end{aligned} \tag{3.9}$$

Dropping terms involving h^2 and higher orders of h and substituting the right hand side of (3.9) into (3.1) we obtain a discrete analog of Poisson's equation

$$\begin{aligned} 4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} &= h^2 f_{i,j} \\ \forall\, (x_i, y_j) &\in G, \end{aligned} \tag{3.10}$$

with the associated boundary conditions

$$u_{i,0} = \phi_{i,0} \quad \text{for} \quad 0 < i < n,$$

$$u_{i,n} = \phi_{i,n} \quad \text{for} \quad 0 < i < n,$$

$$u_{0,j} = \phi_{0,j} \quad \text{for} \quad 0 < j < n, \text{ and}$$

$$u_{n,j} = \phi_{n,j} \quad \text{for} \quad 0 < j < n.$$

Such a discrete analog of a differential equation is called a *finite difference equation.*

3.2 Matrix Form of a Difference Equation

From (3.10) and (3.11), we obtain a system of $(n-1)^2$ linear equations, one equation for each grid point in G. Expressed in matrix form we have

$$A\mathbf{u} = h^2\mathbf{f} + \mathbf{b} \tag{3.11}$$

where

$$A = \begin{bmatrix} B & -I & & & \\ -I & B & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & B & -I \\ & & & -I & B \end{bmatrix} \tag{3.12}$$

is an $(n-1)^2 \times (n-1)^2$ matrix,

$$B = \begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{bmatrix} \quad \text{and } I = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}$$

are $(n-1) \times (n-1)$ matrices, where I is an identity matrix.

$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \vdots \\ \mathbf{u}_{n-1} \end{bmatrix} \text{ and } \mathbf{f} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{f}_3 \\ \vdots \\ \mathbf{f}_{n-1} \end{bmatrix}$$

are $(n-1) \times (n-1)$ component vectors, where

$$\text{each } \mathbf{u}_i = \begin{bmatrix} u_{i,1} \\ u_{i,2} \\ u_{i,3} \\ \vdots \\ u_{i,n-1} \end{bmatrix} \text{ and each } \mathbf{f}_i = \begin{bmatrix} f_{i,1} \\ f_{i,2} \\ f_{i,3} \\ \vdots \\ f_{i,n-1} \end{bmatrix}$$

are $(n-1)$ component vectors. Each $\mathbf{u}_i$ and $\mathbf{f}_i$ correspond to the values of $\mathbf{u}$ and $\mathbf{f}$ associated with the i^{th} column of the grid G.

$$\mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \\ \vdots \\ \mathbf{b}_{n-1} \end{bmatrix}$$

is an $(n-1) \times (n-1)$ component vector which represents the boundary conditions

of the problem, with each $\mathbf{b}_i$ an $(n-1)$ component vector, where

$$\mathbf{b}_1 = \begin{bmatrix} \phi_{0,1} + \phi_{1,0} \\ \phi_{0,2} \\ \phi_{0,3} \\ \vdots \\ \phi_{0,n-2} \\ \phi_{0,n-1} + \phi_{1,n} \end{bmatrix}, \quad \mathbf{b}_{n-1} = \begin{bmatrix} \phi_{n-1,0} + \phi_{n,1} \\ \phi_{n,2} \\ \phi_{n,3} \\ \vdots \\ \phi_{n,n-2} \\ \phi_{n,n-1} + \phi_{n-1,n} \end{bmatrix},$$

and, for $i = 2, ..., n-2$,

$$\mathbf{b}_i = \begin{bmatrix} \phi_{i,0} \\ 0 \\ 0 \\ \vdots \\ 0 \\ \phi_{i,n} \end{bmatrix}.$$

Chapter 4

Methods For Solving Difference Equations

4.1 Direct vs. Iterative Methods

In solving a matrix equation

$$A\mathbf{x} = \mathbf{c}, \tag{4.1}$$

where A is an $n \times n$ nonsingular matrix and $\mathbf{c} \in \Re^n$, there are two general classifications of methods.

The first type of method, a *direct* method, is most often used when the matrix A is *dense*, that is, when the components of A are mostly nonzero.

A direct method yields a solution in a fixed number of steps, assuming computations without roundoff errors. The basic method of this type is *Gaussian Elimination.* Roughly speaking, this is the method of solving a system of linear equations by successively eliminating unknowns by judiciously adding multiples of one row to another. It is such an approach a student is usually introduced to when they see matrices for the first time. More specifically, by a sequence of adding multiples of one equation to another, system (4.1) is reduced to an equivalent *upper triangular* system

$$U\mathbf{x} = \mathbf{g}, \tag{4.2}$$

where U is an upper triangular matrix. System (4.2) can then be easily solved by a simple process called *back substitution.* For further elaboration on this method see Stoer and Bulirsch's book [5].

When using a direct method the matrix A must generally be stored in the main memory of the computer in order to efficiently solve the system. Hence, memory storage limitation is a significant factor to consider when attempting to solve a large problem using such a method.

Among the problems usually considered too large for a direct method are the matrix problems that result from the discretization of partial differential equations. For example, consider equation (3.10) and its matrix form (3.11). With a uniform grid spacing of $h = \frac{1}{64}$, the number of elements of the grid G is $63^2 = 3969$. That is

matrix equation (3.11) is a system of 3969 equations. Thus the number coefficients in the associated matrix A is $3969^2 = 15,752,961$.

The second type of method, an *iterative* method, is most useful when the matrix A is ***sparse***, that is, when the components of A are mostly zero, and when the nonzero coefficients form a special pattern making it possible, by using a simple formula, to generate the coefficients of A as they are needed. Fortunately, the numerical solution of a partial differential equation meets both of these conditions. Consider the previous example of the system where the matrix A contained $15,752,961$ coefficients. Only 19595 of these coefficients are nonzero, the other $15,733,366$ are zero. That is, approximately 99.9 percent of the coefficients are zero. Also the matrix A is completely characterized by equation (3.10), a relatively simple equation, of the previous chapter. The fact that A meets these two conditions eliminates what would otherwise mean a prohibitive use of storage space in main memory.

An iterative method to solve equation (4.1) begins with an initial approximation $\mathbf{x}^{(0)}$ to the solution $\mathbf{x}$, and generates a sequence of vectors $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$ which converge to $\mathbf{x}$. Most iterative methods involves converting the system $A\mathbf{x} = \mathbf{c}$ into an equivalent system of the form

$$\mathbf{x} = T\mathbf{x} + \mathbf{d}. \tag{4.3}$$

After selection the initial approximation $\mathbf{x}^{(0)}$, the sequence of approximate solutions

is generated by

$$\mathbf{x}^{(k+1)} = T\mathbf{x}^{(k)} + \mathbf{d}. \tag{4.4}$$

In the next section we will introduce two classical iterative methods, the *Jacobi* method and the *Gauss-Seidel* method. It is the Gauss-Seidel method that we will use as a basic building block for the more sophisticated iterative methods that we be discussing in later chapters.

4.2 The Jacobi and Gauss-Seidel Methods

The *Jacobi* iterative method consists of solving the i^{th} equation in $A\mathbf{x} = \mathbf{c}$ for x_i, to obtain

$$x_i = \frac{1}{a_{ii}} \left(-\sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n} a_{ij}x_j + c_i \right) \tag{4.5}$$

for each $i = 1, ..., n$,

and then generating each $x_i^{(k+1)}$ from $x_i^{(k)}$ for $k \geq 0$ by

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(-\sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} + c_i \right) \tag{4.6}$$

for each $i = 1, ..., n$.

The Jacobi method can be expressed in the matrix form, $\mathbf{x}^{(k+1)} = T\mathbf{x}^{(k)} + \mathbf{d}$, by splitting the matrix A into a diagonal part, D, a strictly lower triangular part, L, and a strictly upper triangular part, U. Thus we have

$$A\mathbf{x} = (D - L - U)\mathbf{x} = \mathbf{c}, \tag{4.7}$$

which can be rewritten as

$$\mathbf{x} = D^{-1}(L+U)\mathbf{x} + D^{-1}\mathbf{c}. \tag{4.8}$$

This leads us to the Jacobi method expressed in matrix form

$$\mathbf{x}^{(k+1)} = D^{-1}(L+U)\mathbf{x}^{(k)} + D^{-1}\mathbf{c}. \tag{4.9}$$

The *Gauss-Seidel* iterative method offers a potential improvement over the Jacobi method. In the Jacobi method, to compute $x_i^{(k+1)}$, for each $i > 1$, the components of $\mathbf{x}^{(k)}$ are used. Since $x_1^{(k+1)}, ..., x_{i-1}^{(k+1)}$ have already been computed and are supposedly a better approximation to the actual solutions, $x_1, ..., x_{i-1}$ than $x_1^{(k)}, ..., x_{i-1}^{(k)}$, it seems reasonable to compute $x_i^{(k+1)}$ using these most recently calculated values. This gives us

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(-\sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} + c_i \right), \tag{4.10}$$

which can be expressed in matrix form as

$$\mathbf{x}^{(k+1)} = (D-L)^{-1}U\mathbf{x}^{(k)} + (D-L)^{-1}\mathbf{c}. \tag{4.11}$$

4.3 Convergence of Iterative Methods

To be of any use an iterative method must generate a sequence of approximations that converges. In this section, we will briefly discuss the convergence of iterative methods and their associated convergence rates. Most of this material has been

condensed from the work by Varga [2]. For a more complete presentation see this reference.

To begin we first define

$$\mathbf{e^{(k)}} \equiv \mathbf{x}^{(k)} - \mathbf{x}, \quad \text{for} \quad k \geq 0, \tag{4.12}$$

where $\mathbf{e}^{(k)}$ is the error associated with $\mathbf{x}^{(k)}$. In order to have convergence we want

$$\lim_{k \to \infty} \mathbf{e}^{(k)} = \mathbf{0}. \tag{4.13}$$

Subtracting (4.3) from (4.4) we obtain

$$\mathbf{e}^{(k+1)} = T\mathbf{e}^{(k)}. \tag{4.14}$$

From here, by induction, we obtain

$$\mathbf{e}^{(k)} = T^k \mathbf{e}^{(0)}, \quad \text{for} \quad k \geq 0. \tag{4.15}$$

Hence, convergence will always occur, using this iterative method, if

$$\lim_{k \to \infty} T^k \mathbf{e}^{(0)} = \mathbf{0}, \tag{4.16}$$

for any $\mathbf{e}^{(0)}$. This will occur if and only if

$$\lim_{k \to \infty} T^k = O, \tag{4.17}$$

where O is the $n \times n$ zero matrix. If equation (4.17) holds for some matrix T we say that T *converges*. If T represents an iterative method then we also say that *the iterative method converges*.

