Parallel Algorithms and Architectures for Solving Elliptic Partial Differential Equations

by

Chung-Chieh Kuo

B.S.E.E., National Taiwan University

(1980)

SUBMITTED TO THE
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February, 1985

Signature of Author: ______________________
Department of Electrical Engineering and Computer Science
January 30, 1985

Certified by: ______________________
Associate Prof. Bernard C. Levy, Thesis Supervisor

Certified by: ______________________
Assistant Prof. Bruce R. Musicus, Thesis Supervisor

Accepted by: ______________________
Prof. Arthur C. Smith,
Chairman, Departmental Committee on Graduate Students

Parallel Algorithms and Architectures for Solving Elliptic Partial Differential Equations

by

Chung-Chieh Kuo

Submitted to the Department of Electrical Engineering and Computer Science on Junuary 30, 1985 in partial fulfillment of the requirements of the degree of Master of Science in Electrical Engineering

ABSTRACT

Two parallel numerical iterative algorithms for solving linear elliptic partial differential equations (PDEs) suitable for VLSI implementation are proposed. They are the locally accelerated successive over-relaxation (LASOR) and concurrent multigrid (CMG) methods. The supporting architectures for these parallel algorithms are also discussed.

This thesis first examines the implementation of traditional numerical PDE algorithms with mesh-connected processor arrays. The main difficulty of these implementations is that the determination of acceleration factors requires global communication at each iteration. The high communication cost increases the computation time per iteration significantly. Therefore, a new algorithm - the locally accelerated successive over-relaxation (LASOR) algorithm - is proposed to achieve the acceleration effect with very little global communication.

The LASOR scheme requires the broadcasting of one or two elements of global information at the loading stage, then it uses only nearest-neighborhood communication at each iteration. The convergence rate of the LASOR method is shown analytically to be the same as the SOR method in solving the constant-coefficient PDEs. However, computer simulation indicates that the performance of the LASOR method is superior to that of the SOR method for PDEs with space-varying coefficients.

Another algorithm, the CMG method, combines the LASOR algorithm, the multigrid concept, and two dimensional filter design to achieve highly parallel computation. Its supporting architecture has either a pyramidal structure or a structure based on multiple mesh-connected processors.

Thesis Supervisor : Bernard C. Levy

Title : Associate Professor of Electrical Engineering and Computer Science

Thesis Supervisor : Bruce R. Musicus

Title : Assistant Professor of Electrical Engineering and Computer Science

Dedicated to

My Grandmother and Parents

Acknowledgements

I would like to thank Professor Bernard Levy, my advisor, for introducing me to this interesting topic, spending a large amount of time on improving my research skills, and giving much helpful advice throughout my work. I also wish to thank Professor Bruce Musicus, my other advisor, for his unfailing enthusiasm and insight and for being a great teacher as well as a good friend. Their friendship and willingness to allow me to pursue my ideas helped to make my master program enjoyable.

Thanks also go to many of my friends in the Laboratory of Information and Decision Systems (LIDS) and in the Digital Signal Processing Group (DSPG) of the Research Laboratory of Electronics (RLE) for their helpful discussions and friendship. In particular, I want to thank Wei-Kang Kevin Tsai, my good friend and officemate, for his encouragement and help in my work and life.

Finally, no amount of gratitude can adequately repay my family for their continuous support and sacrifice during the past years. To them I am truly indebted.

This work has been supported in part by the Army Research Office under Grant No. DAAG29-84-K-0005, and in part by the Advanced Research Projects Agency monitored by ONR under Contract N00014-81-K-0742, and by the AFOSR Contract F49620-84-C-0004.

Table of Contents

Table of Figures

Chapter 1. Introduction

1.1 Problem statement

Many physical systems can be described by partial differential equations (PDEs). It is usually very difficult to get *analytic* solutions of these PDEs because of the irregular geometry of the problem domain and of other reasons. Therefore, the *numerical* solution of PDEs plays an important role in understanding and simulating these systems. Since the late 40's, the introduction of high speed computing machines has made the field of numerical solutions of PDEs grow very rapidly [1]. The progress in computer technology and numerical algorithms has helped us to solve many complex PDEs; however, there are still many PDEs that cannot be adequately solved using today's most powerful computers. The examples include computational aerodynamics and hydrodynamics, weather forecasting, plasma simulation, structure analysis, and turbulence modeling. How to solve them efficiently is one of the most challenging goals for the next generation of computers.

In order to get high throughput performance, the algorithms and architectures of tomorrow's computer must emphasize parallelism. In fact, parallel computation has been studied intensively for a long time. Several parallel computing machines, such as Cray-1, Cyber 205, Illiac IV, and C.mmp, have been built to speed up large scale computation problems.

Although some kinds of parallelism are adopted in all these machines, the design principles are not the same. This is due to the fact that parallelism can be introduced at several levels and in many different ways. Roughly speaking, we may categorize the parallelism into two levels - circuit level, such as parallel adders and multipliers, and functional level, such as multiprocessor arrays. The functional level parallelism can be further classified into several subclasses. For examples, Cray-1 and Cyber 205 belong to the SIMD (single-instruction-multiple-data) vector pipelining class, Illiac IV falls into the SIMD array class, and C.mmp is an example of MIMD (multiple-instruction-multiple-data) machine. More examples can be found in [2].

To use these parallel machines successfully, good parallel algorithms are needed; therefore, some parallel numerical algorithms have been developed for this purpose. Heller [3] contains a good review of these studies.

A majority of parallel numerical PDE algorithms are direct methods which use the fact that LU decomposition (Gauss elimination method) or QR decomposition (Given's transformation method) of a tridiagonal matrix can be done fast. So, instead of solving a tridiagonal matrix equation, we solve two triangular linear systems, which can be done quickly. Therefore, if a system $A\ x\ =\ b$ can be decomposed into many independent smaller-dimensional tridiagonal subsystems, the solution of all these subsystems can be done in parallel and very efficiently [4].

These direct methods are powerful for some special class of problems; however, they cannot be applied in more general cases such as for problems with irregular domain, or problems in three dimensions, where the system decomposition becomes very difficult. In contrast, the iterative methods are suitable for general problems.

Besides, they also allow high degrees of parallelism. In this thesis, I will concentrate on parallel iterative algorithms.

The hardware implementation of the conventional parallel machines uses SSI (small scale integration) and MSI (median scale integration) technology. However, IC technology has progressed so quickly that VLSI (very large scale integration) chips will become the main building blocks of tomorrow's parallel computers. With VLSI technology, we may put millions of transistors on a single chip, which provides new possibilities as well as challenges, and affects the design of future computers in the following ways.

It is feasible to fabricate many powerful special purpose chips and to use them as peripheral devices to a host computer. As a consequence, one interesting phenomenon is the merging of algorithms and architectures, the ability to implement different algorithms on different hardware designs customized to support the algorithms in the most efficient way. The close relationship between applications, algorithms, and architectures is really one of the most important features of computers implemented with VLSI technology.

At the same time, VLSI introduces new difficulties, which were not so critical in SSI or MS : high system complexity, limited I/O pins, and high communication cost within a chip. In order to manage system complexity, regularity and modularity are two important design principles. The number of I/O pins of a VLSI chip is quite small compared to the number of components within it (currently $O(10^2)$ v.s.

$O(10^6)$ and probably $O(10^3)$ v.s. $O(10^7)$ in the future). Therefore, if computation requires a lot of data transfer among chips or between chips and the host, the I/O problems may become a bottleneck, and will slow down the processing speed. Parallel computation often introduces communication problems among different processors. Communication cost, which is seldom mentioned for a single processor system, is no longer negligible in a multi-processor environment. The problem becomes crucial when VLSI technology is used to implement the hardware, because long range communications not only cost a lot of area and energy but also result in time delays in a VLSI chip. A good algorithm for VLSI systems should be computationally as well as communicationally economical.

The first algorithms which took advantage of VLSI technology as well as consider its constraints were the systolic algorithms proposed by Kung and Leiserson in 1978 [5]. Since then, many algorithms and special purpose supporting architectures have been studied [6]. All these designs are based on the principles of regularity, modularity, and local communication. However, I/O problems, which were not considered carefully in all two dimensional systolic array algorithm and architecture designs, turn out to be a major hindrance to making these designs work at present.

The objective of the research described in this thesis is to combine VLSI technology with numerical analysis algorithms for PDEs in order to obtain special purpose PDE solvers. This combination promises to achieve better performance with lower cost, compared with the traditional parallel computing machines without VLSI.

However, the implementation of traditional numerical algorithms in a VLSI environment encounters serious communication problems of such a magnitude that the acceleration effect is cancelled out by communication cost. In order to solve the communication problem, this thesis proposes two parallel iterative algorithms that can achieve the acceleration effect using only local communication at each iteration.

Both algorithms can be run on a square mesh-connected processor array. This kind of multiprocessor architecture has many advantages. It satisfies the regularity and modularity requirements of VLSI. If we discretize the problem domain of PDE with a square grid, there is a natural correspondence between processors and grid points. Therefore, some relaxation schemes can be performed using only nearest-neighborhood information. In addition, the I/O bottleneck is removed, since communication between fast PDE solvers and the host only occurs in the loading and unloading stages.

PDEs can be broadly classified into two types - the more-or-less hyperbolic type and the more-or-less elliptic type to which the parabolic type is closely related. It has been stated that if an architecture could deal with these two types, it would have almost universal applications [7]. The algorithms of this thesis are mainly directed to linear elliptic type problems. The linear elliptic PDE can usually be discretized into a matrix equation $A\ x = b$, where A is a symmetric and positive definite matrix. (See [15], p. 189). Instead of starting from the differential equations and discussing different discretization schemes, we will focus on the discretized equation and consider how to solve this matrix equation using parallel algorithms and computer architectures.

1.2 Outline

This thesis is organized as follows.

Chapter 2 surveys the traditional iterative methods for solving PDEs and discusses the difficulties of implementing them in a multiprocessor VLSI chip. The central issue is that to obtain the acceleration effect, by which faster convergence rate can be achieved, requires global communication. However, global communication increases computation time per iteration enormously in a multiprocessor environment such that the acceleration effect is cancelled out by communication cost.

Therefore, a new approach known as locally accelerated successive relaxation (LASOR) method is presented in Chapter 3. This method, which is applicable to both linear constant-coefficient PDEs and linear varying-coefficient PDEs, increases the convergence rate with very low communication cost. This algorithm is analyzed by local Fourier analysis and the optimal local relaxation factors are determined by the coefficients of local operators.

The communication complexity, convergence property, and convergence rate analysis and simulation of the LASOR algorithm are shown in Chapter 4. The convergence rate of LASOR is the same as that of SOR in solving linear constant-coefficient PDEs analytically. The computer simulation results indicate that the convergence rate of LASOR is superior to that of SOR in solving linear varying coefficient PDEs.

Chapter 5 proposes another parallel algorithm - concurrent multigrid method (CMG). This method combines the concepts of multigrid discretization, 2-D filtering and local acceleration. Supporting architectures, pyramidal multi-processor arrays and mesh-connected processor arrays, are also discussed.

Some further extensions and conclusions are mentioned in Chapter 6.

Chapter 2. Implementing Traditional Numerical PDE Algorithms with Mesh-Connected Processor Arrays

2.1 Introduction

In this chapter, we will focus on the issue of implementing different traditional numerical PDE algorithms in one specific type parallel computer architecture: mesh-connected processor arrays. The mesh-connected processor array has many advantages. First, The interconnection between processors is simple and regular. Secondly, it has 2-D planer structure which is appropriate for VLSI implementation. Thirdly, there is a good correspondence between the discretized 2-D space points and the processors in this array so that we may take advantage of this structure to obtain parallel computation.

The conventional way to solve PDEs numerically can be divided into two stages: forming a system of equations by some discretization scheme, and solving the system of equations.

Two commonly used discretization approaches are the finite-difference and finite-element methods [8]. The finite-difference methods always give mesh-connected rectangular grids. We may assign each node to one processor, so that there exists a natural mapping between grid points and computational elements. However, this fact does not apply to finite-element methods directly, since both triangular and quadrilateral elements are allowed and each node may have an arbitrary number of connections with other nodes. This flexibility makes the finite-element method more powerful in fitting general boundary shapes, but also introduces more irregularities. Therefore, the mapping from the nodes to the processors is not trivial. In order to make the mapping simpler, we should choose a subclass of general finite-element schemes. One possible choice is to assume that all elements are quadrilateral and that each inner node has exactly four connections with other nodes. For simplicity, we will only consider the finite-difference discretization scheme in this thesis. Most of the results presented can be generalized to the subclass of finite-element methods mentioned above.

Discretizing a linear PDE problem, we get a sparse matrix equation,

$$A\ x = b\ , \tag{2.1}$$

where A is an $N \times N$ sparse matrix, and x and b are two N dimensional vectors. Then, the remaining question is how to solve this matrix equation.

The solution of the linear system $A\ x = b$ has received a large amount of attention over the years. Generally speaking, it can be solved by either direct or iterative methods. In the following, we will survey these methods briefly and, more importantly, discuss their implementation. This serves two purposes. First, we want to show how the parallel processing concept can be applied at the node level, not at the subsystem level such as for the decomposed tridiagonal matrix equations mentioned in Chapter 1. Secondly, we will point out the shortcomings of these implementations.

2.2 Direct methods

The direct methods include the Gaussian elimination, Givens transformation, Householder transformation, Gram-Schmidt transformation methods, and a variety of other orthogonalization procedures [9]. If the matrix A has some special properties such as symmetry or a Toeplitz structure, then more efficient algorithms are available such as the Cholesky decomposition, or the Levinson, and fast Cholesky recursions [10].

Many researchers have proposed VLSI array processor architectures to implement these algorithms. Kung and Leiserson [5] used a hexagonal systolic array to factor the matrix A into lower and upper triangular matrices L and U, and then solved two triangular linear systems , $L\ y = b$ and $U\ x = y$, with a linear systolic array. Kung [11] suggested another structure called a wavefront array processor which can implement the Gaussian elimination method with pivoting as well as without pivoting. The cordic array processors proposed by Ahmed, Delosme, and Morf [12] made use of Givens' algorithm which seems more efficient than the Gaussian elimination method with pivoting in a parallel computing environment. If A is a Toeplitz matrix, Kung and Hu [13] designed a fast Toeplitz solver which can solve the equation quickly. Similar results can also be found in [12]. In terms of computational cost, for a general $N \times N$ matrix A, we need $O(N^2)$ processors to solve the equations in $O(N)$ time. In contrast, for an $N \times N$ Toeplitz matrix, only $O(N)$ processors are required.