Before proceeding further some definitions are required.

definition 4.1 *A vector norm on $\Re^n$ is a function, $\| \cdot \|$, from $\Re^n$ into $\Re$ such that*

i) $\| \mathbf{x} \| \geq 0$ *for all* $\mathbf{x} \in \Re^n$,

ii) $\| \mathbf{x} \| = 0$ *if and only if* $\mathbf{x}=\mathbf{0}$,

iii) $\| \alpha\mathbf{x} \| = |\alpha| \, \| \mathbf{x} \|$ *for all* $\alpha \in \Re$ *and* $\mathbf{x} \in \Re^n$,

iv) $\| \mathbf{x} + \mathbf{y} \| \leq \| \mathbf{x} \| + \| \mathbf{y} \|$ *for all* $\mathbf{x}, \mathbf{y} \in \Re^n$.

example 4.1a: Let $\mathbf{x} \in \Re^n$. Then

$$\| \mathbf{x} \|_2 \equiv \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2} \tag{4.18}$$

is called the *Euclidean* or l_2 *norm* of $\mathbf{x}$.

example 4.1b: Let $\mathbf{x} \in \Re^n$. Then

$$\| \mathbf{x} \|_\infty \equiv \max\{|x_i| : i = 1, 2, ..., n\} \tag{4.19}$$

is called the l_∞ *norm* of $\mathbf{x}$.

definition 4.2 *Let T be an $n \times n$ matrix and $\| \cdot \|$ a vector norm on $\Re^n$. Then*

$$\| T \| \equiv sup \left\{ \frac{\| T\mathbf{x} \|}{\| \mathbf{x} \|} : \mathbf{x} \in \Re^n \text{ and } \mathbf{x} \neq \mathbf{0} \right\} \tag{4.20}$$

is called the induced matrix norm of T, relative to $\| \cdot \|$.

definition 4.3 *Let T be an $n \times n$ matrix with eigenvalues λ_i, $1 \leq i \leq n$. Then*

$$\rho(T) \equiv max\{|\lambda_i| : 1 \leq i \leq n\} \tag{4.21}$$

is called the spectral radius of T.

We present, without proof, the following

Theorem 4.1 *If T is an $n \times n$ complex matrix then*

$$\lim_{k \to \infty} T^k = O \text{ if and only if } \rho(T) < 1. \tag{4.22}$$

So, the spectral radius gives us a useful criterion for determining if an iterative method converges.

The relationship between the induced matrix norm and the spectral radius of a matrix T is expressed in the following

Theorem 4.2 *If T is an $n \times n$ complex matrix then*

$$\rho(T) \leq \| T \| . \tag{4.23}$$

Therefore, if $\| T \| < 1$ the iterative method represented by T converges.

We now wish to briefly discuss methods for estimating the rate at which an iterative method converges. Taking norms of both sides of equation (4.14) we have

$$\begin{aligned} \| \mathbf{e}^{(k+1)} \| &= \| T\mathbf{e}^{(k)} \| \\ &\leq \| T \| \cdot \| \mathbf{e}^{(k)} \|, \text{ for all } k \geq 0. \end{aligned} \tag{4.24}$$

If $\mathbf{e}^{(k)} \neq \mathbf{0}$ then we have the following

definition 4.4

$$\kappa = \frac{\| \mathbf{e}^{(k+1)} \|}{\| \mathbf{e}^{(k)} \|} \tag{4.25}$$

is called a convergence factor for T.

This factor is a measurement of the rate at which the error is being reduced on the k^{th} iteration of the iterative process represented by T. $\| T \|$, being an upper bound for this factor for all $\mathbf{e}$ and all $k \geq 0$, is often used to approximate the rate at which the associated method converges. Another upper bound frequently used for this purpose is the spectral radius, $\rho(T)$. Furthermore, the closer $\| T \|$ or $\rho(T)$ is to 1 the slower the method, while the closer $\| T \|$ or $\rho(T)$ is to 0, the faster the method.

As a representative example consider Poisson's equation on the unit square with zero boundary conditions, that is

$$\begin{aligned} \Delta u &= -f \quad \text{on} \quad \Omega = (0,1) \times (0,1) \\ u &= 0 \quad \text{on} \quad \Gamma = \text{ boundary } (\Omega). \end{aligned} \tag{4.26}$$

As a finite difference equation we will obtain

$$4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} = h^2 f_{i,j}, \tag{4.27}$$

for $0 < i, j < n$, with associated boundary conditions

$$u_{i,0} = u_{i,n} = u_{0,j} = u_{n,j} = 0 \tag{4.28}$$

for $0 < i, j < n$, where $h = 1/n$ is the grid spacing. We can rewrite (4.27) as

$$u_{i,j} = \frac{1}{4}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}) + \frac{1}{4}h^2 f_{i,j}, \tag{4.29}$$

This gives us the Jacobi iterative formula

$$u_{i,j}^{(k+1)} = \frac{1}{4}\left(u_{i+1,j}^{(k)} + u_{i-1,j}^{(k)} + u_{i,j+1}^{(k)} + u_{i,j-1}^{(k)}\right) + \frac{1}{4}h^2 f_{i,j}, \tag{4.30}$$

which can expressed in matrix form as

$$\mathbf{u}^{(k+1)} = D^{-1}(L+U)\mathbf{u}^{(k)} + D^{-1}h^2\mathbf{f}^{(k)} = T\mathbf{u}^{(k)} + D^{-1}h^2\mathbf{f}^{(k)}. \tag{4.31}$$

We now will find the spectral radius of the matrix T. First, we will find a complete set of eigenvalues for T. To determine an eigenvalue λ of T, we want an $(n-1)\times(n-1)$ component vector $\mathbf{v}$ and a scalar λ, such that

$$T\mathbf{v} = \lambda\mathbf{v}. \tag{4.32}$$

This will be accomplished if we find $\mathbf{v}$ and λ such that

$$\lambda v_{i,j} = \frac{1}{4}\left(v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1}\right) \tag{4.33}$$

for $0 < i, j < n$, and

$$v_{i,0} = v_{i,n} = v_{0,j} = v_{n,j} = 0 \tag{4.34}$$

for $0 < i, j < n$. The conditions imposed by (4.33) and (4.34) are met if

$$v_{i,j} = \sin\frac{p\pi i}{n} \cdot \sin\frac{q\pi j}{n} \tag{4.35}$$

and

$$\lambda = \frac{1}{2}\left(\cos\frac{p\pi}{n} + \cos\frac{q\pi}{n}\right), \tag{4.36}$$

where $0 < p, q < n$. The maximum $|\lambda|$ occurs when $p = q = 1$. Hence, we have

$$\rho(T) = \frac{1}{2}\left(\cos\frac{1\cdot\pi}{n} + \cos\frac{1\cdot\pi}{n}\right) = \cos\frac{\pi}{n} < 1, \tag{4.37}$$

and the iterative method converges.

In a similar fashion one can examine the Gauss-Seidel method, which has a matrix representation $S = (D - L)^{-1}U$. Here we wish to solve

$$S\mathbf{w} = \mu\mathbf{w}. \tag{4.38}$$

From here we obtain

$$w_{i,j} = |\lambda|^{i+j}v_{i,j} = |\lambda|^{i+j}\sin\frac{p\pi i}{n}\sin\frac{q\pi j}{n} \tag{4.39}$$

and

$$\mu = \lambda^2 = \frac{1}{4}\left(\cos\frac{p\pi}{n} + \cos\frac{q\pi}{n}\right)^2, \tag{4.40}$$

where $0 < p, q < n$. The maximum $|\mu|$ occurs when $p = q = 1$. Hence, we have

$$\rho(S) = \frac{1}{4}\left(\cos\frac{1\cdot\pi}{n} + \cos\frac{1\cdot\pi}{n}\right)^2 = \cos^2\frac{\pi}{n} < 1, \tag{4.41}$$

and this method also converges. Note however that $\rho(S) < \rho(T)$ so the Jacobi method must converge more slowly than the Gauss-Seidel method.

It is of interest to note that the larger n the closer the spectral radius of either method is to 1. This means that the more accurate the discretization (i.e. the finer the grid) the slower these two iterative methods converge.

It is also of interest that the spectral radius, is not always the best (i.e. smallest) bound on the convergence factor.

To demonstrate this we return to equation (4.14) where, expanding $\mathbf{e}^{(k)}$ as a linear combination of the set V of eigenvectors, we obtain

$$\begin{aligned}
\mathbf{e}^{(k+1)} &= T\mathbf{e}^{(k)} \qquad (4.42)\\
&= T\left(\sum_{\mathbf{v}_m \in V} a_m \mathbf{v}_m\right)\\
&= \sum_{\mathbf{v}_m \in V} a_m T\mathbf{v}_m\\
&= \sum_{\mathbf{v}_m \in V} a_m \lambda_m \mathbf{v}_m\\
&\leq \sum_{\mathbf{v}_m \in V} \max\{|\lambda_m| : a_m \neq 0\} \cdot a_m \mathbf{v}_m\\
&= \max\{|\lambda_m| : a_m \neq 0\} \cdot \left(\sum_{\mathbf{v}_m \in V} a_m \mathbf{v}_m\right)\\
&= \max\{|\lambda_m| : a_m \neq 0\} \cdot \mathbf{e}^{(k)}
\end{aligned}$$

Taking norms of both sides

$$\begin{aligned}
\left|\left|\mathbf{e}^{(k+1)}\right|\right| &\leq \left|\left|\max\{|\lambda_m| : a_m \neq 0\} \cdot \mathbf{e}^{(k)}\right|\right| \qquad (4.43)\\
&= \max\{|\lambda_m| : a_m \neq 0\} \cdot \left|\left|\mathbf{e}^{(k)}\right|\right|
\end{aligned}$$

So, we have

$$\kappa = \frac{\left|\left|\mathbf{e}^{(k+1)}\right|\right|}{||\mathbf{e}^{(k)}||} \leq \max\{|\lambda_m| : a_m \neq 0\} \qquad (4.44)$$

Two observations are pertinent at this point. First of all, notice that the convergence factor, and the resulting rate of convergence, is determined by the eigenvector

components that make up $\mathbf{e}^{(k)}$. Secondly, and more importantly for this thesis, for any error vector $\mathbf{e}^{(k)}$, different eigenvector components are reduced at different rates. For example, using the Jacobi method with $n = 64$, the eigenvector

$$v_{i,j} = \sin \frac{1\pi i}{64} \cdot \sin \frac{3\pi j}{64} \tag{4.45}$$

has a bound on its associated convergence factor of

$$\lambda = \frac{1}{2}\left(\cos \frac{1\pi}{64} + \cos \frac{3\pi}{64}\right) \approx 0.994 \tag{4.46}$$

On the other hand, the eigenvector

$$v_{i,j} = \sin \frac{29\pi i}{64} \cdot \sin \frac{31\pi j}{64} \tag{4.47}$$

has a bound on its associated convergence factor of

$$\lambda = \frac{1}{2}\left(\cos \frac{29\pi}{64} + \cos \frac{31\pi}{64}\right) \approx 0.098 \tag{4.48}$$

Hence, one component is reduced significantly faster than the other. The multigrid method, which is discussed in the next chapter, capitalizes on this very idea.

Chapter 5

A Sequential Multigrid Method

We have noted that the error at any step of an iterative method can be expressed as a linear combination of eigenvectors. A difficulty associated with methods such as the Gauss-Seidel and Jacobi methods is that certain eigenvector components of this error are reduced at a significantly slower rate than other components. This problem, demonstrated at the end of the previous chapter, results in an overall slow rate of convergence for the method. The multigrid method, presented in this chapter, augments a basic method, such as Gauss-Seidel or Jacobi, and alleviates this problem.

5.1 Residual Correction Iterative Methods

In this section we introduce the basic idea of multigrid methods. We begin with a discretized form of an elliptic partial differential on a grid G_h with uniform grid spacing h,

$$A_h \mathbf{u}_h = \mathbf{f}_h. \tag{5.1}$$

We assume that $\mathbf{f}_h$ incorporates both the right hand side of the difference equation as well as the boundary conditions.

Before discussing the multigrid method we will introduce the more general idea of a residual correction iterative methodology. Let $\mathbf{u}_h^{(k)}$ be an approximation to the solution $\mathbf{u}_h$ of equation 5.1, obtained on the k^{th} iteration of some iterative method. The error $\mathbf{e}_h^{(k)}$ associated with $\mathbf{u}_h^{(k)}$ is given by

$$\mathbf{e}_h^{(k)} = \mathbf{u}_h - \mathbf{u}_h^{(k)}. \tag{5.2}$$

The *residual* $\mathbf{r}_h^{(k)}$ of $\mathbf{u}_h^{(k)}$ is defined as

$$\mathbf{r}_h^{(k)} = A_h \mathbf{e}_h^{(k)} = A_h \mathbf{u}_h - A_h \mathbf{u}_h^{(k)} = \mathbf{f}_h - A_h \mathbf{u}_h^{(k)}. \tag{5.3}$$

Notice that solving for $\mathbf{e}_h^{(k)}$ in the residual equation is equivalent to solving for $\mathbf{u}_h$ in the original equation (5.1), since

$$\mathbf{u}_h = \mathbf{u}_h^{(k)} + \mathbf{e}_h^{(k)}. \tag{5.4}$$

Many iterative methods can be expressed in terms of approximations to the residual equation. In these cases the matrix A_h is replaced by an approximation

$\tilde{A}_h$ such that $\tilde{A}_h^{-1}$ exists and the product of $\tilde{A}_h^{-1}$ and an arbitrary vector is easy to compute. The solution $\tilde{\mathbf{e}}_h^{(k)}$ of

$$\tilde{A}_h \tilde{\mathbf{e}}_h^{(k)} = \mathbf{r}_h^{(k)} \tag{5.5}$$

yields a new approximation

$$\mathbf{u}_h^{(k+1)} = \mathbf{u}_h^{(k)} + \tilde{\mathbf{e}}_h^{(k)} \tag{5.6}$$

to the solution of the original equation (5.1). Such a method as described above is called a *residual correction iterative method.*

Expanding (5.6) we obtain

$$\begin{aligned} \mathbf{u}_h^{(k+1)} &= \mathbf{u}_h^{(k)} + \tilde{\mathbf{e}}_h^{(k)} \\ &= \mathbf{u}_h^{(k)} + \tilde{A}_h^{-1}\mathbf{r}_h^{(k)} \\ &= \mathbf{u}_h^{(k)} + \tilde{A}_h^{-1}(\mathbf{f}_h - A_h\mathbf{u}_h^{(k)}) \\ &= \mathbf{u}_h^{(k)} + \tilde{A}_h^{-1}\mathbf{f}_h - \tilde{A}_h^{-1}A\mathbf{u}_h^{(k)} \\ &= (I_h - \tilde{A}_h^{-1}A_h)\mathbf{u}_h^{(k)} + \tilde{A}_h^{-1}\mathbf{f}_h. \end{aligned} \tag{5.7}$$

Hence, the matrix representing this iterative method is $I_h - \tilde{A}_h^{-1}A_h$ and the error associated with the method is

$$\mathbf{e}_h^{(k+1)} = (I_h - \tilde{A}_h^{-1}A_h)\mathbf{e}_h^{(k)}, \tag{5.8}$$

with a convergence factor

$$\frac{\| \mathbf{e}_h^{(k+1)} \|}{\| \mathbf{e}_h^{(k)} \|} \leq \| I_h - \tilde{A}_h^{-1}A_h \| . \tag{5.9}$$

5.2 A Coarse Grid Residual Correction Scheme

In this section we introduce the simplest form of a multigrid method, a two-grid method. In this method we use a second grid G_{2h} in addition to G_h. G_{2h} has a grid spacing twice the size as that of G_h. Other sizes for grid spacing on the coarse grid are possible but we have chosen the most convenient. Using this method, the correction term $\mathbf{e}_h^{(k)}$ is obtained by solving the residual equation on G_{2h}. That is, we solve

$$\tilde{A}_{2h}\tilde{\mathbf{e}}_{2h}^{(k)} = \mathbf{r}_{2h}^{(k)}, \tag{5.10}$$

where $\tilde{A}_{2h}$ is an appropriate coarse grid approximation to A_h, and $\tilde{\mathbf{e}}_{2h}^{(k)}$ and $\mathbf{r}_{2h}^{(k)}$ are coarse grid approximations to $\tilde{\mathbf{e}}_h^{(k)}$ and $\mathbf{r}_h^{(k)}$ respectively.

In order to move between grids we need two transfer operators

$$I_h^{2h} : G_h \rightarrow G_{2h} \tag{5.11}$$

and

$$I_{2h}^h : G_{2h} \rightarrow G_h. \tag{5.12}$$

I_h^{2h} is used to *restrict* $\mathbf{r}_h^{(k)}$ to G_{2h}, that is,

$$I_h^{2h}\mathbf{r}_h^{(k)} = \mathbf{r}_{2h}^{(k)}, \tag{5.13}$$

and I_{2h}^h is used to *interpolate* the correction term $\tilde{\mathbf{e}}_{2h}^{(k)}$ to G_h, that is,

$$I_{2h}^h\tilde{\mathbf{e}}_{2h}^{(k)} = \tilde{\mathbf{e}}_h^{(k)}. \tag{5.14}$$

One iteration step can be summarized by the following sequence of steps:

1. compute the residual on the fine grid G_h: $\mathbf{r}_h^{(k)} = \mathbf{f}_h - A_h\mathbf{u}_h^{(k)}$,

2. transfer the residual to the coarse grid G_{2h}: $\mathbf{r}_{2h}^{(k)} = I_h^{2h}\mathbf{r}_h^{(k)}$,

3. solve the residual equation on G_{2h}: $\tilde{\mathbf{e}}_{2h}^{(k)} = \tilde{A}_{2h}^{-1}\mathbf{r}_{2h}^{(k)}$,

4. transfer error term to the fine grid G_h: $\tilde{\mathbf{e}}_h^{(k)} = I_{2h}^h\tilde{\mathbf{e}}_{2h}^{(k)}$,

5. compute new approximation: $\mathbf{u}_h^{(k+1)} = \mathbf{u}_h^{(k)} + \tilde{\mathbf{e}}_h^{(k)}$.

So we have

$$\begin{aligned}
\mathbf{u}_h^{(k+1)} &= \mathbf{u}_h^{(k)} + \tilde{\mathbf{e}}_h^{(k)} \qquad (5.15)\\
&= \mathbf{u}_h^{(k)} + I_{2h}^h\tilde{\mathbf{e}}_{2h}^{(k)}\\
&= \mathbf{u}_h^{(k)} + I_{2h}^h\tilde{A}_{2h}^{-1}\mathbf{r}_{2h}^{(k)}\\
&= \mathbf{u}_h^{(k)} + I_{2h}^h\tilde{A}_{2h}^{-1}I_h^{2h}\mathbf{r}_h^{(k)}\\
&= \mathbf{u}_h^{(k)} + I_{2h}^h\tilde{A}_{2h}^{-1}I_h^{2h}(\mathbf{f}_h - A_h\mathbf{u}_h^{(k)})\\
&= \mathbf{u}_h^{(k)} - I_{2h}^h\tilde{A}_{2h}^{-1}I_h^{2h}A_h\mathbf{u}_h^{(k)} + I_{2h}^h\tilde{A}_{2h}^{-1}I_h^{2h}\mathbf{f}_h\\
&= (I_h - I_{2h}^h\tilde{A}_{2h}^{-1}I_h^{2h}A_h)\mathbf{u}_h^{(k)} + I_{2h}^h\tilde{A}_{2h}^{-1}I_h^{2h}\mathbf{f}_h.
\end{aligned}$$

Thus we see that the iteration matrix for coarse grid correction is given by $T = (I_h - I_{2h}^h\tilde{A}_{2h}^{-1}I_h^{2h}A_h)$, the error associated with the method is

$$\mathbf{e}_h^{(k+1)} = T\mathbf{e}_h^{(k)}, \qquad (5.16)$$

and the associated convergence factor is

$$\frac{\| \mathbf{e}_h^{(k+1)} \|}{\| \mathbf{e}_h^{(k)} \|} \leq \| T \| . \tag{5.17}$$

A problem exists, however, with this method used alone. There can exist $\mathbf{v}_h \in G_h$ such that $I_h^{2h} A_h \mathbf{v}_h = 0$. Hence $T\mathbf{v}_h = \mathbf{v}_h$. So, we have $\| T \| \geq 1$ and the coarse grid correction method does not converge. The difficulty lies in the fact that the residual equation on the coarse grid

$$\tilde{A}_{2h} \tilde{\mathbf{e}}_{2h}^{(k)} = \mathbf{r}_{2h}^{(k)}, \tag{5.18}$$

is not a good enough approximation to the residual equation on the fine grid

$$A_h \mathbf{e}_h^{(k)} = \mathbf{r}_h^{(k)}. \tag{5.19}$$

In particular, the problem is that certain components of $\mathbf{e}_h^{(k)}$ cannot be accurately enough represented on the coarse grid G_{2h}.