One difficulty with using the above mentioned systems to solve the sparse matrix equations arising from PDE problems is that too many processors are needed. For a general PDE problem, A is not a symmetric matrix, because of the discretization of the irregular shaped boundary. It is indeed very hard to find any special structure in the matrix A for an arbitrary given problem. Therefore, in order to handle the general situation, we have to view A as an ordinary (though sparse) N by N matrix. For a practical real world problem, N is a very large number, say, $O(10^4)$. The systolic schemes would require $O(10^8)$ processors, which seems unacceptable even if modern VLSI technology is used. Schemes for using small systolic arrays to solve large linear equation problems have been proposed, but they are quite complicated. These schemes also involve unnecessary waste since most processors are idle during the processing, due to the sparsity of the matrix A.

Another difficulty is due to the fact that $A\ x = b$ is not necessarily a good approximation of the original PDE, so that further grid size refinements may be needed. If direct methods are used, then the solutions of previous runs are totally useless for the current run. Iterative algorithms, on the other hand, are able to use old solutions from an old discretization efficiently to solve linear equations for the new discretization.

2.3 Iterative methods

Iterative solutions of the linear system $A\ x = b$ rely on two different principles - relaxation and minimization.

By appropriately decomposing the matrix A and rearranging, we may form various iterative relations, which are known as *relaxation* methods [14]. For example, we can express A as the matrix sum

$$A = D - E - F\ , \tag{2.2}$$

where D is a diagonal matrix, and E and F are respectively strictly lower and upper triangular matrices. Rearranging equation (2.1), we get two different forms equivalent to the original equation,

$$x = D^{-1}(E + F)x + D^{-1}b \ , \tag{2.3}$$

$$x = (D - E)^{-1}F\,x + (D - E)^{-1}b \ , \tag{2.4}$$

therefore, we can define the following two iterative equations,

$$x^{(n+1)} = D^{-1}(E + F)\,x^{(n)} + D^{-1}b \quad n \geq 0 \ , \tag{2.5}$$

$$x^{(n+1)} = (D - E)^{-1}F\,x^{(n)} + (D - E)^{-1}b \quad n \geq 0 \ . \tag{2.6}$$

Equations (2.5) and (2.6) are called the Jacobi and Gauss-Seidel relaxations respectively, and $D^{-1}(E+F)$ and $(D-E)^{-1}F$ are the corresponding relaxation matrices. Other examples include successive overrelaxation (SOR), Chebyshev semi-iteration (CSI), and alternating-direction implicit iteration (ADI).

The design and analysis of these relaxation schemes heavily depends on the eigen-structure of the relaxation matrix. If the spectral radius of the relaxation matrix is less than 1, then the algorithm converges. In addition, the smaller the spectral radius is, the faster the convergence rate is. In order to improve the convergence rate, we may introduce some acceleration parameters to reduce the spectral radius. Theoretically, these optimal parameters can be determined by a formula which uses the knowledge of the eigen-structure of the relaxation matrix. Unfortunately, in practice, this information is not available. Therefore, some adaptive parameter estimation schemes have been derived to overcome these difficulties [15].

We may also view $A\ x = b$ as the result of a minimization problem. If we assume that the matrix A is symmetric positive definite (SPD), to solve $A\ x = b$ is equivalent to minimizing the following quadratic form

$$F(x) = \tfrac{1}{2}x^T Ax - b^T x \quad . \tag{2.7}$$

This can be done by the steepest descent or conjugate direction (CD) method, of which the conjugate gradient (CG) method is a special case [15]. For an ill-conditioned matrix A, the steepest descent method may be very slow, but for the CD or CG methods, we need at most N steps to find the solution in the absence of rounding errors. In this sense, the CD or CG methods act like direct methods. With the rounding errors, it is still possible to form some iterative relation and, thus, we get another important class of iterative algorithms.

The implementation of these algorithms on a sequential machine involves only coding. The analysis described above helps us to understand the performance of algorithms implemented in this way. However, the same analysis does not contain enough information for us to evaluate the performance of these algorithms when they are implemented on a VLSI multiprocessor chip, because it ignores an important factor - communication among processors.

Computation time for an iterative algorithm equals the product of the number of iterations and time per iteration. In a sequential machine, time per iteration is determined by operation counts, especially, by the number of floating point operations required. If this quantity is almost constant, then the total computational time can be thought to be proportional to the number of iterations only. However, time per iteration may change a lot for different algorithms in a multiprocessor environment. Consider a mesh-connected processor arrays. Algorithms using local communication take $O(1)$ communication time, while those using global communication require $O(\sqrt{N})$ communication time per iteration ($O(\sqrt{N})$ describes the number of processors across the array). Given this observation, we have to consider both factors together, i.e. we seek algorithms with larger convergence rate and shorter time per iteration for a given computer architecture.

In order to clarify the following discussion, we will focus on the Poisson equation with Dirichlet boundary conditions defined on the closed unit square. We will call this the *model problem*. Mathematically, we solve for $u(x_1, x_2)$ satisfying

$$\frac{\partial^2 u(x_1, x_2)}{\partial x_1^2} + \frac{\partial^2 u(x_1, x_2)}{\partial x_2^2} = f(x_1, x_2) \quad 0 < x_1, x_2 < 1. \quad (2.8.a)$$

and

$$u(x_1, x_2) = g(x_1, x_2) \qquad (x_1, x_2) \in \Gamma \ . \qquad (2.8.b)$$

where Γ denotes the set of points on the boundary of the square.

For a uniform square grid with spacing h, by using a finite-difference discretization, we obtain

$$4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} = -h^2 f_{i,j} \qquad (2.9)$$

for each inner node, which is called the *local equation.* The local equation describes the local behavior of the physical system that we are trying to solve. In terms of implementation, the local equation is more informative than the matrix equation. For analysis purposes, researchers are accustomed to the matrix formulation, but it will be shown in Chapters 3 and 5 that analysis from a local point of view is possible.

Starting from equation (2.9), we can discuss the details of implementing different iterative algorithms with a mesh-connected processor array.

(1) basic relaxation methods : the Jacobi method and the Gauss-Seidel method

The Jacobi method assumes the following iterative relation

$$u_{i,j}^{(n+1)} = \frac{1}{4}\left(u_{i-1,j}^{(n)} + u_{i+1,j}^{(n)} + u_{i,j-1}^{(n)} + u_{i,j+1}^{(n)} - h^2 f_{i,j} \right) \quad n \geq 0 \tag{2.10}$$

Let $M + 1 = \frac{1}{h}$, then there are $N = M^2$ inner grid points. Assuming that we use an $M \times M$ mesh-connected square processor array, we have a one to one mapping between inner grid points and processors. According to equation (2.10), in the Jacobi relaxation each processor takes the values of its nearest neighbors at the previous iteration to update its value at the current iteration. The time for each iteration is constant, because both communication time and computation time are constant.

If we call the grid point (i , j) a red point, when $i + j$ is even, and a black point, when $i + j$ is odd, the Jacobi method can be viewed in space and time as consisting of two interleaved, and totally independent, computational waves alternating between red and black points. Figure 2.1 illustrates these waves for the one dimensional case. In fact, these two waves result in unnecessary redundancy. We need only one wave to get the answer, since both waves converge to the same final values. If we delete one computation wave, the utilization of the processors becomes one half, i.e., every processor works only half of the time. Therefore, we may group one red point and one black point together and assign them to a single processor. This saves half of the hardware cost without loss of computational efficiency.

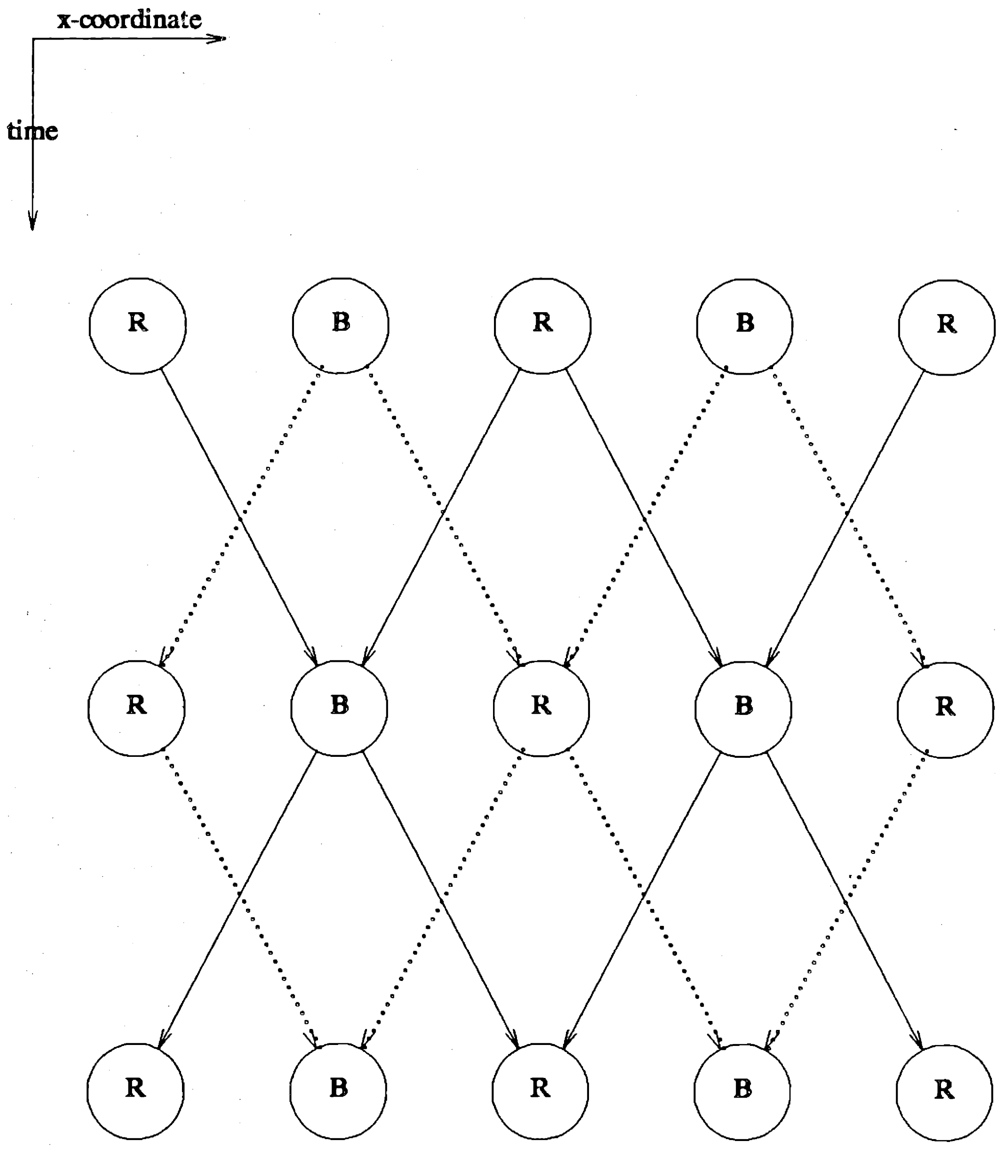

Fig 2.1 1-D Gauss-Seidel Relaxation with Red/Black Partitioning

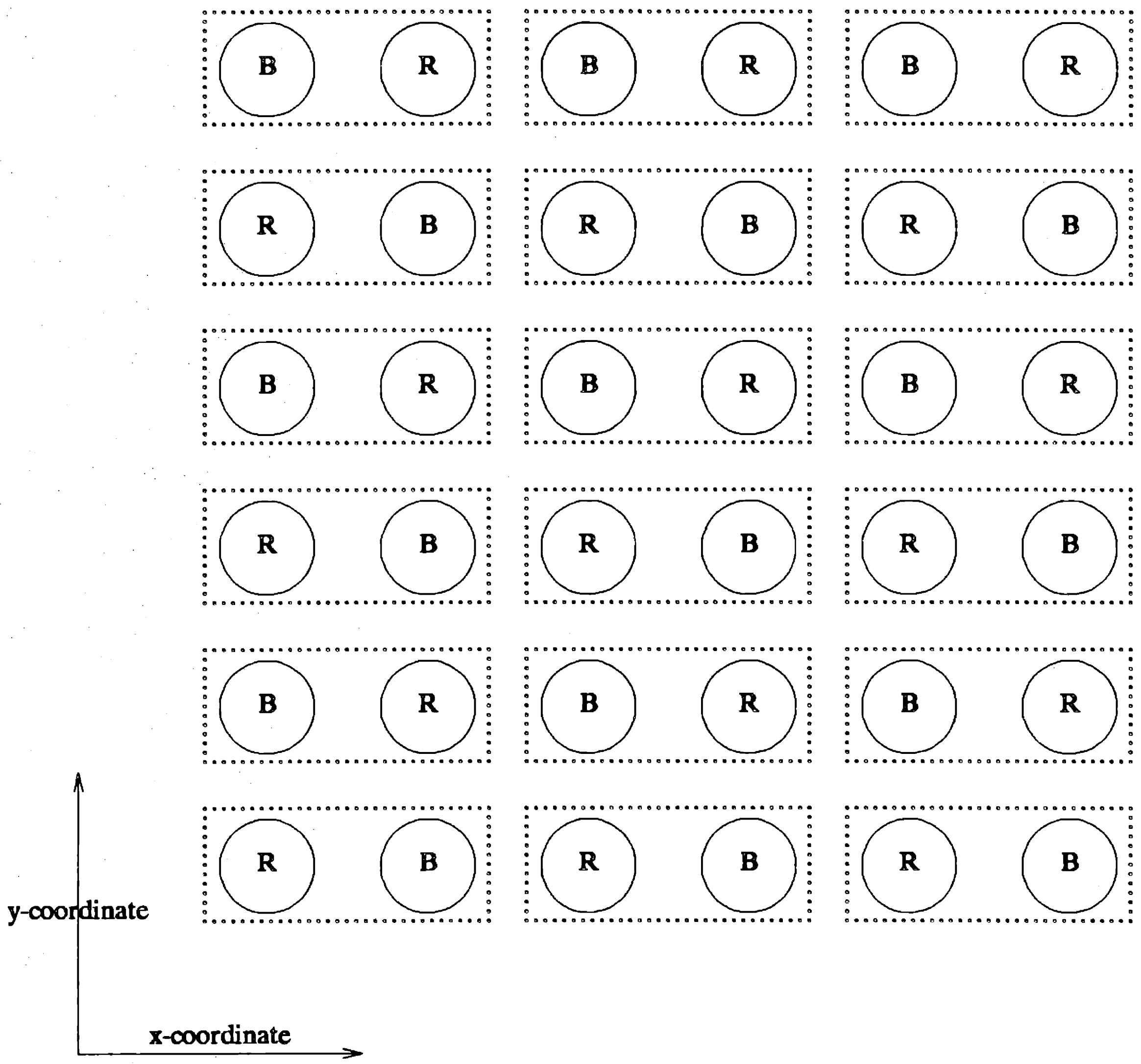

Fig 2.2 2-D Red/Black Partitioning and Grouping

We present one possible grouping scheme in Figure 2.2. It turns out that Figure 2.2 is the implementation of a popular Gauss-Seidel relaxation scheme - red/black point partitioning. We can write the local equation as follows

red points ($i + j$ is even) :

$$u_{i,j}^{(n+1)} = \frac{1}{4} \left(u_{i-1,j}^{(n)} + u_{i+1,j}^{(n)} + u_{i,j-1}^{(n)} + u_{i,j+1}^{(n)} - h^2 f_{i,j} \right) \quad n \geq 0 \quad (2.11.a)$$

black points ($i + j$ is odd) :

$$u_{i,j}^{(n+1)} = \frac{1}{4} \left(u_{i-1,j}^{(n+1)} + u_{i+1,j}^{(n+1)} + u_{i,j-1}^{(n+1)} + u_{i,j+1}^{(n+1)} - h^2 f_{i,j} \right) \quad n \geq 0 \quad (2.11.b)$$

There are many ways to choose Gauss-Seidel relaxation schemes; however, the red/black partitioning is usually preferred, because of its efficiency and simplicity. Note that the difference between the Jacobi and the Gauss-Seidel relaxation methods is that the Jacobi method updates the values of all nodes at one iteration while the Gauss-Seidel method updates the values of one half of these nodes at one iteration and updates the other half at the following iteration, based on the previously updated information. We still can give a simple explanation of why Gauss-Seidel relaxation is superior to Jacobi relaxation without any spectral analysis.

The chief shortcoming of the Jacobi or Gauss-Seidel iterative methods lies in their slow convergence rate. It usually happens that the spectral radius of the relaxation matrix is very close to 1, which causes the convergence rate to be extremely slow. The number of iterations needed is proportional to $O(\frac{1}{h^2})$ in this example.

(2) accelerated relaxation methods : CSI acceleration and SOR acceleration

Different acceleration methods may be combined with different basic relaxation methods resulting in numerous possible algorithms. We choose two typical examples to illustrate the general situations - the Chebyshev semi-iterative (CSI) method applied to Jacobi relaxation and the successive over-relaxation (SOR) method applied to Gauss-Seidel relaxation with red/black partitioning.

The acceleration schemes use carefully chosen relaxation parameters to reduce the spectral radii of the iterative matrices so that the iterative algorithms converge faster. The relaxation parameter for the SOR algorithm, denoted by ω, is introduced to accelerate the two-step Gauss-Seidel relaxation while the relaxation parameter for the CSI algorithm, denoted by ω_n, is introduced to minimize the spectral radius of a matrix polynomial. The details of these two algorithms are discussed in [14].

In general, CSI acceleration requires a knowledge of the largest and smallest eigenvalues of the basic relaxation matrix. However, if these two quantities are approximately equal in the absolute value, which is the case in the model problem, there exists a simpler version requiring the knowledge of spectral radius only [14].

Let $T_n(x)$ denote the Chebyshev polynomial with parameter n, i.e.,

$$T_n(x) = \begin{cases} cos(n \cos^{-1} x) & -1 \le x \le 1 \ , \ n \ge 0 \\ cosh(n \cosh^{-1} x) & x \ge 1 \ , \ n \ge 0 \end{cases} \tag{2.12}$$

and let ρ be the spectral radius of Jacobi relaxation matrix. Then the local equation for the Jacobi method with CSI acceleration can be written as

$$u_{i,j}^{(n+1)} = (1 - \omega_{n+1}) u_{i,j}^{(n-1)} \tag{2.13.a}$$

$$+ \frac{\omega_{n+1}}{4} (u_{i-1,j}^{(n)} + u_{i+1,j}^{(n)} + u_{i,j-1}^{(n)} + u_{i,j+1}^{(n)} - h^2 f_{i,j}) \quad n \ge 1 \ .$$

where

$$\omega_{n+1} = 1 + \frac{T_{n-1}(\frac{1}{\rho})}{T_{n+1}(\frac{1}{\rho})} \quad n \ge 1 \ , \quad \omega_1 = 1 \ . \tag{2.13.b}$$

On the other hand, the local equation for the SOR accelerated Gauss-Seidel method with red/black partitioning is

red points ($i + j$ is even) :

$$u_{i,j}^{(n+1)} = (1 - \omega) u_{i,j}^{(n)} \tag{2.14.a}$$

$$+ \frac{\omega}{4} (u_{i-1,j}^{(n)} + u_{i+1,j}^{(n)} + u_{i,j-1}^{(n)} + u_{i,j+1}^{(n)} - h^2 f_{i,j}) \quad n \ge 0 \ .$$

black points ($i + j$ is odd) :

$$u_{i,j}^{(n+1)} = (1 - \omega) u_{i,j}^{(n)} \tag{2.14.b}$$

$$+ \frac{\omega}{4} (u_{i-1,j}^{(n+1)} + u_{i+1,j}^{(n+1)} + u_{i,j-1}^{(n+1)} + u_{i,j+1}^{(n+1)} - h^2 f_{i,j}) \quad n \ge 0 \ .$$

where

$$\omega = \frac{2}{1 + \sqrt{1 - \rho^2}} \tag{2.14.c}$$

and ρ is still the spectral radius of the Jacobi relaxation matrix.

For a given mesh-connected processor array, if we know ρ a priori and broadcast it to all processors in the loading stage, then each processor can compute the acceleration parameters ω_{n+1} (CSI) or ω (SOR) on its own without additional communication cost. In this case, although the accelerated schemes (2.13) and (2.14), require more computation and memory than the basic schemes, (2.10) and (2.11), they present some significant advantages. The reason is that the number of iterations needed is reduced tremendously, becoming $O\left(\frac{1}{h}\right)$ for both acceleration schemes. However, we do not know ρ in advance in most cases and have to estimate it by some adaptive procedure. To our knowledge, all the estimation procedures developed require the knowledge of the norms of some global vectors. Therefore, global communication cannot be avoided. This means that communication cost for a single iteration becomes $O(M)$ or, equivalently, $O\left(\frac{1}{h}\right)$! The total running time is proportional to $O\left(\frac{1}{h^2}\right)$ again.

Comparing this result with that of the basic relaxation methods, it seems that we do not benefit from acceleration schemes in parallel implementation of iterative algorithms. This can be easily explained by noting the fact that in a single processor, there is no distinction between local and global communications, since all data are fetched from the same memory, while for a multiprocessor, long range communication costs much more than short range communication.

(3) minimization iterative methods : the CD method and the CG method

To minimize the quadratic function

$$F(x) = \tfrac{1}{2}x^T A x - b^T x \tag{2.15}$$

by the CD or CG method, we have to know a set of conjugate vectors of matrix A. Two vectors p, q are called mutually A-conjugate, if $p^T A q = 0$. Let vectors $p^{(0)}, p^{(1)}, \ldots, p^{(N-1)}$ be nonzero and mutually A-conjugate and $x^{(0)}$ be the initial guess. Since A is SPD, the set $\{p^{(n)}, 0 \le n \le N-1\}$ is also linearly independent. Because $\{p^{(n)}, 0 \le n \le N-1\}$ spans the whole space, there exists constants

$c_0, c_1, c_2, \ldots, c_N$ such that the minimum point $\bar{x}$ can be represented as

$$\bar{x} = x^{(0)} + c_0 p^{(0)} + c_1 p^{(1)} + \ldots + c_{N-1} p^{(N-1)} \quad , \tag{2.16.a}$$

and

$$c_n = \frac{(p^{(n)}, b - Ax^{(0)})}{(p^{(n)}, Ap^{(n)})} \quad , \tag{2.16.b}$$

which is the CD method. The set of vectors $\{ p^{(n)} , 0 \le n \le N-1 \}$ can be chosen in many different ways and if we choose

$$\begin{aligned} p^{(n)} &= r^{(n)} && \text{if } n = 0 \quad , \\ p^{(n)} &= r^{(n)} - \frac{(r^{(n)}, Ap^{(n-1)})}{(p^{(n-1)}, Ap^{(n-1)})} p^{(n-1)} && \text{if } n > 0 \quad , \end{aligned} \tag{2.17}$$

where

$$r^{(n)} = b - A\, x^{(n)} \quad ,$$

then the CD method becomes the CG method.

If we write down the local equation for the CD or CG methods explicitly, then it is easy to see that $A\, x^{(n)}$ and $A\, p^{(n)}$ can be computed in parallel locally. The main obstacle to localizability comes from the inner product operation. To see this, note that for any set $\{ p^{(n)} , 0 \le n \le N-1 \}$, some of the vectors $p^{(n)}$ must be global vectors, i.e., almost all of their entries are nonzero. Therefore, computing the inner product operation requires global communication. We encounter therefore the same difficulties for accelerated relaxation methods.

(4) Block iterative methods

Block iterative schemes are commonly used in practice in single processor computations. The idea is as follows. We may partition the grid points into several groups and divide the computation into two levels - inter-block level and intra-block level. The inter-block level usually adopts some accelerated relaxation method while the intra-block level may take advantage of the special structure of the submatrices, say, diagonal or tridiagonal matrices, and invert them directly. Two popular partitioning schemes are line partitioning and red/black partitioning, because they result in tridiagonal and diagonal submatrices respectively in most cases. The main advantage of block iterative methods is that global block relaxation converges faster than point relaxation.

In a mesh-connected processor array, line partitioning does not seem very useful; however, red/black partitioning is still attractive. In fact, we may prefer rectangular partitioning, of which red/black partitioning is a special case, i.e., 2×1 points per block. Using the rectangular partitioning, we can assign the nodes within the same rectangle to a single processor. The mapping is very simple because the geometry resulting from the partitioning matches the computer architecture very well. The rectangular block iterative methods not only increase the convergence rate but also provide an approach to solve a commonly encountered problem i.e., the number of grid points is usually larger than the number of processors available. Since the submatrices arising from rectangular regions have no special structure, we cannot solve them effectively using direct methods. Neverthless, we may choose some accelerated iterative methods in this case.

The two level computation concept seems a natural extension from one- processor computation to multiprocessor computation. Locally, by solving a smaller problem within a single processor as before, we can apply all kinds of acceleration schemes without worrying about communication cost. Globally, we come back to the same issue again - global information cost is high. So we may choose simple relaxation schemes without acceleration, say, Jacobi relaxation and Gauss-Seidel relaxation.

Given a problem with 100×100 discretized space points, we may group 10×10 points together and assign them to a single processor. Then, a 10×10 mesh-connected processor array is needed. The SOR, CSI, and CG algorithms can be used within each block and the optimal acceleration parameters are determined based on the 10×10 local information, which means it does not have any information contained in other processors except for the updated values of the points surrounding the block needed for next-step iterations. Similarly, if the block size becomes smaller, say, 5×5 points per block, then 20×20 processors are needed and the acceleration parameters will be determined based on the 5×5 local information, which should be different from the 10×10 points per block case.

Consider the intra-block accelerations (SOR, CSI, and CG methods) for a special case where each block only contains a single point (or a pair of points in the Gauss-Seidel relaxation case). It is known that ρ, the spectral radius of the Jacobi relaxation matrix for a square grid with $M \times M$ inner grid points, is $\cos\left(\frac{\pi}{M+1}\right)$ for the model problem. Therefore, ρ comes close to 0 as M approaches 1. Applying the result to equations (2.13.b) and (2.14.c), we have

$$\lim_{\rho \to 0} \omega_{n+1} = \lim_{\rho \to 0} \left[1 + \frac{T_{n-1}(\frac{1}{\rho})}{T_{n+1}(\frac{1}{\rho})} \right] = 1 \quad n \geq 1 , \tag{2.18}$$

$$\lim_{\rho \to 0} \omega = \lim_{\rho \to 0} \frac{2}{1 + \sqrt{1 - \rho^2}} = 1 \ . \tag{2.19}$$

Thus, the CSI accelerated block Jacobi method, (2.13.a) is reduced to the ordinary Jacobi method (2.10), and the SOR accelerated block Gauss-Seidel method (2.14.a), is reduced to the Gauss-Seidel method (2.11).

As to the CD or CG accelerated block Jacobi method, the intra-block computation, equation (2.15), should be interpreted as

$$A = diag\ (-\frac{1}{4} , -\frac{1}{4} , 1 , -\frac{1}{4} , -\frac{1}{4}) \tag{2.20.a}$$

where "diag" means a 5×5 diagonal matrix,

$$x = (u_{i-1,j} , u_{i+1,j} , u_{i,j} , u_{i,j-1} , u_{i,j+1})^T \tag{2.20.b}$$

$$b = (0 , 0 , -h^2 f_{i,j} , 0 , 0)^T \tag{2.20.c}$$

and only one vector $p^{(0)}$ has to be determined in (2.16.a), which assumes the form

$$p^{(0)} = (0 , 0 , 1 , 0 , 0)^T \ . \tag{2.20.d}$$

If we let $x^{(n)}$ denote $(u^{(n)}_{i-1,j} , u^{(n)}_{i+1,j} , u^{(n)}_{i,j} , u^{(n)}_{i,j-1} , u^{(n)}_{i,j+1})^T$, the equation (2.16.a) can be written as

$$x^{(n+1)} = x^{(n)} + \frac{(p^{(0)} , b - A\, x^{(n)})}{(p^{(0)} , A\, p^{(0)})} p^{(0)} \ . \tag{2.21}$$

Substituting (2.20) into (2.21) and simplifying it, we obtain the same result as Jacobi relaxation (2.10) again.

It seems natural that when the rectangular blocks become smaller and smaller, the behavior of the two-level computation scheme comes closer and closer to that of inter-block computational algorithms and the acceleration effect within a block becomes less important. The above examples tell us that we cannot get the acceleration effect using only local information. Therefore, the goal in Chapter 3 is to find some acceleration schemes using as little global information as possible.

Chapter 3. Locally Accelerated Successive Over-relaxation (LASOR) Algorithm in Mesh-connected Processor Arrays

3.1 Introduction

In this chapter, I will introduce an accelerated relaxation scheme called locally accelerated relaxation (LASOR) method, which differs from the SOR method mentioned in Chapter 2 in that it does not require global communication in every relaxation step.