We can be observe this by considering a case where G_h is a 64×64 grid, G_{2h} is a 32×32 grid, and $\mathbf{e}_h^{(k)}$ is expressed as a linear combination of terms of the form

$$\sin p\pi ih \cdot \sin q\pi jh, \tag{5.20}$$

where p and q are integers such that $-32 < p, q < 32$. That is,

$$\mathbf{e}_h^{(k)} = \sum_{p=-31}^{31} \sum_{q=-31}^{31} c_{pq} \sin p\pi ih \cdot \sin q\pi jh. \tag{5.21}$$

Note that on grid G_{2h} the only components of $\mathbf{e}_h^{(k)}$ which are visible are those such that $-16 < p, q < 16$. The other components are not visible. This phenomena is

explained by Shannon's Theorem [6], the fundamental theorem of signal processing, which tells us that the only terms that will be visible on a given grid will be those that that contain only frequencies (p and q) that are less than one-half the frequency n of the grid spacing.

Components of $\mathbf{e}_h^{(k)}$ such that $-16 < p, q < 16$ are referred to as *low frequency* or *smooth* error. Components outside this region, $-32 < p, q \leq -16$ or $16 \leq p, q < 32$, are referred to as *high frequency* error. Hence, $\mathbf{e}_h^{(k)}$ can be well approximated on a grid G_{2h} only if it is primarily composed of smooth error, that is, its high frequency components are small compared to its low frequency components.

5.3 Smoothing the High Frequency Error

A multigrid method uses coarse grid correction by first reducing the high frequency error components. This is referred to as *smoothing the error.* Only after the error has been smoothed is a coarse grid correction scheme performed.

It so happens that many iterative methods, such as the Gauss-Seidel and the Jacobi methods, have the characteristic that they reduce high frequency error very efficiently, in fact, much more efficiently than they reduce low frequency error. This fact was demonstrated in the example at the end of the last chapter for the Jacobi method. In this example we saw that a high frequency term, where $p = 29$ and $q = 31$, had an associated convergence factor of 0.098 while a low frequency term,

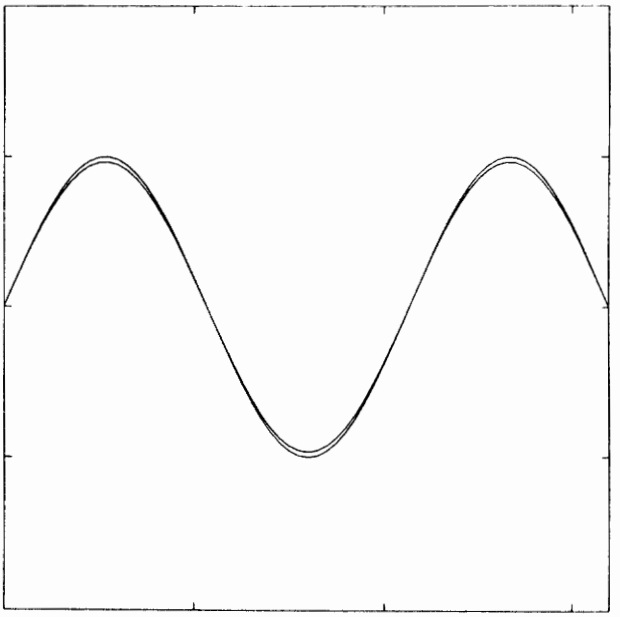

Figure 5.1: low frequency

where $p = 1$ and $q = 3$, had an associated convergence factor of 0.994. As a further illustration of this fact see the figures included in this chapter. In each figure we show two graphs of cross sections of error terms on a 64×64 grid. Each cross section is taken at $i = 32$. The higher and lower amplitude graphs represent the same error term before and after three iterations of the Gauss-Seidel method. Figure 5.1 represents the effect of Gauss-Seidel method on low frequency error terms. The higher amplitude graph represents

$$\mathbf{e} = \sin(1\pi ih) \cdot \sin(3\pi jh), \tag{5.22}$$

while the lower amplitude graph represents

$$S^3\mathbf{e} = S^3[\sin(1\pi ih) \cdot \sin(3\pi jh)], \tag{5.23}$$

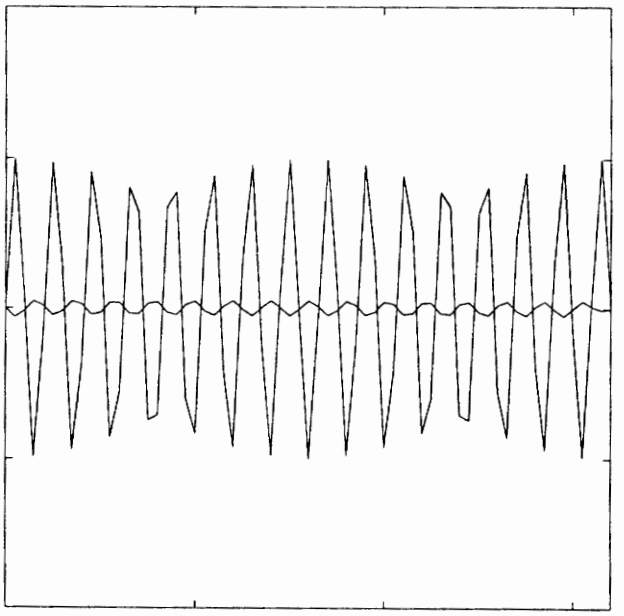

Figure 5.2: high frequency

where S is the matrix representing one iteration of the Gauss-Seidel method and $h = \frac{1}{64}$. Similarly, figure 5.2 represents the effect on high frequency error terms. Here we have

$$\mathbf{e} = \sin(29\pi ih) \cdot \sin(31\pi jh) \tag{5.24}$$

and

$$S^3\mathbf{e} = S^3[\sin(29\pi ih) \cdot \sin(31\pi jh)]. \tag{5.25}$$

Figure 5.3 shows the effect on a combination of two frequencies, where

$$\mathbf{e} = \sin(1\pi ih) \cdot \sin(3\pi jh) + \sin(29\pi ih) \cdot \sin(31\pi jh) \tag{5.26}$$

and

$$S^3\mathbf{e} = S^3[\sin(1\pi ih) \cdot \sin(3\pi jh) + \sin(29\pi ih) \cdot \sin(31\pi jh)]. \tag{5.27}$$

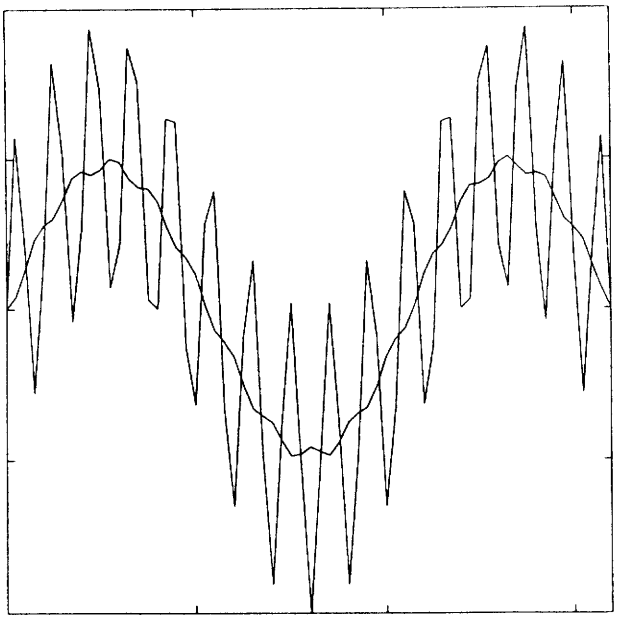

Figure 5.3: mixed frequencies

Notice in these figures how significantly the higher frequency error is reduced with only 3 iterations of the Gauss-Seidel method while the lower frequency error is hardly affected.

5.4 Increasing the Convergence Rate For Lower Frequencies

After reducing or *smoothing* the high frequency error, the remaining error can then be transferred to the coarser grid. This transfer is now effective since the remaining error is of low frequency and can reasonably be approximated on the coarser grid.

Moreover, transferring the low frequency error to the coarser grid has the desired effect of increasing its frequency relative to the coarser grid. This makes the remaining error conveniently susceptible to further reduction using a method such as Gauss-Seidel or Jacobi on the coarser grid.

5.5 A Two-Grid Multigrid Method

A basic two-grid multigrid method begins with the choice of the two grids to be used. In the results presented in this section we have used a 64×64 and a 32×32 grid. As a model problem, for this algorithm and those to follow we are using Example 4.3.1, that is we wish to solve Poisson's equation

$$\Delta u = -f \quad \text{on} \quad \Omega = (0,1) \times (0,1) \tag{5.28}$$

$$u = \phi \quad \text{on} \quad \Gamma = \text{boundary}(\Omega).$$

We first obtain a discretization of the problem on the fine grid, which we will express as,

$$A_h \mathbf{u}_h = \mathbf{f}_h, \tag{5.29}$$

where $h = \frac{1}{64}$. We then proceed with the algorithm as follows:

ALGORITHM 5.1

(A Two-Grid Method)

step i:

Set the iteration counter k at 0.