If we compare the relaxation and minimization principles for iterative methods based on a local communication criterion, relaxation methods seem more attractive than conjugate gradient (CG) minimization methods. Relaxation methods without acceleration can be executed locally. However, CG minimization methods need to use all intermediate results to compute new search directions for the minimum.

The minimization approach may be modified in such a way that the new directions to be searched are restricted to a small subregion. Information exchange between these subregions is through their interfaces. This is equivalent to the two level computation strategy mentioned in Chapter 2, i. e., the block iterative methods which combine a global relaxation scheme with a local minimization scheme. Therefore, we conclude that relaxation methods provides a better choice for global level computation with VLSI implementations due to their local information exchange property.

Although block relaxation methods are promising in practice, for simplicity, we will focus on *point relaxation methods*. From a conceptual point of view, point relaxation methods are not very different from block relaxation methods. Consider a small subregion assigned to a processor. If there is no further discretization within it, we call this subregion a point; in contrast, if there is some inner structure, i.e. more discretized points, we call it a block. The intra-block level computation is the same as the traditional single processor computation problem, which is not our major concern. What we are really interested in is inter-block computation, the communication requirements between blocks and the time required to solve the problem. Internal block structure will not strongly effect these issues, and thus we will primarily focus on problems in which the block is just a single point. The results obtained for the analysis of point relaxation methods are also applicable to block relaxation.

In summary, this chapter attempts to find a good parallel computation algorithm under the following assumptions: (1) The computer architecture is a mesh-connected processor array, and each processor corresponds to one discretized node. (2) Communication only occurs between the neighboring processors. (3) Computation of the new value at a single node is based on its own last state (previous value at the same node) and the up-to-date values of neighboring nodes.

We restrict the problem domain to be the unit square. That is $\Omega = (0,1) \times (0,1)$, with the boundaries $\Gamma_1 = \{ (x_1, x_2) \mid x_1 = 0, 0 < x_2 < 1 \}$, $\Gamma_2 = \{ (x_1, x_2) \mid x_2 = 0, 0 < x_1 < 1 \}$, $\Gamma_3 = \{ (x_1, x_2) \mid x_1 = 1, 0 < x_2 < 1 \}$, and $\Gamma_4 = \{ (x_1, x_2) \mid x_2 = 1, 0 < x_1 < 1 \}$. This constraint not only simplifies the analysis but also has a practical reason. The processor array has usually a regular shape such as a square. In order to utilize all processors effectively, the irregular problem domain has to be mapped into this regular region [16]. Such a mapping is not trivial and usually introduces some edge effects which make the solution near the edges rather complicated. However, these issues will not be discussed here. Let us just assume such a mapping is possible and that after this mapping the linear property of the original PDE is still preserved.

Our approach to analyze distributed numerical PDE algorithms can be stated as follows.

Consider a linear second order partial differential operator L, four linear first order boundary condition operators B_i, and the functions u, f and g_i's satisfying

$$L\ u = f \qquad \textit{defined on } \Omega \tag{3.1.a}$$

$$B_i\ u = g_i \qquad \textit{defined on } \Gamma_i \qquad i = 1, 2, 3, 4 \tag{3.1.b}$$

Applying a discretization scheme to (3.1.a), we get

$$L_h\ u_h = f_h \qquad \textit{defined on } \Omega_h \tag{3.2}$$

where L_h is the discretized operator of L, u_h and f_h are discretized functions of u and f, and Ω_h is the set of discretized space points in Ω.

There are two ways to look at equation (3.2). We may view L_h as a matrix and u_h and f_h as vectors composed of the values of the functions at discretized space points. This is the central computation point of view. Alternately, we may interpret L_h as a group of operators distributed on Ω_h and u_h and f_h as input and output *waveforms*. This approach constitutes the distributed computation point of view.

To analyze a distributed numerical PDE algorithm, the *waveform interpretation* is convenient, because it relates all space points in the problem domain as a whole. In contrast, *matrix iterative analysis* [14] provides us with a loosely coupled large scale system. This system is very difficult to understand and manipulate locally, because subregions far apart interact with each other in an indirect way.

One essential part of the PDE operator analysis is to find a set of good basis functions so that we may simplify the analytic procedure. The spectral theory of operators gives us many results in this respect [16]. Generally speaking, the best choice of basis functions are the eigenfunctions ξ_n of the global operator L satisfying the boundary conditions, because they form a *complete orthogonal* set with respect to the operator L and the boundary conditions. Thus, we have $L\ \xi_n = \lambda_n\ \xi_n$, where λ_n are the eigenvalues of L. The orthogonality implies that the inner product $(\ \xi_m\ , L\ \xi_n\ ) = 0$ when $m \neq n$, and the complete property means that any function u which satisfies the same boundary conditions can always be represented as a linear combination of ξ_n's, i.e., we can find a set of coefficients α_n's such that $u = \sum_n \alpha_n\ \xi_n$.

In distributed computation, we ask the same question - how to choose a set of good basis functions. Suppose we discretize a linear constant-coefficient difference operator within a local region. The eigenfunctions of a linear constant-coefficient operator are the sinusoidal functions, denoted by s_n. Suppose we use the sinusoidal functions as our basis functions. They form a *complete* set with respect to the boundary conditions. However, for constant-coefficient PDEs, they are orthogonal with respect to the global operator L, i.e., the inner product $(s_n , L\, s_m) = 0$ when $m \neq n$, but *not* with respect to the local operators $L_{h,i,j}$, i.e., the inner product $(s_{n,h} , L_{h,i,j}\, s_{m,h}) = s_{n,h}^T L\, s_{m,h} \neq 0$ when $m \neq n$, where $s_{n,h}$ is the discretized functions of s_n. For varying-coefficient PDEs, they are *not orthogonal* with respect to either the global operator L or the local operators $L_{h,i,j}$.

Fortunately, the orthogonal property is not relevant to the analysis of relaxation methods. The relaxation procedure in numerical PDE algorithms can be interpreted as a low-pass filtering process. The completeness of a basis is important, because the basis should be able to represent any error functions allowed in the problem domain. But since we do not intend to decompose the error function into components of the basis, the orthogonality is not useful.

In fact, the error is totally unknown during the relaxation procedure; as a consequence, there is no way to figure out the weighting of each basis function, whatever basis we choose. All we can do is to concentrate on the algorithm itself without considering what the error could be. Asymptotically, only the lowest frequency component remains due to the low-pass filtering property. The asymptotic assumption provides a common base to compare the convergence rates of different algorithms, which are called the *asymptotic convergence rates*.

Since we are only interested in the lowest frequency, the remaining question is which lowest frequency is more appropriate for our purpose - the smallest eigenvalue of the global operator L or that of the local operator $L_{h,i,j}$. The former requires some global information and a complicated computation procedure. The latter only needs information about the boundary conditions and a simple computation, discussed in next Sections.

3.2 Admissible Error Function Space and Its Lowest Fourier Component

It is more convenient to analyze the relaxation in the error function space rather than the solution space, because the error equations are homogeneous equations. The error space formulation can be obtained as follows. Let $\bar{u}$ be the actual solution, then

$$L\,\bar{u} = f \qquad \textit{defined on } \Omega \quad , \tag{3.3.a}$$

$$B_i\,\bar{u} = g_i \qquad \textit{defined on } \Gamma_i \quad i = 1, 2, 3, 4 \quad . \tag{3.3.b}$$

Substracting (3.3.a), (3.3.b) from (3.1.a), (3.1.b) respectively, we obtain the homogeneous PDE in the error space.

$$L\,e = 0 \qquad \textit{defined on } \Omega \quad , \tag{3.4.a}$$

$$B_i\,e = 0 \qquad \textit{defined on } \Gamma_i \quad i = 1, 2, 3, 4 \quad . \tag{3.4.b}$$

The functions defined on the unit square and satisfying the homogeneous boundary conditions (3.4.b) are called the *admissible error functions*. All admissible error functions form the *admissible error function space*. The sinusoidal functions in the admissible error function space can be chosen as a basis of this space. The goal of this section is to find in this basis the sinusoidal functions with the lowest frequency.

For simplicity, we assume that all B_i's are constant-coefficient operators. Under this assumption, B_1 and B_3 are independent of the x_2-direction, B_2 and B_4 are independent of the x_1-direction, and since the problem domain is square, the admissible Fourier components can be written in the separable form as $s_1(x_1)\,s_2(x_2)$, where $s_1(\cdot)$ and $s_2(\cdot)$ are two 1-D sinusoidal functions. The boundary condition on Γ_1 becomes

$$B_1 s_1(x_1)\,s_2(x_2) = s_2(x_2)\,B_1 s_1(x_1) = 0 \quad , \tag{3.5.a}$$

i. e.,

$$B_1 s_1(x_1) = 0 \quad . \tag{3.5.b}$$

Similarly, we simplify the boundary conditions on Γ_2, Γ_3, and Γ_4, then decompose the 2-D problem into two independent 1-D problems.

$$(I)\ B_1 s_1(x_1) = 0 \quad when\ x_1 = 0 \quad B_3 s_1(x_1) = 0 \quad when\ x_1 = 1\ , \tag{3.6.a}$$

$$(II)\ B_2 s_2(x_2) = 0 \quad when\ x_2 = 0 \quad B_4 s_2(x_2) = 0 \quad when\ x_2 = 1\ , \tag{3.6.b}$$

From (3.6.a) and (3.6.b), we can determine the lowest frequencies $\hat{k}_1$ and $\hat{k}_2$ separately. We only show how to get $\hat{k}_1$ from (I); then $\hat{k}_2$ can be obtained from (II) in the same way.

Consider the mixed type boundary conditions,

$$B_1 = b_1 + b_2 \frac{d}{d\,x_1} \quad when\ x_1 = 0 \quad \text{and} \quad B_3 = b_3 + b_4 \frac{d}{d\,x_1} \quad when\ x_1 = 1 \tag{3.7}$$

The Fourier component of frequency k_1, $s\,(k_1, x_1)$, can be written as a linear combination of two complex sinusoids $e^{i k_1 x_1}$ and $e^{-i k_1 x_1}$, i. e.,

$$s\,(k_1, x_1) = c\,(k_1)\,e^{i k_1 x_1} + c\,(-k_1)\,e^{-i k_1 x_1} \quad . \tag{3.8}$$

Substituting (3.8) into (3.7), we obtain

$$\begin{gathered}(b_1 + i\,b_2 k_1)\,c\,(k_1) + (b_1 - i\,b_2 k_1)\,c\,(-k_1) = 0 \\ (b_3 + i\,b_4 k_1)\,e^{i k_1 x_1} c\,(k_1) + (b_3 - i\,b_4 k_1)\,e^{-i k_1 x_1} c\,(-k_1) = 0\end{gathered} \tag{3.9}$$

In order to get nonzero values for $c\,(k_1)$ and $c\,(-k_1)$, the determinant of the $2 \times$

2 coefficient matrix should equal zero, or equivalently,

$$e^{i\,2k_1} = \frac{(\,b_1 + i\,b_2 k_1\,)\,(\,b_3 - i\,b_4 k_1\,)}{(\,b_1 - i\,b_2 k_1\,)\,(\,b_3 + i\,b_4 k_1\,)} \tag{3.10}$$

Therefore, we conclude that the frequency k_1 of any admissible 1-D sinusoidal function with respect to the boundary conditions (3.7) must satisfy equation (3.10).

Let us look at two examples. If both Γ_1 and Γ_3 are Dirichlet type boundary conditions, which means b_2 and b_4 are zeros, then (3.10) becomes

$$e^{i\,2k_1} = 1 \quad or \quad \cos 2\,k_1 + i\,\sin 2\,k_1 = 1 \quad . \tag{3.11}$$

The solutions are $k_1 = n\,\pi$, $n = 0, \pm 1, \pm 2, \ldots$. However, it is easy to see that the zero frequency (d.c. error) cannot be allowed. Thus, the lowest Fourier frequency $\hat{k}_1$ in the admissible error space is π. If we change the boundary condition on Γ_3 to be of Neumann type, i. e., $b_3 = 0$ but $b_4 \neq 0$, then (3.10) becomes

$$e^{i\,2k_1} = -\,1 \quad or \quad \cos 2\,k_1 + i\,\sin 2\,k_1 = -\,1 \quad . \tag{3.12}$$

The solutions are $k_1 = \frac{1}{2}\,n\,\pi$, n is odd and the lowest frequency $\hat{k}_1$ is $\frac{\pi}{2}$.

The same procedure applies to other boundary conditions such as periodic boundary condition. Notice that the determination of the lowest Fourier components of a given PDE is only related to its boundary conditions and the geometry of the problem domain. The procedure has nothing to do with the PDE operator L itself! The lowest frequencies $\hat{k}_1$, $\hat{k}_2$ are important for determining the spectral radius of a local operator $L_{h,i,j}$, as shown in the following section.

3.3 Local Relaxation Operator and Its Properties

We use the model problem mentioned in Chapter 2 as an illustrative example. We rewrite equation (2.8) as

$$\frac{\partial^2 u(x_1,x_2)}{\partial x_1^2} + \frac{\partial^2 u(x_1,x_2)}{\partial x_2^2} = f(x_1,x_2) \quad (x_1,x_2) \in \Omega \text{ ,} \tag{3.13.a}$$

$$u(x_1,x_2) = g(x_1,x_2) \quad (x_1,x_2) \in \Gamma \text{ ,} \tag{3.13.b}$$

where

$$\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4 \quad .$$

The discretized function $u(x_1,x_2)$ is represented as a 2-D space sequence and $u_{i,j}$. An element of this sequence is defined at *the node* (i,j), or on *the space point* (ih, jh), as

$$u_{i,j} = u(ih, jh) \ , \ h = \frac{1}{M+1} \ , \ (i,j) \in \Omega_M \ or \ (i,j) \in \Gamma_M \tag{3.14}$$

where

$$\Omega_M = \left\{ (i,j) \mid i = 1, 2, \cdots M, \ j = 1, 2, \cdots M. \right\}$$

$$\Gamma_M = \left\{ (i,j) \mid i = 0 \ or \ M+1, \ or \ \ j = 0 \ or \ M+1 \right\} \ .$$

$f(x_1,x_2)$ and $g(x_1,x_2)$, can be discretized similarly, except that the 2-D sequence data element $f_{i,j}$ is defined when $(i,j) \in \Omega_M$ and the 1-D sequence data element $g_{i,j}$ is defined when $(i,j) \in \Gamma_M$.

The partial operator $\frac{\partial^2}{\partial x_1^2}$ and $\frac{\partial^2}{\partial x_2^2}$ are discretized by performing the following substitution

$$\frac{\partial^2}{\partial x_1^2} \longleftrightarrow \frac{E_1 - 2 + E_1^{-1}}{h^2} \quad , \quad \frac{\partial^2}{\partial x_2^2} \longleftrightarrow \frac{E_2 - 2 + E_2^{-1}}{h^2} \quad . \tag{3.15}$$

where E_1 and E_1^{-1} (E_2 and E_2^{-1}) are called x_1 - *direction (x_2 - direction) forward-shifting* and *backward-shifting operators* separately. Their definitions are

$$\begin{array}{ll} E_1 v_{i,j} = v_{i+1,j} & E_1^{-1} v_{i,j} = v_{i-1,j} \\ E_2 v_{i,j} = v_{i,j+1} & E_2^{-1} v_{i,j} = v_{i,j-1} \end{array} \tag{3.16}$$

These shifting operators are an indispensable tool in the representation of the discrete local operators. Here, let us digress a while to discuss some of their properties. If P and Q are shifting operators, then their sum and product can be defined as

$$(P + Q) f = P f + Q f \quad \text{and} \quad P Q f = P (Q f) \tag{3.17}$$

Based on these shifting operators, a general discrete linear local operator at a node can be represented as

$$P = \sum_{r_1} \sum_{r_2} a (r_1 , r_2) E_1^{r_1} \ E_2^{r_2}$$

where r_1 and r_2 are integers, as long as the grid spacing is uniform and the domain of the operator is within Ω_M. It can be shown rigorously that all possible discrete local operators defined above with the addition and multiplication operation form a ring [17]. Since they have such nice properties, it is convenient to use these operators to understand the behavior of distributed numerical computation problems.

Usually, one-step local operators contain only several low order terms, say, the $\frac{1}{4} (E_1 + E_1^{-1} + E_2 + E_2^{-1})$, which means that only neighboring nodes interact. However, multi-step operators, which are composed of several one-step local operators, usually contain high order terms, indicating that as time increases more and more nodes interact. For example, an m-step operator can be formed by repeating the above one-step operator m times, and denoted by $[\frac{1}{4}(E_1 + E_1^{-1} + E_2 + E_2^{-1})]^m$, We call this phenomenon the *propagation of local operators*, which has no counterpart from a central computation point of view and is believed to be an important issue for distributed computation. We will briefly touch on this problem in Chapter 4.

The discretized form of equation (3.13) is

$$L_{h,i,j}\, u_{i,j} = f_{i,j} \quad (i, j) \in \Omega_M \quad , \tag{3.18.a}$$

$$u_{i,j} = g_{i,j} \qquad (i, j) \in \Gamma_M \quad . \tag{3.18.b}$$

where $L_{h,i,j} \equiv \dfrac{E_1 + E_1^{-1} + E_2 + E_2^{-1} - 4}{h^2}$ is called the *local discretized differential operator at node* (i, j). The Jacobi relaxation at a local node can be written as

$$u_{i,j}^{(n+1)} = J_{i,j}\, u_{i,j}^{(n)} - h^2 f_{i,j} \qquad m = 0, 1, 2, \cdots \qquad (i,j) \in \Omega_M \tag{3.19}$$

where $J_{i,j}$, the *local Jacobi relaxation operator*, is $\dfrac{E_1 + E_1^{-1} + E_2 + E_2^{-1}}{4}$.

From the error space point of view, we get

$$e_{i,j}^{(n+1)} = J_{i,j}\, e_{i,j}^{(n)} \qquad m = 0, 1, 2, \cdots \qquad (i,j) \in \Omega_M \tag{3.20}$$

The low pass filtering property of the local relaxation operator $J_{i,j}$ can easily be understood if we assume that the input is the complex sinusoid $e^{i(k_1x_1+k_2x_2)}$, which is an eigenfunction of $J_{i,j}$ with eigenvalue $\lambda_{i,j}(k_1, k_2) = \dfrac{\cos k_1h + \cos k_2h}{2}$. The eigenvalue function $\lambda_{i,j}(k_1, k_2)$ is usually called the *frequency response* in signal processing [18] and the relaxation operator can be thought of as a filtering process in the frequency domain. The frequency response function $\dfrac{\cos k_1h + \cos k_2h}{2}$, in fact, represents a 2-D notch filter instead of a low pass filter. However, we have to consider another constraint arising from the discretization procedure. The easiest way to explain the discretization comes from the Taylor's series approximation of a function $f\ (x\ )$, i. e.,

$$f\ (x\ ) = f\ (x_0) + (x - x_0) f'(x_0) + \frac{(x - x_0)^2}{2!} f''(x_0) + \cdots \quad (3.21)$$

Supposing $f\ (x\ ) = e^{ikx}$ and $x = x_0 + h$, the high order terms are negligible only if the product kh is reasonably small, say, less than 1. That means that as long as the magnitude of wavevector k is bounded we can always fine a discretization spacing h, which is fine enough so that the dimensionless frequencies $\theta_1 = k_1 h$ and $\theta_2 = k_2 h$ are always within the unit circle in the (θ_1 , θ_2) plane. In this region, the notch filter indeed behaves like a low pass filter.

If $| \lambda_{i,j} (k_1 , k_2) | < 1$ for all admissible frequencies, then the local relaxation operator is said to be *convergent*. A distributed numerical algorithm is said to be convergent if all local operators are convergent.

The low pass filtering property makes the error at higher frequencies converge to zero faster than that at lower frequencies. The eigenvalue with the largest magnitude, occurring in the lowest frequency, is the dominant factor in the asymptotic convergence rate analysis. We call this magnitude the *spectral radius* of the local operator. In order to determine the spectral radius relaxation operator, we only have to know the lowest admissible Fourier component corresponding to the given boundary conditions, discussed in Section 3.2, and then to compute $\lambda_{i,j} (\hat{k}_1 , \hat{k}_2)$.

In the model problem example, we know that the lowest Fourier frequency allowed by the Dirichlet boundary conditions is $(\hat{k}_1, \hat{k}_2) = (\pi, \pi)$, so that the spectral radius of the operator $J_{i,j} = \frac{E_1 + E_1^{-1} + E_2 + E_2^{-1}}{4}$ is $\frac{1}{2}(\cos \pi h + \cos \pi h)$.

Finally, we want to discuss the symmetric and Hermitian properties of the local operator, because they will be used in the analysis of the LASOR algorithm. We say that a linear local operator P is *symmetric* if $P(E_1, E_2) = P(E_1^{-1}, E_2^{-1})$ and P is *Hermitian* if $P(E_1, E_2) = \bar{P}(E_1^{-1}, E_2^{-1})$, where $\bar{P}$ means taking the complex conjugate of all coefficients of the various shifting operators. It can be proved easily that the eigenvalues of a local Hermitian operator are real.

3.4 LASOR Algorithm

From the above discussion, we know how to determine the spectral radius of a local relaxation operator, which not only gives us some knowledge of the local convergence rate but also provides the information required for computing the local optimal relaxation factor, discussed below.

The LASOR method can be applied to any local region where the Jacobi relaxation operator $J_{i,j}$ is convergent, Hermitian and smooth. The convergent and Hermitian assumptions guarantee that all eigenvalues of $J_{i,j}$ are real and less than 1. However, the smoothness property needs some explanations.

We divide the problem domain into red and black points and update one color at each time. Suppose we start with the relaxation of the red points as

$$e_{i,j}^{(n+1)} = (1 - \omega_{i,j})\, e_{i,j}^{(n)} + \omega_{i,j} J_{i,j}\, e_{i,j}^{(n)} \qquad i+j \text{ is even} \;,\; n \ge 0 \;. \qquad (3.22)$$

The next step is to relax the neighboring black points,

$$e_{i,j}^{(n+1)} = (1 - \omega_{i,j})\, e_{i,j}^{(n)} + \omega_{i,j} J_{i,j}\, e_{i,j}^{(n+1)} \qquad i+j \text{ is odd} \;,\; n \ge 0 \;. \qquad (3.23)$$

If all $J_{i,j}$'s are approximately the same within that small region, then we can combine (3.22) with (3.23) and rewrite equation (3.23) to be

$$e_{i,j}^{(n+1)} = (1 - \omega_{i,j})\, e_{i,j}^{(n)} + \omega_{i,j}(1 - \omega_{i,j}) J_{i,j}\, e_{i,j}^{(n)} + \omega_{i,j}^2 J_{i,j}^2\, e_{i,j}^{(n)} \qquad (3.24)$$
$$i+j \text{ is odd} \;,\; n \ge 0 \;.$$

The smoothness assumption is appropriate for regions where the coefficients of the differential operator are continuous.

Now, equations (3.22) and (3.23) can be written in the following form

$$e_R^{(n+1)} = (1 - \omega_{i,j})\, e_R^{(n)} + \omega_{i,j} J_{i,j}\, e_B^{(n)} \quad , \; n \ge 0 \;, \qquad (3.25.a)$$

$$e_B^{(n+1)} = (1 - \omega_{i,j})\, e_B^{(n)} + \omega_{i,j} J_{i,j}\, e_R^{(n+1)} \quad , \; n \ge 0 \;, \qquad (3.25.b)$$

where e_R and e_B represent the errors at the red and black points around the node (i, j) separately. Notice that equation (3.25) describes, in fact, the relation of two waves - the *red* and *black* waves in that local subregion around the node (i, j). Rearranging and simplifying (3.25), we obtain the following iteration relationship between two successive steps,

$$\begin{pmatrix} e_R^{(n+1)} \\ e_B^{(n+1)} \end{pmatrix} = \begin{bmatrix} 1 - \omega_{i,j} & \omega_{i,j} J_{i,j} \\ \omega_{i,j}(1 - \omega_{i,j}) J_{i,j} & 1 - \omega_{i,j} + \omega_{i,j}^2 J_{i,j}^2 \end{bmatrix} \begin{pmatrix} e_R^{(n)} \\ e_B^{(n)} \end{pmatrix} \qquad n \ge 0 \; (3.26)$$

where the 2×2 matrix operator, denoted by $G_{i,j}(\omega_{i,j}, J_{i,j})$, is called *the local*

LASOR operator with relaxation factor $\omega_{i,j}$ at node (i , j).

Assuming that an eigenfunction of the LASOR operator $G_{i,j}$ has the form $(c_1 e^{i(k_1 x_1 + k_2 x_2)} , c_2 e^{i(k_1 x_1 + k_2 x_2)})^T$ and that the corresponding eigenvalue is $\mu_{i,j}$, we may write

$$G_{i,j} (\omega_{i,j} , J_{i,j}) \begin{pmatrix} c_1 e^{i(k_1 x_1 + k_2 x_2)} \\ c_2 e^{i(k_1 x_1 + k_2 x_2)} \end{pmatrix} = \mu_{i,j} \begin{pmatrix} c_1 e^{i(k_1 x_1 + k_2 x_2)} \\ c_2 e^{i(k_1 x_1 + k_2 x_2)} \end{pmatrix} \quad .(3.27)$$

The equation (3.27) can be further simplified because of the assumption that the local operator $J_{i,j}$ is approximately constant in the region around the node $(i . j)$ and has the eigenvalue $\lambda_{i,j}$ for this complex sinusoid $e^{i(k_1 x_1 + k_2 x_2)}$. We obtain

$$G_{i,j} (\omega_{i,j} , \lambda_{i,j}) \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \mu_{i,j} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \quad . \tag{3.28}$$

Note that the eigenvalues of the operator matrix $G_{i,j} (\omega_{i,j} , J_{i,j})$ are the same as those of the matrix $G_{i,j} (\omega_{i,j} , \lambda_{i,j})$.

Furthermore, from (3.28), we know that $\lambda_{i,j} (k_1 , k_2)$ and $\mu_{i,j} (k_1 , k_2)$ can be related via

$$\left| G_{i,j} (\omega_{i,j} , \lambda_{i,j}) - \mu_{i,j} I \right| = 0 \quad , \tag{3.29}$$

or, equivalently,

$$\mu_{i,j}^2 - (2 - 2 \omega_{i,j} + \omega_{i,j}^2 \lambda_{i,j}^2) \mu_{i,j} + (1 - \omega_{i,j})^2 = 0 \quad . \tag{3.30}$$

Therefore, we get

$$\mu_{i,j} = 1 - \omega_{i,j} + \frac{\omega_{i,j}^2 \lambda_{i,j}^2}{2} \pm \frac{\sqrt{\Delta}}{2} \tag{3.31}$$

$$\textit{where} \quad \Delta = 4 (1 - \omega_{i,j}) \omega_{i,j}^2 \lambda_{i,j}^2 + \omega_{i,j}^4 \lambda_{i,j}^4 \quad .$$

Let us consider the special case, $\omega_{i,j} = 1$, which corresponds to the Gauss-Seidel relaxation method. The eigenvectors of the 2×2 matrix $G_{i,j}(\omega_{i,j}, \lambda_{i,j})$ are $(1, 0)^T$ and $(1, \lambda_{i,j})^T$ and the corresponding eigenvalues are 0 and $\lambda_{i,j}^2$. This means that if we start with two sinusoidal waveforms at the same frequency but with different amplitudes, one of them, the red wave represented by the vector $(1, 0)$, disappears in one step. Only the other wave remains and alternates between the red and black points thereafter, as mentioned in Chapter 2. The ratio of the updated wave and the old wave is equal to the constant $\lambda_{i,j}$, so that the amplitude is reduced by a factor of $\lambda_{i,j}^2$ per cycle.

The purpose of introducing the relaxation parameter $\omega_{i,j}$ is to make the eigenvalue of the new operator $G_{i,j}$, i. e., $\mu_{i,j}(k_1, k_2, \omega_{i,j})$, smaller than that of the old operator $J_{i,j}$, i. e., $\lambda_{i,j}(k_1, k_2)$. For a fixed real $\lambda_{i,j}(k_1, k_2)$, the relationship between μ and $\omega_{i,j}$ can be described by the root locus technique described in Figure 3.1.