Guess $\mathbf{u}_h^{(0)}$ as an approximation to $\mathbf{u}_h$.

step ii:

Repeat until $\|\mathbf{u}_h^{(k)} - \mathbf{u}_h\| \leq \varepsilon$, for some iteration k and some error tolerance ε:

— Perform three Gauss-Seidel sweeps on $A_h\mathbf{u}_h^{(k)} = \mathbf{f}_h$.

— Compute the residual: $\mathbf{r}_h^{(k)} = A_h\mathbf{u}_h^{(k)} - \mathbf{f}_h$.

— Transfer the residual to the coarse grid: $\mathbf{r}_{2h}^{(k)} = I_h^{2h}\mathbf{r}_h^{(k)}$.

We use injection for transferring the residual from the fine to the coarse grid.

— Approximate the solution to the coarse grid residual equation: $A_{2h}\mathbf{e}_{2h}^{(k)} = \mathbf{r}_{2h}^{(k)}$.

This yields us $\tilde{\mathbf{e}}_{2h}^{(k)}$, a good approximation to $\mathbf{e}_{2h}^{(k)}$. We accomplish this with 30 iterations of Gauss-Seidel, using an initial guess $\mathbf{e}_{2h}^{(k)} = \mathbf{0}$.

— Transfer the error term to the fine grid: $\tilde{\mathbf{e}}_h^{(k)} = I_{2h}^h\tilde{\mathbf{e}}_{2h}^{(k)}$.

We use bilinear interpolation to transfer the error term from the coarse to the fine grid.

— Compute new approximation: $\mathbf{u}_h^{(k+1)} = \mathbf{u}_h^{(k)} + \tilde{\mathbf{e}}_h^{(k)}$.

— Increment the iteration counter k.

step iii:

Perform three Gauss-Seidel sweeps on $A_h \mathbf{u}_h^{(k)} = \mathbf{f}_h$.

This smooths out any high frequency error introduced on the final iteration by the interpolation operator. □

One iteration of the multigrid method is called a *V-cycle*. If S is the matrix representing one iteration of the basic *smoothing* iterative method then the matrix representing one iteration (i.e. one V-cycle) of the two-grid multigrid method is given by

$$S_h^{l_2}(I_h - I_{2h}^h \tilde{A}_{2h}^{-1} I_h^{2h} A_h) S_h^{l_1}, \tag{5.30}$$

where l_1 and l_2 are the numbers of iterations of the smoothing iterative method that are applied before and after the coarse grid correction process. So, in matrix form the algorithm can be expressed as

$$[S_h^{l_2}(I_h - I_{2h}^h \tilde{A}_{2h}^{-1} I_h^{2h} A_h) S_h^{l_1}]^k. \tag{5.31}$$

5.6 A Full Multigrid Method

A full multigrid method is a simple extension of the basic method. The idea is that in solving the problem on the coarse level an even coarser level can be used. In fact, an entire hierarchy of different size grids can be used.

One V-cycle for a full multigrid method can be defined recursively by

$$T_1 = S_1^{l_2}(I_1 - I_0^1 \tilde{A}_0^{-1} I_1^0 \tilde{A}_1) S_1^{l_1}, \tag{5.32}$$

and

$$T_p = S_p^{l_2}(I_p - I_{p-1}^p (I_{p-1} - T_{p-1}^m)\tilde{A}_{p-1}^{-1} I_p^{p-1} \tilde{A}_p) S_p^{l_1}, \tag{5.33}$$

where p denotes the p^{th} finest level and if q is the finest level then

$$\tilde{A}_q = A_q. \tag{5.34}$$

We present an example of a three-grid multigrid method. Here the coarse grid residual equation:

$$A_{2h}\mathbf{e}_{2h}^{(k)} = \mathbf{r}_{2h}^{(k)}, \tag{5.35}$$

is solved by computing the residual of this residual equation and transferring it to an even coarser grid (a 16×16 grid). We use a slightly different notation to assist us in explaining this method. We use $\mathbf{f}_{2h}^{(k)}$ rather than $\mathbf{r}_{2h}^{(k)}$ to represent the first residual computed on grid G_{2h}. Similarly, we use $\mathbf{u}_{2h}^{(k)}$ rather than $\mathbf{e}_{2h}^{(k)}$ to represent the error on grid G_{2h}. We use $\mathbf{f}_{4h}^{(k)}$ to represent the second residual (ie. the residual of the residual) computed on grid G_{4h}. Similarly, we use $\mathbf{u}_{4h}^{(k)}$ to represent the error on grid G_{4h}. We then proceed with the algorithm as follows:

ALGORITHM 5.2

(A Three-Grid Method)

step i:

Set the iteration counter k at 0.

Guess $\mathbf{u}_h^{(0)}$ as an approximation to $\mathbf{u}_h$.

step ii:

Repeat until $\|\mathbf{u}_h^{(k)} - \mathbf{u}_h\| \leq \varepsilon$, for some iteration k and some error tolerance ε:

— Perform three Gauss-Seidel sweeps on $A_h\mathbf{u}_h^{(k)} = \mathbf{f}_h$.

— Compute $\mathbf{f}_{2h}^{(k)} = I_h^{2h}\mathbf{r}_h^{(k)}$.

— Perform three Gauss-Seidel sweeps on $A_{2h}\mathbf{u}_{2h}^{(k)} = \mathbf{f}_{2h}^{(k)}$, with initial guess $\tilde{\mathbf{u}}_{2h}^{(k)} = \mathbf{0}$.

— Compute $\mathbf{f}_{4h}^{(k)} = I_{2h}^{4h}\mathbf{r}_{2h}^{(k)}$.

— Perform 30 Gauss-Seidel sweeps on $A_{4h}\mathbf{u}_{4h}^{(k)} = \mathbf{f}_{4h}^{(k)}$, with initial guess $\tilde{\mathbf{u}}_{4h}^{(k)} = \mathbf{0}$.

— Correct $\mathbf{u}_{2h}^{(k)} = \mathbf{u}_{2h}^{(k)} + I_{4h}^{2h}\mathbf{u}_{4h}^{(k)}$.

— Perform three Gauss-Seidel sweeps on $A_{2h}\mathbf{u}_{2h}^{(k)} = \mathbf{f}_{2h}^{(k)}$, with initial guess $\tilde{\mathbf{u}}_{2h}^{(k)}$.

— Correct $\mathbf{u}_h^{(k)} = \mathbf{u}_h^{(k)} + I_{2h}^{h}\mathbf{u}_{2h}^{(k)}$.

— Increment the iteration counter k.

step iii:

Perform three Gauss-Seidel sweeps on $A_h\mathbf{u}_h^{(k)} = \mathbf{f}_h$. □

Table 5.1 at the end of this chapter shows some actual results using algorithms 5.1 and 5.2. For a comparison we also show results using a simple Gauss-Seidel method. The stopping criteria we use is $\| \mathbf{u}_h^{(k)} - \mathbf{u}_h \| \leq \varepsilon = 10^{-6}$. In the table, k is the number of iterations, and t is the cpu time in seconds.

For solving our model problem we see the speed of execution, relative to the Gauss-Seidel method, increase by factors of $\frac{164.0}{58.3} = 2.9$ and $\frac{164.0}{41.0} = 4.0$ for algorithms 5.1 and 5.2 respectively. There are two factors contributing to this improved performance. First of all, as we have already discussed, when the problem is transferred to a coarser grid the lower frequency components of the error are increased, relative to the coarser grid, resulting in an improved convergence rate for those components. Secondly, a significant amount of the work of the multigrid algorithms is performed on the coarser grids in executing Gauss-Seidel iterations for residual equations.. Relative to the work required to perform a Gauss-Seidel iteration on the fine grid G_h, a Gauss-Seidel iteration on G_{2h} requires only $\frac{1}{4}$ of the work and on G_{4h} only $\frac{1}{16}$ of the work.

Though initial studies introducing and investigating multigrid methods are credited to Fedorenka [7, 8] in 1962-64 and Bakvalow [9] in 1966, the recognition of the efficiency and potential power of multigrid methods is due to A. Brandt [10] in 1976. Since Brandt's original paper, multigrid methods have gradually become recognized

as offering significant possibilities in the area of numerical methods for partial differential equations [11]. For an introduction to multigrid methods the book by Briggs [12] is very good. For a more complete treatment see the article by Stuben and Trottenburg [11].

Table 5.1: Sequential Multigrid and Gauss-Seidel Methods

method	k	t
Gauss-Seidel	6250	164
Algorithm 5.1	209	58.3
Algorithm 5.2	239	41

Chapter 6

A Parallel Domain Decomposition Method

In the previous chapter we introduced an essentially sequential method that utilized multiple grids in order to accelerate the convergence of a basic iterative method (such as the Gauss-Seidel method). In this chapter we take a different approach while continuing to use the same basic iterative method. Here, rather than attempting to increase the rate of convergence we break the problem up into a number of subproblems and solve the subproblems simultaneously on multiple processors. In this case, the convergence rate is not improved yet the execution speed is decreased due to the use of multiprocessing.

A numerical parallel Schwarz method is an iterative method which involves decomposing the domain over which a partial differential equation is to be solved into multiple overlapping subdomains. The subdomains are then distributed over multiple processors. Then, using a standard iterative method, each processor works to solve a smaller version of the original problem for the specific subdomains to which that processor has been assigned. The processors work in parallel. Information exchanges between any two processors are primarily restricted to subdomain boundary value approximations for the overlapping regions of their respective subdomains.

6.1 The Schwarz Alternating Principle

A nonparallel form of this domain decomposition method was originally proposed in the 1860's by H. A. Schwarz for solving the Dirichlet problem for harmonic functions [13]. In 1890, Picard, who called the method the "Schwarz Alternating Procedure," used it to solve a nonlinear elliptic partial differential equation [14]. A good description of the Schwarz alternating procedure can be found in the book by Kantorovich and Krylow [15]. Numerical analogs to this method were developed by K. Miller [16]. The suitability of variants of the method as parallel algorithms has been explored in recent years by Kang [17] and Rodrigue and Simon [18, 19].