When $0 < \omega_{i,j} < 2$, the magnitude of μ is less than one, and the local LASOR operator $G_{i,j}$ converges. In addition, when $\Delta = 0$, the two eigenvalues μ_1 and μ_2 coincide and the largest possible magnitude of these two eigenvalues, $\mu_{i,j,m} \equiv \max(|\mu_{i,j,1}|, |\mu_{i,j,2}|)$, is minimized. This $\omega_{i,j}$, called the *optimal relaxation factor with respect to a specific* $\lambda_{i,j}$, and denoted by $\omega_{i,j,opt}(\lambda_{i,j})$, can be found by solving

$$4(1 - \omega_{i,j})\,\omega_{i,j}^2\,\lambda_{i,j}^2 + \omega_{i,j}^4\,\lambda_{i,j}^4 = 0 \quad \text{and} \quad 0 < \omega_{i,j} < 2 \quad . \qquad (3.32)$$

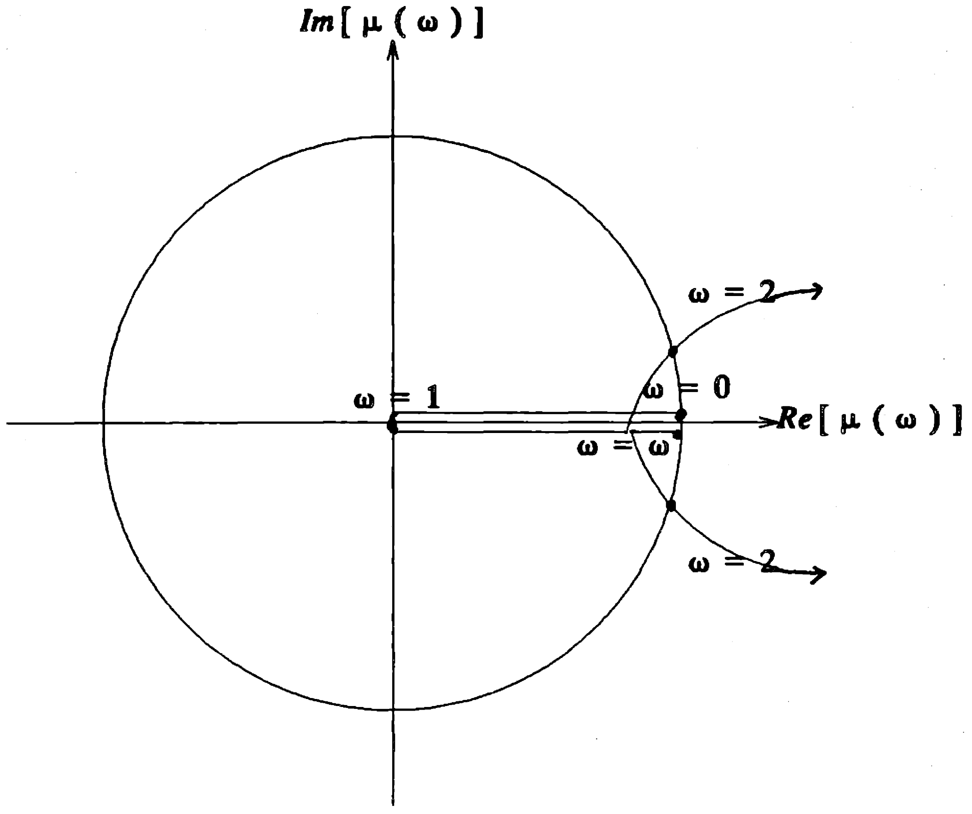

Fig 3.1 Root Loci of $\mu(\omega)$ with Fixed λ

The solution is

$$\omega_{i,j,opt}(\lambda_{i,j}) = \frac{2}{1 + \sqrt{1 - \lambda_{i,j}^2}} \quad . \tag{3.33}$$

The general relation between $\mu_{i,j,m}$ and $\omega_{i,j}$ can be derived in a straightforward way from equation (3.31) and is given by

$$\mu_{i,j,m} = \omega_{i,j} - 1 \qquad \omega_{i,j,opt}(\lambda_{i,j}) \le \omega_{i,j} < 2 \ , \tag{3.34.a}$$

$$\mu_{i,j,m} = \left[\frac{\omega_{i,j}\lambda_{i,j} + \sqrt{\omega_{i,j}^2 \lambda_{i,j}^2 + 4(\omega_{i,j} - 1)}}{2} \right]^2 \quad 0 < \omega_{i,j} \le \omega_{i,j,opt} \tag{3.34.b}$$

The minimum value of all possible $\mu_{i,j,m}$'s is, therefore, $\omega_{i,j,opt}(\lambda_{i,j}) - 1$.

Since $\lambda_{i,j}$ is a function of frequency, equation (3.33) implies that differing frequencies require different optimal relaxation factors. However, we are allowed to choose only one $\omega_{i,j}$, so we have to consider the overall performance, i.e., $\omega_{i,j}$ has to be selected so that the spectral radius of the LASOR operator $G_{i,j}(\omega_{i,j}, J_{i,j})$ is minimized over all frequencies. Let $\rho_{i,j}$ be the spectral radius of the local operator $J_{i,j}$ and $\lambda_{i,j}$ be an another arbitrary eigenvalue of $J_{i,j}$. By definition, $|\lambda_{i,j}| < \rho_{i,j}$, so we know that $\omega_{i,j,opt}(\lambda_{i,j}) < \omega_{i,j_{opt}}(\rho_{i,j})$ from (3.33). Using the relation in (3.34), we reason as follows. If we choose $\omega_{i,j} = \omega_{i,j,opt}(\lambda_{i,j})$, $\mu_{i,j,m}(\lambda_{i,j})$ is minimized to be $\omega_{i,j,opt}(\lambda_{i,j}) - 1$ while $\mu_{i,j,m}(\rho_{i,j})$ is greater than $\omega_{i,j,opt}(\rho_{i,j}) - 1$. On the other hand, if $\omega_{i,j,opt}(\rho_{i,j})$ is chosen to be the relaxation factor, both $\mu_{i,j,m}(\lambda_{i,j})$ and $\mu_{i,j,m}(\rho_{i,j})$ are equal to $\omega_{i,j,opt}(\rho_{i,j}) - 1$. Comparing these two cases, the latter choice is the best scheme to minimize the spectral radius of $G_{i,j}(\omega_{i,j}, J_{i,j})$. This optimal $\omega_{i,j}$ for overall consideration is denoted as $\omega_{i,j}^*$, and

$$\omega_{i,j}^{*} = \omega_{i,j,opt}\,(\,\rho_{i,j}\,) = \frac{2}{1+\sqrt{1-\rho_{i,j}^{2}}} \quad . \tag{3.35}$$

We summarize the above result in the following theorem.

Theorem 3.1 :

Let $J_{i,j}$ be a local Jacobi relaxation operator defined at the node $(\,i\,,j\,)$, which is smooth, convergent, and Hermitian in the neighborhood of the node $(\,i\,,j\,)$. In addition, the region is partitioned into red and black points and the grid points of the same color cannot be coupled through $J_{i,j}$. Then, the LASOR algorithm at the node $(\,i\,,j\,)$ can be described by the two successive relaxations

$$e_{i,j}^{(n+1)} = (1-\omega_{i,j}\,)\,e_{i,j}^{(n)} + \omega_{i,j}\,J_{i,j}\,e_{i,j}^{(n)} \qquad i+j \text{ is even (red points) },\ n \geq 0\ ,$$

$$e_{i,j}^{(n+1)} = (1-\omega_{i,j}\,)\,e_{i,j}^{(n)} + \omega_{i,j}\,J_{i,j}\,e_{i,j}^{(n+1)} \qquad i+j \text{ is odd (black points) },\ n \geq 0\ ,$$

Equivalently, the LASOR matrix operator can be represented as

$$G_{i,j}\,(\,\omega_{i,j}\,,J_{i,j}\,) = \begin{bmatrix} 1-\omega_{i,j} & \omega_{i,j}\,J_{i,j} \\ \omega_{i,j}\,(1-\omega_{i,j}\,)\,J_{i,j} & 1-\omega_{i,j}+\omega_{i,j}^{2}\,J_{i,j}^{2} \end{bmatrix} \quad .$$

Then the spectral radius $\rho_{i,j}\,(\,G_{i,j}\,)$ of $G_{i,j}$ is minimized if and only if

$$\omega_{i,j} = \omega_{i,j}^{*} = \frac{2}{1+\sqrt{1-\rho_{i,j}^{2}(\,J_{i,j}\,)}}$$

where $\rho_{i,j}\,(\,J_{i,j}\,)$ is the spectral radius of $J_{i,j}$. And the minimum spectral radius satisfies

$$0 < \rho_{i,j}\,(\,G_{i,j}\,) = \omega_{i,j}^{*} - 1 < 1$$

The above theorem applies to general linear second order elliptic PDEs without the crossover term $\dfrac{\partial^2}{\partial x_1\,\partial x_2}$, because the crossover term mixes in the red and black points. However, if we have a crossover term, it is still possible to use the above analysis. In the following, we consider the simple case mentioned in [19].

Let the linear partial differential operator L be

$$L = \frac{\partial^2}{\partial x_1^2} + a(x_1, x_2) \frac{\partial^2}{\partial x_1 \partial x_2} + \frac{\partial^2}{\partial x_2^2} \quad \textit{where} \quad |a(x_1, x_2)| < 2 \tag{3.36}$$

The condition $| a(x_1, x_2) | < 2$ is required to guarantee that L is an elliptic operator. We also assume $a(x_1, x_2)$ is sufficiently smooth so that it can be viewed approximately as a constant locally. The following discretization schemes are used

$$\frac{\partial^2}{\partial x_1^2} \longleftrightarrow \frac{E_1 - 2 + E_1^{-1}}{h^2} \quad , \quad \frac{\partial^2}{\partial x_2^2} \longleftrightarrow \frac{E_2 - 2 + E_2^{-1}}{h^2} \quad , \tag{3.37}$$

$$\frac{\partial^2}{\partial x_1 \partial x_2} \longleftrightarrow \frac{E_1 E_2 + E_1^{-1} E_2^{-1} - E_1^{-1} E_2 - E_1 E_2^{-1}}{4 h^2} \quad .$$

The local Jacobi relaxation operator $J_{i,j}$ can be decomposed into two parts $J_{i,j,1}$ and $J_{i,j,2}$, i. e.,

$$J_{i,j} = J_{i,j,1} + J_{i,j,2} \tag{3.38.a}$$

where

$$J_{i,j,1} = \frac{E_1 + E_1^{-1} + E_2 + E_2^{-1}}{4} \quad , \tag{3.38.b}$$

$$J_{i,j,2} = a \, \frac{E_1 E_2 + E_1^{-1} E_2^{-1} - E_1^{-1} E_2 - E_1 E_2^{-1}}{16} \quad , \tag{3.38.c}$$

and

$$a = a(ih, jh) \ . \tag{3.38.d}$$

The $J_{i,j,2}$ operator couples the nodes of the same color together. It has been proposed to use a multi-color scheme to achieve decoupling [19]. However, a multi-color is not necessary in our distributed computation approach and, in fact, makes the analysis more complicated. Here, we still choose the red/black partition and show that it really makes little difference from the previous decoupled LASOR algorithm.

First, we write the coupled LASOR algorithm for the error in the local region as

$$e_R^{(n+1)} = (1 - \omega_{i,j}) e_R^{(n)} + \omega_{i,j} \left[J_{i,j,1} e_B^{(n)} + J_{i,j,2} e_R^{(n)} \right] \quad , n \ge 0 \quad ,(3.39.a)$$

$$e_B^{(n+1)} = (1 - \omega_{i,j}) e_B^{(n)} + \omega_{i,j} \left[J_{i,j,1} e_R^{(n+1)} + J_{i,j,2} e_B^{(n)} \right] \quad , n \ge 0 (3.39.b)$$

or, equivalently,

$$\begin{pmatrix} e_R^{(n+1)} \\ e_B^{(n+1)} \end{pmatrix} = G_{i,j} (\omega_{i,j} , J_{i,j,1} , J_{i,j,2}) \begin{pmatrix} e_R^{(n)} \\ e_B^{(n)} \end{pmatrix} \quad n \ge 0 \quad , \qquad (3.40)$$

where

$$G_{i,j} (\omega_{i,j} , J_{i,j,1} , J_{i,j,2}) =$$
$$\begin{bmatrix} 1 - \omega_{i,j} + \omega_{i,j} J_{i,j,2} & \omega_{i,j} J_{i,j,1} \\ \omega_{i,j} (1 - \omega_{i,j}) J + \omega_{i,j}^2 J_{i,j,1} J_{i,j,2} & 1 - \omega_{i,j} + \omega_{i,j} J_{i,j,2} + \omega_{i,j}^2 J_{i,j,1}^2 \end{bmatrix}$$

$\lambda_{i,j,1} (k_1 , k_2)$, $\lambda_{i,j,2} (k_1 , k_2)$, and $\lambda_{i,j} (k_1 , k_2)$, the eigenvalues of $J_{i,j,1}$, $J_{i,j,2}$, and $J_{i,j}$, are

$$\lambda_{i,j,1} (k_1 , k_2) = \frac{1}{2} (\cos k_1 h + \cos k_2 h) \qquad (3.41.a)$$

$$\lambda_{i,j,2} (k_1 , k_2) = \frac{a}{8} [\cos (k_1 + k_2) h - \cos (k_1 - k_2) h]$$
$$= - \frac{a}{4} \sin k_1 h \sin k_2 h \qquad (3.41.b)$$

$$\lambda_{i,j} (k_1 , k_2) = \frac{1}{2} (\cos k_1 h + \cos k_2 h) - \frac{a}{4} \sin k_1 h \sin k_2 h \quad (3.41.c)$$

It is enough to consider the frequency response within the unit circle of the (θ_1 , θ_2) plane. In this region, $J_{i,j,1}$ is a low pass filter while $J_{i,j,2}$ is a high pass filter. $J_{i,j}$ is similar to $J_{i,j,1}$ in the very low frequency range, because $\lambda_{i,j,2}$ is almost zero for the very low frequencies. In this case, we can view $J_{i,j,2}$ as a perturbation. Note that $J_{i,j}$, still a low pass filter, has better filtering capability than $J_{i,j,1}$ for all frequencies within the unit circle. Its spectral radius $\rho_{i,j}\,(J_{i,j})$ is determined by the lowest admissible frequency as before.

Following the procedure used in deriving the decoupled LASOR algorithm, we find that the optimal local relaxation factor for (3.39) is

$$\omega_{i,j}^{*} = \frac{2}{1-\epsilon+\sqrt{(1-\epsilon)^2-\rho_{i,j,1}^2}} \cong \frac{2}{1+\sqrt{1-\rho_{i,j,1}^2}} \qquad (3.42)$$

where $\epsilon = \lambda_{i,j,2}\,(\hat{k}_1\,,\hat{k}_2) \cong 0$ and where $\rho_{i,j,1}$, the spectral radius of $J_{i,j,1}$, is $\lambda_{i,j,1}\,(\hat{k}_1\,,\hat{k}_2)$. The spectral radius of the LASOR operator $G_{i,j}$ for this problem is

$$\mu_{i,j}\,(G_{i,j}\,(\omega_{i,j}^{*}\,,J_{i,j,1}\,,J_{i,j,2})) = \omega_{i,j}^{*} - 1 + \omega_{i,j}^{*}\,\epsilon \cong \omega_{i,j}^{*} - 1 \quad . \qquad (3.43)$$

Chapter 4. Performance Analysis of LASOR Algorithm

4.1 Introduction

The previous chapter described how to form the LASOR operator and how to choose the optimal local acceleration factor. This chapter is a continuation of Chapter 3 and will focus on the performance of the LASOR method.

The performance of a numerical iterative algorithm depends on two factors - time per iteration and number of iterations. As discussed in Chapter 2, time per iteration is primarily determined by communication cost in a multiprocessor environment. Therefore, communication complexity will be considered first in Section 4.2. Then, before going to a convergence rate analysis, I will prove the convergence of the LASOR algorithm in Section 4.3, and show that it holds not only for *synchronous computation* but also for *asynchronous computation*. Finally, Sections 4.4 and 4.5 present analytic and simulated results of the convergence rate of the LASOR method separately.

4.2 Communication Complexity

In order to understand the communication complexity of the LASOR method, rewrite equations (3.22) and (3.23) in the solution domain as follows,

$$u_{i,j}^{(n+1)} = (1 - \omega_{i,j})\, u_{i,j}^{(n)} + \omega_{i,j}\,(J_{i,j}\, u_{i,j}^{(n)} - h^2 f_{i,j}) \quad i+j \textit{ is even} , \qquad (4.1.a)$$

$$u_{i,j}^{(n+1)} = (1 - \omega_{i,j})\, u_{i,j}^{(n)} + \omega_{i,j}\,(J_{i,j}\, u_{i,j}^{(n+1)} - h^2 f_{i,j}) \quad i+j \textit{ is odd} . \qquad (4.1.b)$$

Then, the optimal local relaxation factor $\omega_{i,j}$ is

$$\omega_{i,j} = \omega_{i,j}^{*} = \frac{2}{1 + \sqrt{1 - \rho^2(J_{i,j})}} , \qquad (4.2)$$

where $\rho (J_{i,j})$ is the spectral radius of $J_{i,j}$

$$\rho (J_{i,j}) = \lambda (J_{i,j} , \hat{k}_1 , \hat{k}_2) \ . \tag{4.3}$$

Communication cost required by the LASOR method is almost the same as for the Jacobi or Gauss-Seidel methods except that the admissible lowest frequencies $\hat{k}_1$ and $\hat{k}_2$ need to be broadcast to all local processors in the loading stage. In the constant coefficient case, we may do even better. Because all local operators have the same spectral radii, the optimal local relaxation factors must be the same also. Thus, we only have to broadcast the unique optimal relaxation factor ω^*, instead of the lowest frequencies $\hat{k}_1$ and $\hat{k}_2$.

After the loading operation, communications for each iteration are only local, and have nothing to do with the size of the processor array. The time per iteration is, therefore, a constant, and not of the order $O (\frac{1}{h})$, as is the case of SOR implemented in a VLSI mesh-connected processor array.

4.3 Convergence Property

The convergence of the LASOR algorithm depends on the spectral radii of the local relaxation operators. If the spectral radii of *all* local Jacobi operators are less than one, it can be shown that LASOR converges in either synchronous or asynchronous computation.

4.3.1 Synchronous Case

First, let us consider the synchronous case. The convergence property can be stated as the following theorem.

Theorem 4.1 (Convergence Theorem of the Synchronous LASOR Algorithm) :
If all local Jacobi relaxation operators are smooth, convergent, and Hermitian and all local relaxation factors are between 0 and 2, then the synchronous LASOR algorithms converges.

---

proof:

In order to prove the convergence of the LASOR algorithm, we use an iterative matrix formulation. Starting from the linear system of equations, $A\ x = b$, where A is an $N \times N$ symmetric positive definite matrix, we may rewrite A to be

$$A = D - E - F = D\ (I - L - U)\ , \text{and} \quad L^T = U\ , \tag{4.4}$$

where I, D, E and F represent identity, diagonal, lower and upper triangular matrices accordingly, and $L = D^{-1} E \quad U = D^{-1} F$. Let W be the diagonal matrix formed by the local relaxation parameters, i.e., $W = diag\ (\omega_1, \omega_2, \ldots, \omega_N)$. Then, the LASOR algorithm can be written in matrix iterative form as

$$x^{(n+1)} = (I - W\ L)^{-1} [\ (I - W\ ) + W\ U]\ x^{(n)} + (I - W\ L)^{-1} D^{-1} W\ b\ . \tag{4.5}$$

The matrix iterative equation in the error space becomes

$$e^{(n+1)} = (I - W\ L\ )^{-1} [\ (I - W\ ) + W\ U\ ]\ e^{(n)} = G\ (\ W\ )\ e^{(n)}\ . \tag{4.6}$$

We use $G\ (\ W\ )$, an explicit function of the matrix W, to represent the iterative operator of the LASOR algorithm. If we can show that $\rho\ [\ G\ (\ W\ )\ ]$, the spectral radius of the iterative operator, is less than 1 under the condition $0 < \omega_i < 2\ , 1 \le i \le N$, then the theorem is proved.

Let λ be an arbitrary eigenvalue of $G(W)$ with the eigenvector μ, then $G(W)\mu = \lambda\mu$, or, equivalently,

$$[(I - W) + WU]\mu = \lambda(I - WL)\mu . \tag{4.7}$$

Premultiplying by $\mu^T W^{-1}$ on both sides, we obtain

$$(\mu, W^{-1}\mu) - (\mu, \mu) + (\mu, U\mu) = \lambda(\mu, W^{-1}\mu) - \lambda(\mu, L\mu) . \tag{4.8}$$

Defining $z = \dfrac{(\mu, L\mu)}{(\mu, \mu)}$ and $\dfrac{1}{\omega} = \dfrac{(\mu, W^{-1}\mu)}{(\mu, \mu)}$ and using the property

$$(\mu, U\mu) = (U^T\mu, \mu) = (L\mu, \mu) = \overline{(\mu, L\mu)} , \tag{4.9}$$

the above equation can be simplified as

$$\frac{1}{\omega} - 1 + \bar{z} = \frac{\lambda}{\omega} - \lambda z , \tag{4.10}$$

or, equivalently,

$$\lambda = \frac{1 - \omega + \omega\bar{z}}{1 - \omega z} . \tag{4.11}$$

Let $z = r e^{j\theta}$, then

$$|\lambda|^2 = \lambda\bar{\lambda} = 1 - \frac{\omega(2 - \omega)(1 - 2r\cos\theta)}{(1 - \omega r\cos\theta)^2 + \omega^2 r^2 \sin^2\theta} \tag{4.12}$$

We know that $|\lambda|^2$ is always positive. If we can show that the second term in the above expression is also positive, then we can conclude that $|\lambda|$ is less than 1. The denominator of the second term of equation (4.12) is positive, so we only have to consider the numerator.

$$2r\cos\theta = 2\,\mathrm{Re}(z) = \bar{z} + z = \frac{(\mu, L\mu)}{(\mu, \mu)} + \frac{(\mu, U\mu)}{(\mu, \mu)}$$

$$= 1 - \frac{(\mu, A\mu)}{(\mu, \mu)} < 1 , \tag{4.13}$$

where the inequality is due to the fact that A is positive definite with eigenvalues less than 1. Therefore, we know that $1 - 2r\cos\theta > 0$. Now, consider the range of the parameter ω. Given $W^{-1} = diag\,(\omega_1^{-1}, \omega_2^{-1}, \ldots, \omega_N^{-1})$ and assuming that all relaxation factors are positive, we have

$$\frac{1}{\omega_{\max}} < \frac{(\mu, W^{-1}\mu)}{(\mu, \mu)} = \frac{1}{\omega} < \frac{1}{\omega_{\min}}, \tag{4.14}$$

where $\omega_{\max}$ and $\omega_{\min}$ are the largest and smallest eigenvalues of the matrix W. Furthermore, if we set $0 < \omega_{\min} \le \omega_{\max} < 2$, then

$$0 < \omega_{\min} \le \omega \le \omega_{\max} < 2\ . \tag{4.15}$$

Under this condition, the second term in equation (4.12) is always positive, so that, the eigenvalues of the matrix $G\,(W)$ are all less than 1 and the LASOR algorithm converges.

Q.E.D.

4.3.2 Asynchronous Case

Next, let us consider the asynchronous case. Our analysis is based on a general theorem for the distributed asynchronous computation of fixed points proposed by Bertsekas [20]. In his paper, he gave several special cases which fit this general theorem and, therefore, can be shown to converge in an asynchronous and distributed way. There is one special case called *contraction mapping with respect to sup-norms* to which the LASOR operator belongs. As a consequence, the asynchronous convergence of the LASOR algorithm can be derived.

The convergence of distributed asynchronous computation of contraction mappings with respect to sup-norms can be stated as follows.

Lemma 4.2 (Convergence Theorem of Contraction Mappings with Respect to Sup-Norms) :

Consider a mapping $f : R^n \rightarrow R^n$ which has a fixed point x^*. Let x_i and f_i denote the coordinates of x and f, and define the sup-norm as

$$|| \, x \, || = Sup \; \alpha_i \; | \, x_i \, | \qquad \text{for } all \; i \tag{4.16}$$

where α_i's are nonnegative scalars. Assume that f is a contraction mapping with respect to the norm (4.16) in the sense that, for some $p < 1$ we have

$$|| \, f \, (x) - f \, (y) \, || \le p \; || \, x - y \, || \quad x \, , y \in R^n \; . \tag{4.17}$$

Then the asynchronous distributed iterative computation algorithm

$$x_i^{new} = f_i \, (\, x^{old} \,) \; \text{ for } all \; i$$

converges to the fixed point x^*. Above, x^{old} denotes the data available at the time of updating and x^{new} means the updated value after one iteration.

---

proof: See [20], p. 116

Based on the above lemma, we can derive the following theorem.

Theorem 4.3 (Convergence Theorem of the Asynchronous LASOR Algorithm) :

If all local Jacobi relaxation operators are smooth, convergent, and Hermitian and the local relaxation factor is between 0 and 2, i. e., $0 < \omega_{i,j} < 2$, then the spectral radii of all local LASOR operators

$$\rho_{i,j} \, [\, G_{i,j} \, (\, \omega_{i,j} \, , J_{i,j} \,) \,] < 1 \; , \tag{4.18}$$

and the asynchronous LASOR algorithm converges.

---

proof:

As a consequence of Theorem 3.1, we know that the spectral radii of all local LASOR operators are less than 1. All we have to prove here is convergence.

Define

$$\rho_{max} = \max_{i,j} (\rho_{i,j}) \quad , \tag{4.19}$$

then since all spectral radii are less than 1, ρ_{max} is also less than 1. Let us choose the sup-norm as

$$|| x || = | x_i | \qquad \text{for } all\ i \quad , \tag{4.20}$$

which is a special case of equation (4.16), i.e., $\alpha_i = 1$. We need to check whether the global LASOR operator given in equation (4.5) satisfies the condition (4.17). It is easy to see that

$$\begin{aligned} &|| f (x^{(n)}) - f (y^{(n)}) || = || x^{(n+1)} - y^{(n+1)} || \\ &= || (I - W L)^{-1} [(I - W) + W U] (x^{(n)} - y^{(n)}) || \\ &\leq \rho_{max} || x^{(n)} - y^{(n)} || \leq || x^{(n)} - y^{(n)} || \quad . \end{aligned} \tag{4.21}$$

Thus, the LASOR algorithm satisfies the condition of Lemma 4.2 and, therefore, it converges if implemented asynchronously.

Q.E.D.

4.4 Convergence Rate Analysis - Linear Constant Coefficient PDEs

In addition to the above convergence property, it is possible to discuss the convergence rate under stronger assumptions. First, assume that the relaxation is synchronous, which means that within some fixed time interval Δt, all processors have to complete one relaxation step and that the values at all nodes are updated at the end of this time interval. The second assumption is that the local relaxation operator is very smooth so that the eigenvalues of these operators are almost the same for the whole problem domain.

We have briefly discussed the smoothness issue in Chapter 3, where one-step LASOR is, in fact, composed of two Jacobi relaxation steps. The relationship between the multiple-step relaxation operator and the one-step relaxation operator can be explained using the concept of *propagation of local operators*. Assume that the local relaxation operators $J_{i,j}$ and $G_{i,j}$ are space-invariant and the grid is uniform and extends to infinity. Then, by mathematical induction, the p-step relaxation operators $J_{i,j}^{(p)}$ and $G_{i,j}^{(p)}$ at any node can be found to be

$$J_{i,j}^{(1)} = J_{i,j} \quad \text{and} \quad J_{i,j}^{(p)} = J_{i,j}^{p} \quad , \quad p \geq 1 \quad , \tag{4.22.a}$$

$$G_{i,j}^{(1)} = G_{i,j} \quad \text{and} \quad G_{i,j}^{(p)} = G_{i,j}^{p} \quad , \quad p \geq 1 \quad , \tag{4.22.b}$$

where the p-step relaxation operators $J_{i,j}^{(p)}$ and $G_{i,j}^{(p)}$ are defined as

$$u_{i,j}^{(n+p)} = J_{i,j}^{(p)} \, u_{i,j}^{(n)} \qquad n \geq 1 \quad \text{and} \quad p \geq 1 \quad , \tag{4.23.a}$$

$$\begin{pmatrix} e_R^{(n+p)} \\ e_B^{(n+p)} \end{pmatrix} = G_{i,j}^{(p)} \begin{pmatrix} e_R^{(n)} \\ e_B^{(n)} \end{pmatrix} \qquad n \geq 1 \quad \text{and} \quad p \geq 1 \quad . \tag{4.23.b}$$

However, once we set up the boundary conditions, and, furthermore, allow the local operator to be a function of space, the determination of the local p-step relaxation operator becomes nontrivial. The smoothness assumption of the local operator can help us to simplify the analysis if the propagation only occurs within a small area, i.e., if p is a small number. But for large p, we have to consider the boundary effect, which is still very complicated. This kind of approach to analyze the local relaxation behavior is called *pure local analysis*.