We will now introduce the sequential Schwarz alternating procedure, using two

subdomains, to solve Poisson's equation

$$\Delta u = -f \quad \text{on} \quad \Omega = (0,1) \times (0,1) \tag{6.1}$$

$$u = \phi \quad \text{on} \quad \Gamma = \text{boundary}(\Omega).$$

We begin by partitioning the domain Ω into two overlapping subdomains Ω_1 and Ω_2 such that $\Omega = \Omega_1 \bigcup \Omega_2$. Let $0 = x_{l_1} < x_{l_2} \leq x_{r_1} < x_{r_2} = 1$. Then define $\Omega_1 = (x_{l_1}, x_{r_1}) \times (0,1)$ and $\Omega_2 = (x_{l_2}, x_{r_2}) \times (0,1)$. The boundary of each subdomain will include part of the original boundary, $\Gamma_i \cap \Gamma$, as well as a new part, called a *pseudo-boundary*, $\Gamma_i - \Gamma$. See the figure below. Next, we solve the problem on

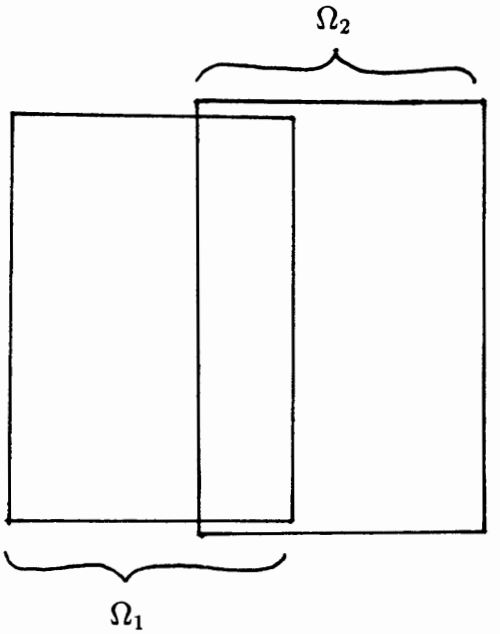

Figure 6.1: $\Omega = \Omega_1 \cup \Omega_2 = (0,1) \times (0,1)$

subdomain Ω_1, using a guess Ψ for the solution on the pseudo-boundary, $\Gamma_1 - \Gamma$. We then solve the problem on subdomain Ω_2, using values we obtained in solving the

subdomain Ω_1 problem as the boundary values on pseudo-boundary, $\Gamma_2 - \Gamma$. From there we go back to subdomain Ω_1 and solve the problem using updated values for the pseudo-boundary $\Gamma_1 - \Gamma$, obtained from subdomain Ω_2. We continue to alternate between the two subdomains in such a manner until the approximation is close enough for our needs.

The algorithm can be described as follows:

ALGORITHM 6.1

(A Sequential Method Using Two Subdomains)

step i:

Divide domain Ω into two subdomains Ω_1 and Ω_2, as described previously.

step ii:

Guess $u_1 = \psi$ on $\Gamma_1 \cap \Omega_2$.

step iii:

Initialize iteration counter k to 1.

Solve

$$\begin{aligned} \Delta u_1^{(k)} &= -f \quad \text{on} \quad \Omega_1 \\ u_1^{(k)} &= \phi \quad \text{on} \quad \Gamma_1 \cap \Gamma \\ u_1^{(k)} &= \psi \quad \text{on} \quad \Gamma_1 \cap \Omega_2. \end{aligned} \tag{6.2}$$

Then solve

$$\begin{aligned} \Delta u_2^{(k)} &= -f \quad \text{on} \quad \Omega_2 \\ u_2^{(k)} &= \phi \quad \text{on} \quad \Gamma_2 \cap \Gamma \\ u_2^{(k)} &= u_1^{(k)} \quad \text{on} \quad \Gamma_2 \cap \Omega_1. \end{aligned} \tag{6.3}$$

step iv:

Repeat until $\|u_1^{(k)} - u\| \leq \varepsilon$ on Ω_1 and $\|u_2^{(k)} - u\| \leq \varepsilon$ on Ω_2, for some iteration k and some error tolerance ε.

$k = k + 1$.

Solve

$$\begin{aligned} \Delta u_1^{(k)} &= -f \quad \text{on} \quad \Omega_1 \\ u_1^{(k)} &= \phi \quad \text{on} \quad \Gamma_1 \cap \Gamma \\ u_1^{(k)} &= u_2^{(k-1)} \quad \text{on} \quad \Gamma_1 \cap \Omega_2. \end{aligned} \tag{6.4}$$

Then solve

$$\begin{aligned} \Delta u_2^{(k)} &= -f \quad \text{on} \quad \Omega_2 \\ u_2^{(k)} &= \phi \quad \text{on} \quad \Gamma_2 \cap \Gamma \\ u_2^{(k)} &= u_1^{(k)} \quad \text{on} \quad \Gamma_2 \cap \Omega_1. \end{aligned} \tag{6.5}$$

step v:

Let $\tilde{u} = u_1^{(k)}$ on $\Omega_1 - \Omega_2$ and $\tilde{u} = u_2^{(k)}$ on Ω_2. □

Then $\|\tilde{u} - u\| < \varepsilon$ so $\tilde{u}$ is a suitable approximation to u.

6.2 A Parallel Schwarz Method

Because of the dependency that the solution on one subdomain of Algorithm 6.1 has on the previous solution of the other subdomain, this algorithm is inherently sequential. By breaking the domain into more than two subdomains we can use a similar scheme and obtain a parallel method. The first parallel method we consider uses 4 subdomains and 2 processors. See figure below.

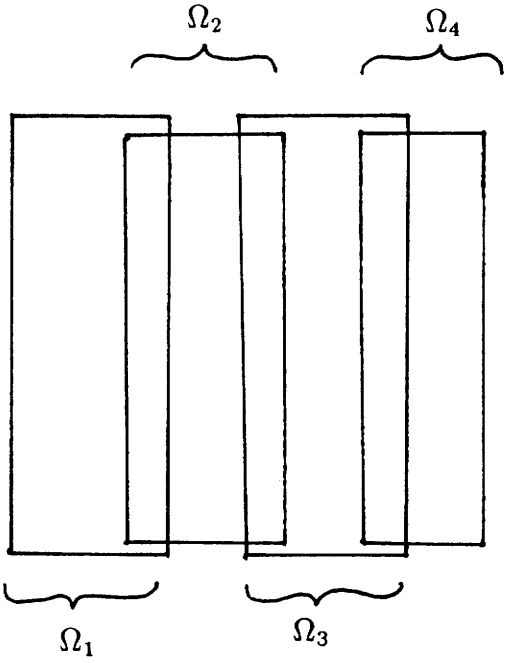

Figure 6.2: $\Omega = \bigcup_{i=1}^{4} \Omega_i = (0,1) \times (0,1)$

The algorithm can be described as follows:

ALGORITHM 6.2

(A Parallel Method Using Four Subdomains and Two Processors)

step i:

Define two sequences of real numbers,

$$\{x_{r_i}\}_{i=1}^{4} \quad \text{and} \quad \{x_{l_i}\}_{i=1}^{4} \tag{6.6}$$

such that

$$0 = x_{l_1} < x_{l_2} \leq x_{r_1} < x_{l_3} \leq x_{r_2} < x_{l_4} \leq x_{r_3} < x_{r_4} = 1. \tag{6.7}$$

Then define

$$\Omega_i = (x_{l_i}, x_{r_i}) \times (0,1), \quad \text{for} \quad i = 1,2,3,4. \tag{6.8}$$

step ii:

Guess $u_1 = \psi_1$ on $\Gamma_1 - \Gamma$,

and $u_3 = \psi_3$ on $\Gamma_3 - \Gamma$.

step iii:

Initialize iteration counter k to 1.

Solve

$$\begin{aligned} \Delta u_1^{(k)} &= -f \quad \text{on} \quad \Omega_1 \\ u_1^{(k)} &= \phi \quad \text{on} \quad \Gamma_1 \cap \Gamma \\ u_1^{(k)} &= \psi_1 \quad \text{on} \quad \Gamma_1 \cap \Omega, \end{aligned} \tag{6.9}$$

on processor 1, while solving

$$\begin{aligned} \Delta u_3^{(k)} &= -f \quad \text{on} \quad \Omega_3 \\ u_3^{(k)} &= \phi \quad \text{on} \quad \Gamma_3 \cap \Gamma \\ u_3^{(k)} &= \psi_3 \quad \text{on} \quad \Gamma_3 \cap \Omega, \end{aligned} \tag{6.10}$$

on processor 2.

Then solve

$$\Delta u_2^{(k)} = -f \quad \text{on} \quad \Omega_2 \tag{6.11}$$

$$\begin{aligned} u_2^{(k)} &= \phi \quad \text{on} \quad \Gamma_2 \cap \Gamma \\ u_2^{(k)} &= u_1^{(k)} \quad \text{on} \quad \Gamma_2 \cap \Omega_1 \\ u_2^{(k)} &= u_3^{(k)} \quad \text{on} \quad \Gamma_2 \cap \Omega_3, \end{aligned}$$

on processor 1, while solving

$$\begin{aligned} \Delta u_4^{(k)} &= -f \quad \text{on} \quad \Omega_4 \\ u_4^{(k)} &= \phi \quad \text{on} \quad \Gamma_4 \cap \Gamma \\ u_4^{(k)} &= u_3^{(k)} \quad \text{on} \quad \Gamma_4 \cap \Omega_3, \end{aligned} \tag{6.12}$$

on processor 2.

step iv:

Repeat until $\|u_i - u\| \leq \varepsilon$ on Ω_i, for $i = 1, 2, 3, 4$, for any iteration k and for some error tolerance ε.

$k = k + 1$.

Solve

$$\begin{aligned} \Delta u_1^{(k)} &= -f \quad \text{on} \quad \Omega_1 \\ u_1^{(k)} &= \phi \quad \text{on} \quad \Gamma_1 \cap \Gamma \\ u_1^{(k)} &= u_2^{(k-1)} \quad \text{on} \quad \Gamma_1 \cap \Omega_2, \end{aligned} \tag{6.13}$$

on processor 1, while solving

$$\Delta u_3^{(k)} = -f \quad \text{on} \quad \Omega_3 \tag{6.14}$$

$$
\begin{aligned}
u_3^{(k)} &= \phi \quad \text{on} \quad \Gamma_3 \cap \Gamma \\
u_3^{(k)} &= u_2^{(k-1)} \quad \text{on} \quad \Gamma_3 \cap \Omega_2 \\
u_3^{(k)} &= u_4^{(k-1)} \quad \text{on} \quad \Gamma_3 \cap \Omega_4,
\end{aligned}
$$

on processor 2.

Then solve

$$
\begin{aligned}
\Delta u_2^{(k)} &= -f \quad \text{on} \quad \Omega_2 \qquad (6.15) \\
u_2^{(k)} &= \phi \quad \text{on} \quad \Gamma_2 \cap \Gamma \\
u_2^{(k)} &= u_1^{(k)} \quad \text{on} \quad \Gamma_2 \cap \Omega_1 \\
u_2^{(k)} &= u_3^{(k)} \quad \text{on} \quad \Gamma_2 \cap \Omega_3,
\end{aligned}
$$

on processor 1, while solving

$$
\begin{aligned}
\Delta u_4^{(k)} &= -f \quad \text{on} \quad \Omega_4 \qquad (6.16) \\
u_4^{(k)} &= \phi \quad \text{on} \quad \Gamma_4 \cap \Gamma \\
u_4^{(k)} &= u_3^{(k)} \quad \text{on} \quad \Gamma_4 \cap \Omega_3,
\end{aligned}
$$

on processor 2.

step v:

Let $\tilde{u} = u_1^{(k)}$ on $\Omega_1 - \Omega_2$,

$\tilde{u} = u_2^{(k)}$ on Ω_2,

$\tilde{u} = u_3^{(k)}$ on $\Omega_3 - (\Omega_2 \cup \Omega_4)$,

and $\tilde{u} = u_4^{(k)}$ on Ω_4. $\square$

Then $\|\tilde{u} - u\| < \varepsilon$ so $\tilde{u}$ is a suitable approximation to u.

We can generalize Algorithm 6.2 and obtain a parallel method which uses $2n$ subdomains and n processors.

This algorithm can be described as follows:

ALGORITHM 6.3

(A Parallel Method Using 2n Subdomains and n Processors)

step i:

Define two sequences of real numbers,

$$\{x_{r_i}\}_{i=1}^{2n} \quad \text{and} \quad \{x_{l_i}\}_{i=1}^{2n} \tag{6.17}$$

such that

$$0 = x_{l_1} < x_{l_2} \leq x_{r_1} < x_{l_3} \leq x_{r_2} < \cdots < x_{l_{2n}} \leq x_{r_{2n-1}} < x_{r_{2n}} = 1. \tag{6.18}$$

Then define

$$\Omega_i = (x_{l_i}, x_{r_i}) \times (0,1), \quad \text{for} \quad i = 1, \cdots, 2n. \tag{6.19}$$

step ii:

Guess $u_i = \psi_i$ on $\Gamma_i - \Gamma$, for $i = 1, 3, ..., 2n-1$ (ie. guess boundary values on pseudo boundaries for odd regions).

step iii:

Initialize iteration counter k to 1.

For $i = 1, 3, ..., 2n-1$, solve (in parallel on all n processors)

$$\begin{aligned} \Delta u_i^{(k)} &= -f \quad \text{on} \quad \Omega_i \\ u_i^{(k)} &= \phi \quad \text{on} \quad \Gamma_i \cap \Gamma \\ u_i^{(k)} &= \psi_i \quad \text{on} \quad \Gamma_i \cap \Omega. \end{aligned} \tag{6.20}$$

Then, for $i = 2, 4, ..., 2n-2$, solve (in parallel on n-1 processors)

$$\begin{aligned} \Delta u_i^{(k)} &= -f \quad \text{on} \quad \Omega_i \\ u_i^{(k)} &= \phi \quad \text{on} \quad \Gamma_i \cap \Gamma \\ u_i^{(k)} &= u_{i-1}^{(k-1)} \quad \text{on} \quad \Gamma_i \cap \Omega_{i-1} \\ u_i^{(k)} &= u_{i+1}^{(k-1)} \quad \text{on} \quad \Gamma_i \cap \Omega_{i+1}, \end{aligned} \tag{6.21}$$

while also solving (on the n^{th} processor), for $i = 2n$

$$\begin{aligned} \Delta u_{2n}^{(k)} &= -f \quad \text{on} \quad \Omega_{2n} \\ u_{2n}^{(k)} &= \phi \quad \text{on} \quad \Gamma_{2n} \cap \Gamma \\ u_{2n}^{(k)} &= u_{2n-1}^{(k-1)} \quad \text{on} \quad \Gamma_{2n} \cap \Omega_{2n-1}. \end{aligned} \tag{6.22}$$

(Note: Pseudo boundaries for even regions do not require an initial guess.)

step iv:

Repeat until $\|u_i - u\| \leq \varepsilon$ on each Ω_i, $i = 1, 2, ..., 2n$, for some iteration k and for some error tolerance ε.

$k = k + 1$.

Solve

$$\begin{aligned} \Delta u_1^{(k)} &= -f \quad \text{on} \quad \Omega_1 \\ u_1^{(k)} &= \phi \quad \text{on} \quad \Gamma_1 \cap \Gamma \\ u_1^{(k)} &= u_2^{(k-1)} \quad \text{on} \quad \Gamma_1 \cap \Omega_2, \end{aligned} \tag{6.23}$$

on one processor, while also solving (on the remaining n-1 processors) for

$i = 3, 5, ..., 2n - 1$

$$\begin{aligned} \Delta u_i^{(k)} &= -f \quad \text{on} \quad \Omega_i \\ u_i^{(k)} &= \phi \quad \text{on} \quad \Gamma_i \cap \Gamma \\ u_i^{(k)} &= u_{i-1}^{(k-1)} \quad \text{on} \quad \Gamma_i \cap \Omega_{i-1} \\ u_i^{(k)} &= u_{i+1}^{(k-1)} \quad \text{on} \quad \Gamma_i \cap \Omega_{i+1}. \end{aligned} \tag{6.24}$$

Then solve, for $i = 2, 4, ..., 2n - 2$, (on n-1 processors)

$$\begin{aligned} \Delta u_i^{(k)} &= -f \quad \text{on} \quad \Omega_i \\ u_i^{(k)} &= \phi \quad \text{on} \quad \Gamma_i \cap \Gamma \end{aligned} \tag{6.25}$$

$$u_i^{(k)} = u_{i-1}^{(k)} \quad \text{on} \quad \Gamma_i \cap \Omega_{i-1}$$

$$u_i^{(k)} = u_{i+1}^{(k)} \quad \text{on} \quad \Gamma_i \cap \Omega_{i+1},$$

while solving (on the n^{th} processor)

$$\begin{aligned} \Delta u_{2n} &= -f \quad \text{on} \quad \Omega_{2n} \\ u_{2n}^{(k)} &= \phi \quad \text{on} \quad \Gamma_{2n} \cap \Gamma \\ u_{2n}^{(k)} &= u_{2n-1}^{(k)} \quad \text{on} \quad \Gamma_{2n} \cap \Omega_{2n-1}. \end{aligned} \tag{6.26}$$

step v:

Let $\tilde{u} = u_i^{(k)}$ on Ω_i for $i = 2, 4, ..., 2n$,

$\tilde{u} = u_1^{(k)}$ on $\Omega_1 - \Omega_2$,

and $\tilde{u} = u_i^{(k)}$ on $\Omega_i - (\Omega_{i-1} \cup \Omega_{i+1})$, $\square$

Then $\|\tilde{u} - u\| < \varepsilon$ so $\tilde{u}$ is a suitable approximation to u.

6.3 Numerical Schwarz Methods

For a numerical implementation of these Schwarz methods, the differential equation on each subdomain Ω_i for each iteration k becomes a finite difference or matrix equation (as presented in Chapter 3), where each subdomain becomes a subgrid. Thus for each subdomain Ω_i and iteration k we have a subgrid G_i and an associated

matrix equation

$$A_i \mathbf{u}_i^{(k)} = \mathbf{c}_i^{(k)}, \tag{6.27}$$

where A_i is the matrix associated with the i^{th} subdomain Ω_i, $\mathbf{c}_i^{(k)}$ is a vector incorporating the values of $\mathbf{f}_i$ from Ω_i, the boundary conditions from the original problem (6.1), and the pseudo-boundary conditions obtained from neighboring solutions, and $\mathbf{u}_i^{(k)}$ are solutions to equations (6.27).

Of course, numerically we do not solve for each $\mathbf{u}_i^{(k)}$ exactly but rather obtain an approximation. In the numerical algorithm we use we have incorporated the Gauss-Seidel method to obtain an approximation $\tilde{\mathbf{u}}_i^{(k)}$ for each Ω_i and iteration k of the parallel Schwarz process.

As an initial vector $\mathbf{v}_i^{(0)}$ for the Gauss-Seidel process on Ω_i at the k^{th} iteration of the Schwarz process, we use

$$\mathbf{v}_1^{(0)} = \begin{cases} \mathbf{u}_1^{(k-1)} & \text{on} \quad \Omega_i - \Omega_2 \\ \\ \mathbf{u}_2^{(k-1)} & \text{on} \quad \Omega_1 \bigcap \Omega_2, \end{cases}$$

$$\mathbf{v}_i^{(0)} = \begin{cases} \mathbf{u}_{i-1}^{(k-1)} & \text{on} \quad \Omega_i \bigcap \Omega_{i-1} \\ \\ \mathbf{u}_i^{(k-1)} & \text{on} \quad \Omega_i - (\Omega_{i-1} \bigcup \Omega_{i+1}) \\ \\ \mathbf{u}_{i+1}^{(k-1)} & \text{on} \quad \Omega_i \bigcap \Omega_{i+1}, \end{cases}$$

for $i = 3, 5, \ldots, 2n-1$,

$$\mathbf{v}_i^{(0)} = \begin{cases} \mathbf{u}_{i-1}^{(k)} & \text{on} \quad \Omega_i \bigcap \Omega_{i-1} \\ \mathbf{u}_i^{(k-1)} & \text{on} \quad \Omega_i - (\Omega_{i-1} \bigcup \Omega_{i+1}) \\ \mathbf{u}_{i+1}^{(k)} & \text{on} \quad \Omega_i \bigcap \Omega_{i+1}, \end{cases}$$

for $i = 2, 4, 6, \ldots, 2n-2$, and

$$\mathbf{v}_{2n}^{(0)} = \begin{cases} \mathbf{u}_{2n-1}^{(k)} & \text{on} \quad \Omega_{2n} \bigcap \Omega_{2n-1} \\ \mathbf{u}_{2n}^{(k-1)} & \text{on} \quad \Omega_{2n} - \Omega_{2n-1}. \end{cases}$$

Table 6.1 shows some actual computational results using the numerical version of algorithm 6.3. In this case we have used 16 subdomains on 8 processors. The number of iterations of Gauss-Siedel taken on each subdomain to approximate each respective subdomain problem is given by s and α is the number of columns of grid points in the overlapping region between two adjacent subdomains. Once again, as a stopping criteria we used $\| \mathbf{u}_h^{(k)} - \mathbf{u}_h \| \leq \varepsilon = 10^{-6}$. We experimented with different values for s and α. Our best results were obtained using $s = 1$ and $\alpha = 4$, indicating that for this version of the parallel Schwarz method run on a shared memory machine, the best results are obtained when a minimum number (ie. 1) of Gauss-Seidel iterations are performed on each subdomain problem and the overlap between adjacent subdomains is maximal.

In all cases we see a marked improvement over the basic Gauss-Seidel method. In particular, for $s = 1$ and $\alpha = 4$ we see the speed of execution, again relative to the Gauss-Seidel method, increase by a factor of $\frac{164.0}{22.2} = 7.4$.

Table 6.1: A Parallel Schwarz Method

	$s = 1$		$s = 3$		$s = 10$		$s = 20$	
	k	t	k	t	k	t	k	t
$\alpha = 0$	6250	24.2	2250	25.6	1000	37.7	850	64
$\alpha = 2$	4800	25.6	1700	26.8	550	28.8	350	36.6
$\alpha = 4$	3250	22.2	1100	22.2	350	23.5	200	26.8

Chapter 7

A Parallel Multigrid Method

7.1 The Algorithm

In chapters 5 and 6 we have presented two methods, one sequential the other parallel, that serve to enhance more basic iterative methods such as the Gauss-Seidel and Jacobi methods. In this chapter, we introduce a further improvement by combining the two-grid multigrid method and the parallel Schwarz method. The idea is to take our two-grid sequential multigrid method and use the parallel Schwarz method in place of the basic Gauss-Seidel method. The algorithm is as follows:

ALGORITHM 7.1

To solve

$$\Delta u = -f \quad \text{on} \quad \Omega = (0,1) \times (0,1) \tag{7.1}$$

$$u = \phi \quad \text{on} \quad \Gamma,$$

we begin by discretizing the problem thus obtaining the matrix equation

$$A_h \mathbf{u}_h = \mathbf{f}_h \quad \text{on} \quad G_h. \tag{7.2}$$

We then proceed as follows:

step i:

Initialize iteration counter k to 0.

Guess $\mathbf{u}_h^{(0)}$ as an approximation to $\mathbf{u}_h$.

step ii:

Repeat until $\|\mathbf{u}_h^k - \mathbf{u}_h\| \leq \varepsilon$, for some iteration k and some error tolerance ε.

— Perform three parallel Schwarz iterations on $A_h \mathbf{u}_h^{(k)} = \mathbf{f}_h$ (ie. this is algorithm 6.3 with $k = 3$). The method used on each subdomain is s iterations of the Gauss-Seidel method.

— Compute the residual on the fine grid G_h: $\mathbf{r}_h^{(k)} = \mathbf{f}_h - A_h \mathbf{u}_h^{(k)}$.

— Transfer the residual to the coarse grid G_{2h}: $\mathbf{r}_{2h}^{(k)} = I_h^{2h} \mathbf{r}_h^{(k)}$.

— Perform thirty parallel Schwarz iterations on the coarse grid to obtain a good

approximation to the solution to the coarse grid residual equation

$$A_{2h}\mathbf{e}_{2h} = \mathbf{r}_{2h}. \tag{7.3}$$

This yields $\tilde{\mathbf{e}}_{2h}$, a good approximation to $\mathbf{e}_{2h}$.

— Transfer error term to the fine grid G_h: $\tilde{\mathbf{e}}_h^{(k)} = I_{2h}^h \tilde{\mathbf{e}}_{2h}^{(k)}$.

— Compute new approximation: $\mathbf{u}_h^{(k+1)} = \mathbf{u}_h^{(k)} + \tilde{\mathbf{e}}_h^{(k)}$.

— Increment the iteration counter k.

step iii:

Perform three final parallel Schwarz iterations on $A_h\mathbf{u}_h^{(k)} = \mathbf{f}_h$, as was done in step ii. This reduces high frequency error introduced during the final interpolation. □

Table 7.1 shows some computational results using this algorithm. As with the parallel Schwarz method of the previous chapter, the number of iterations of Gauss-Siedel taken on each subdomain to approximate each respective subdomain problem is given by s and α is the number of columns of grid points in the overlapping region between two adjacent subdomains. Also, as before, as a stopping criteria we used $\| \mathbf{u}_h^{(k)} - \mathbf{u}_h \| \leq \varepsilon = 10^{-6}$.

Table 7.1: A Parallel Multigrid Method

	$s = 1$		$s = 3$		$s = 10$		$s = 20$	
	k	t	k	t	k	t	k	t
$\alpha = 0$	237	12.5	102	14.6	68	31.6	64	58.9
$\alpha = 2$	189	12.8	82	15.6	50	30.9	41	50.4
$\alpha = 4$	112	9.29	54	12.7	34	26.3	26	40.1

Chapter 8

Conclusion

In performing the research for this paper I explored two significantly different iterative algorithms, one essentially sequential and the other essentially parallel, for solving elliptic partial differential equations. Both algorithms serve to significantly enhance a more basic traditional method, in this case the Gauss-Seidel method.

The sequential multigrid method uses one or more coarser grids to accelerate the convergence of a basic method such as Gauss-Seidel. The multigrid approach is to address the problem, associated with the basic method, that low frequency error components are reduced at a much slower rate than those of higher frequency. By transferring these low frequency components to a coarser grid their frequencies, relative to the coarser grid, are increased. Then the basic method can be employed to more rapidly reduce this part of the error. The use of coarser grids is also helpful

because the data representing the problem is significantly reduced on the coarser grids making iterations of the basic method that much faster.

The parallel Schwarz method takes advantage of the fact that, in methods such as Gauss-Seidel, the update of any grid point of G_h is independent of all but its immediate neighboring grid points. Thus, using a machine such as the Alliant FX/8, the workload can be judiciously distributed over multiple processors, realizing a significant increase in the speed of execution.

I combined these two methods into the parallel multigrid method presented in Algorithm 7.1. This algorithm capitalizes on both of the previous approaches. The use of a coarser grid is used to improve the rate of convergence while multiple processors are used to distribute the workload. Whereas, our best execution speeds, relative to the speed of the Gauss-Seidel method, were $\frac{164.0}{41.0} = 4.0$ and $\frac{164.0}{22.2} = 7.4$ for the sequential multigrid and parallel Schwarz methods, respectively, the best results using the combined method showed a relative speed of $\frac{164.0}{9.29} = 17.6$. As in the pure parallel Schwarz method, the best results were obtained using $s = 1$ and $\alpha = 4$. Table 8.1 summarizes these results.

One may view the parallel multigrid method as a parallel enhancement of the essentially sequential multigrid method or as a multigrid enhancement of a parallel Schwarz method. It is not unusual to find a sequential method, which contains potentially parallel sections of code, enhanced by placing it in a parallel environment.

What is unusual though is to find such a highly parallel method such as the Schwarz method enhanced by placing it in an inherently sequential environment.

Table 5.1: Relative Speeds of Execution

method	time	factor
Gauss-Seidel	164.0	1.0
sequential multigrid	41.0	4.0
parallel Schwarz	22.2	7.4
parallel multigrid	9.29	17.6

Bibliography

[1] T. Meis and U. Marcowitz, *Numerical Solution of Partial Differential Equations,* Springer-Verlag, New York, 1981.

[2] R. Varga, *Matrix Iterative Analysis,* Prentice-Hall,Inc., New Jersey, 1962.

[3] D. Young, *Iterative Solutions of Large Linear Systems,* Academic Press,Inc., New York, 1971.

[4] Alliant Computer Systems Corporation, *FX/Series Product Summary,* 1971.

[5] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis,* Springer-Verlag, New York, 1980.

[6] A. Oppenheim and R. Schafer, *Digital Signal Processing,* Prentice Hall, New Jersey, 1975.

[7] R. P. Fedorenko, *A relaxation method for solving elliptic difference equations,* U.S.S.R. Comp. Math. and Math. Physics, 1 no. 5, 1962.

[8] R. P. Fedorenko, *The speed of convergence of an iterative process,* U.S.S.R. Comp. Math. and Math. Physics, 4 no. 3, 1964.

[9] N. S. Bakhvalow, *On the convergence of a relaxation method with natural constraints on the elliptic operator,* U.S.S.R. Comp. Math. and Math. Physics, 6 no. 5, 1966.

[10] A. Brandt, *Multi-level adaptive solutions to boundary-value problems,* Mathematics of Computation, Volume 31, No. 138, April 1977.

[11] K. Stuben and U. Trottenberg, *Multigrid methods: fundamental algorithms, model problem analysis and applications,* Multigrid Methods, Springer-Verlag, Berlin, 1982.

[12] W. Briggs, *A Multigrid Tutorial,* SIAM, 1987.

[13] H. A. Schwarz, *Uber einen Grenzubergang durch alternirendes Verfahren,* Ges. Math. Abhandlungen, Bd. 1,Berlin, 1890.

[14] E. Picard, *Memoir sur la theorie des equations sux derivees partielles et la methode des approximations successives,* J. Math. Pures et Appl., ser. 4,6, 1980.

[15] L. V. Kantorovich and V. I. Krylov, *Approximate Methods of Higher Analysis,* P. Noordhoff, Ltd., Groningen, Nederlands, 1958.

[16] K. Miller, *Numerical analogs to the Schwarz alternating procedure,* Numerische Mathematik 7, 1965.

[17] Kang Lishan, *The Schwarz algorithm,* Wuhan University Journal, 1981.

[18] G. Rodrigue and J. Simon, *A generalization of the numerical Schwarz algorithm,* Proceedings of the 6th International Conference of Comp. Math. in Appl. Sci. and Eng., INRIA, Dec. 1983.

[19] G. Rodrigue, *Inner/outer iterative methods and numerical Schwarz algorithms,* Parallel Computing 2, North Holland, 1985.