In stead of using pure local analysis, we may adopt a simpler point of view to interpret multiple-step relaxation phenomena. For example, consider the model problem mentioned in Chapter 2. We know that the global eigenfunctions and the local eigenfunctions are the same in this case. So, if the input waveform is an admissible sinusoidal function and all processors perform the relaxation once, then the output waveform should have the same shape but with a smaller amplitude. Since the eigenvalue of every local operator is the same as that of the global operator, the local spectral radius, equal to the global spectral radius, provides us the information of global convergence rate. The asymptotic convergence rate of a global iterative operator J, denoted by $R_\infty(J)$, is defined as

$$R_\infty(J) = -\ln \rho(J) . \tag{4.24}$$

Under the condition $\lambda(J) = \lambda(J_{i,j})$, the asymptotic convergence rate is given by

$$R_\infty(J) = -\ln \rho(J_{i,j}) . \tag{4.25}$$

Therefore, the global asymptotic convergence rate of the Jacobi relaxation can be computed as

$$\begin{aligned} R_\infty(J) &= -\ln \rho(J_{i,j}) = -\ln \cos \pi h \\ &\cong -\ln\left(1 - \frac{1}{2}\pi^2 h^2\right) \cong \frac{1}{2}\pi^2 h^2 \quad . \end{aligned} \tag{4.26}$$

Similarly, the asymptotic convergence rates of the Gauss-Seidel and the LASOR methods can be found to be

$$\begin{aligned} R_\infty[G(1, J)] &= -\ln \mu[G_{i,j}(1, J_{i,j})] = -\ln \cos^2 \pi h \\ &\cong -2\ln\left(1 - \frac{1}{2}\pi^2 h^2\right) \cong \pi^2 h^2 \quad , \end{aligned} \tag{4.27}$$

and

$$R_\infty [(G (W^* , J)] = - \ln \mu [(G_{i,j} (\omega^* , J_{i,j})] = - \ln (\frac{1-\sin\pi h}{1+\sin\pi h})$$

$$= - \ln (1 - 2 \pi h) = 2 \pi h \quad . \tag{4.28}$$

We call this approach a *semi-global analysis*. The difference between pure local analysis and semi-global analysis is that the former tries to build the multiple-step operator based on one-step local operators while the latter discusses how each global waveform is modified in multiple steps.

4.5 Computer Simulation

In this section, a simple numerical example is used to illustrate the convergence rate of the LASOR algorithm. The convergence rates of the SOR and CG methods are also shown for the purpose of comparison, For the SOR and CG methods, I use the ellpack software package [31].

The example chosen is

$$e^{xy} \frac{\partial^2 u}{\partial x^2} + e^{-xy} \frac{\partial^2 u}{\partial y^2} + e^{xy} y \frac{\partial u}{\partial x} - e^{xy} x \frac{\partial u}{\partial y} + \frac{1}{1 + x + y} u \tag{4.29.a}$$
$$= e^{xy} \sin (\pi x) \sin (\pi y)$$

in the unit square with the boundary conditions

$$u (x , y) = 0 , \qquad x = 0 \; or \; x = 1 \; or \; y = 0 \; or \; y = 1 \; . \tag{4.29.b}$$

For this test problem, two 5-point discretization schemes are used: one with grid spacing $\frac{1}{10}$, another with spacing $\frac{1}{30}$. Starting from the initial guess $u^{(0)} (x , y) = 0$ for all grid points, the maximum value of each iteration is plotted in Figures 4.1. The results indicate that the convergence rate of the LASOR algorithm is better than that of SOR algorithm, but worse than the CG algorithm. We also note that when the grid spacing is smaller, the convergence rate also becomes slower. This is consistent with our analysis before.

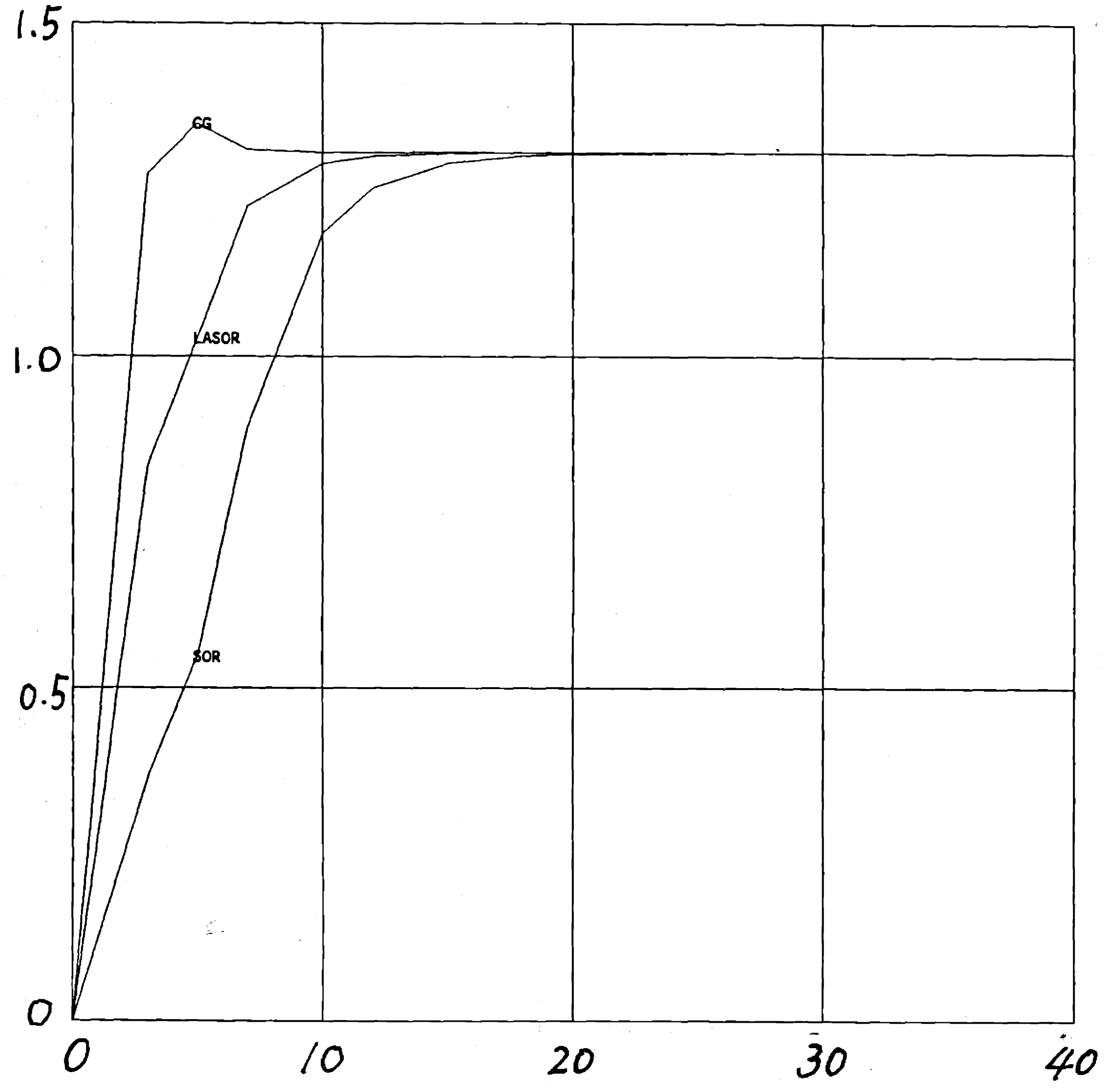

Fig 4.1.a Computer simulation result for the given example with 11 × 11 Grid. The x-axis is the number of iterations and the y-axis is the maximum value at each iteration.

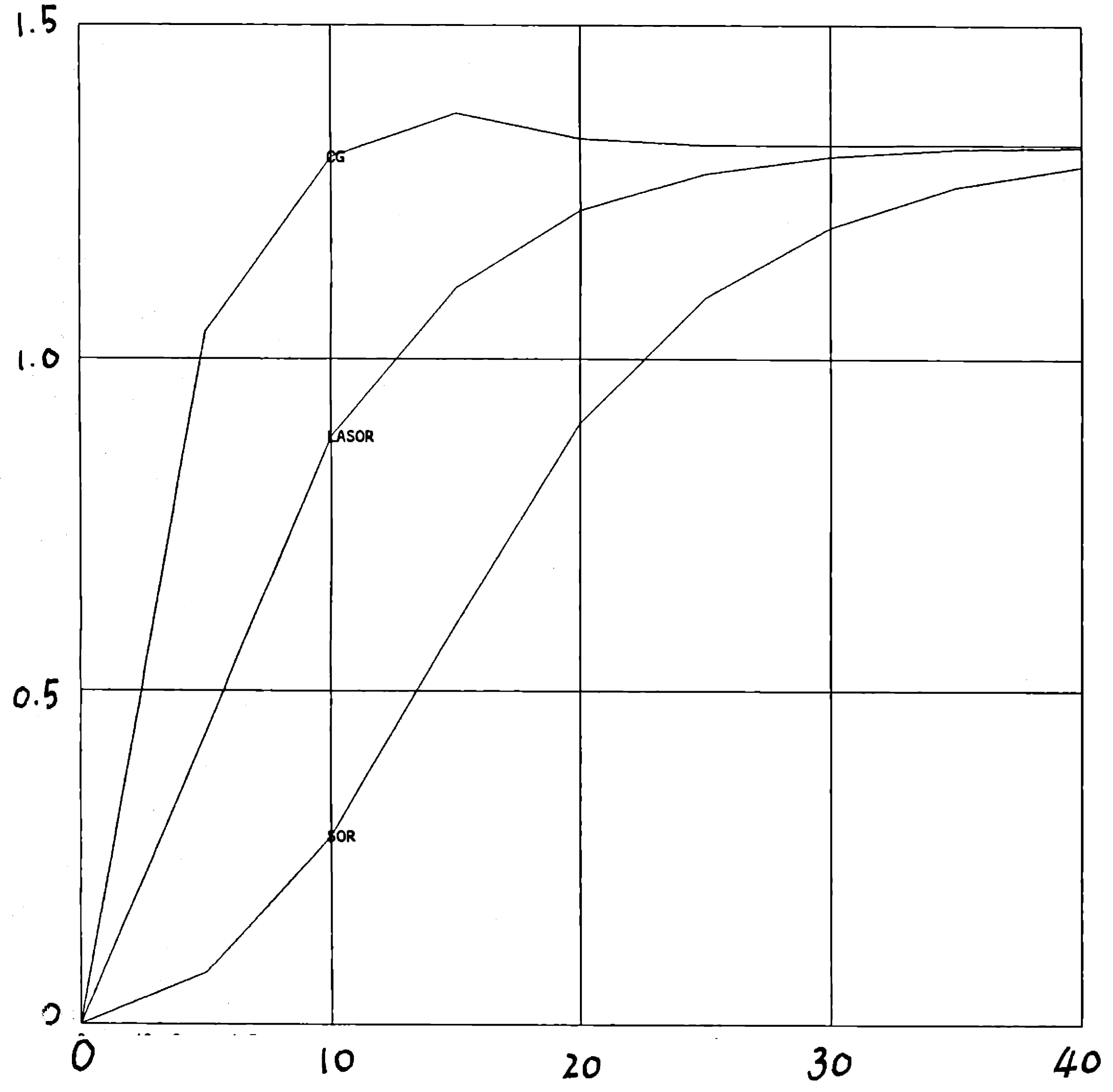

Fig 4.1.b Computer simulation result for the given example with 31 × 31 Grid. The x-axis is the number of iterations and the y-axis is the maximum value at each iteration.

Chapter 5. Concurrent Multigrid Method (CMG) and Supporting Architectures

5.1 Introduction

Multigrid methods (MG) have been studied intensively over the past ten years. The original idea dates back to Fedorenko and Bakhvalov[21][22]. But it is Brandt [23] who really recognized the actual efficiency of these methods and formulated them in a systematic way. MG methods can be simply described as follows. First, we start relaxation on a very fine grid level. When the relaxation on this grid level becomes inefficient, which implies that the error is very smooth, we transfer the information to a coarser grid and apply a new relaxation scheme. The procedure iterates back and forth between various grid spacings until the final error is within some predetermined small number.

The success of MG methods comes from several reasons. MG methods utilize the concept of splitting low and high frequencies approximated by coarse and fine grids separately and take advantage of the fact that different frequency bands have different optimal relaxation schemes. Combining these two features and applying them in a nested and iterative fashion lead to very efficient algorithms.

In other words, the three basic components of MG methods are :

(1) error smoothing by relaxation

(2) finer/coarser (or coarser/finer) grid transfer

(3) nested iteration

Because the last component implies repeating the first two steps iteratively, the real computational procedures needed in MG methods are relaxation and grid transfer. These two types of computation are both space distributed. In addition, they only require local information exchange. As a consequence, it seems obvious that MG methods can be implemented in parallel.

However, from the current literature on MG methods, it appears that relaxation operations for different grid layers cannot be performed simultaneously. If this shortcoming can be overcome, then more parallelism will be achieved. That means we can not only obtain parallelism in the space domain but also in the frequency domain.

In this Chapter, I shall propose a new multigrid scheme, called the concurrent multigrid method (CMG), which achieves the above mentioned goal. Before discussing the CMG method, it is useful to explain why MG methods do not allow parallelism on different grid levels.

Usually, there are two ways to analyze MG methods : *model problem analysis* and *local Fourier analysis* [24]. Unfortunately, both have their own disadvantages. The model problem analysis, although rigorous, can be applied directly to a limited class of problems. The local Fourier analysis, based on idealized assumptions, is more intuitive but not as rigorous. Undoubtedly, there is still a gap between the theoretical and experimental aspects of MG methods. More formalism is required and further theoretical explorations need to be made. In the following, the local Fourier analysis approach will be adopted.

Local Fourier analysis was used by Brandt [23] to analyze the property of the local relaxation operator. Consider one example - Jacobi relaxation of the model problem in the error space,

$$e_{i,j}^{(n+1)} = \frac{1}{4} \left(e_{i-1,j}^{(n)} + e_{i+1,j}^{(n)} + e_{i,j-1}^{(n)} + e_{i,j+1}^{(n)} \right) \quad n \ge 0 \quad . \tag{5.1}$$

As in Chapter 3, the local relaxation operator is

$$J = \frac{E_1 + E_1^{-1} + E_2 + E_2^{-1}}{4}. \tag{5.2}$$

We may choose the sinusoidal functions $e^{i(k_1x_1+k_2x_2)}$ as the basis functions, where $e^{i(k_1x_1+k_2x_2)}$ is the eigenfunction of the local relaxation operator J with eigenvalue $\frac{\cos k_1h + \cos k_2h}{2}$. Therefore, the Jacobi relaxation procedure is equivalent to a low pass filtering operation on the error. In fact, the analytic procedure of the LASOR algorithm in Chapter 3 is another example of local Fourier analysis.

In Chapter 3, we assumed the grid size h was fixed, so that the error smoothing rate was determined by k_1 and k_2 only. Now, we allow different grid sizes, i. e., h becomes a variable, so that the error smoothing rate is determined by the products of $k_1 h$ and $k_2 h$. This additional degree of freedom helps us in getting faster convergence rates. For very small k_1 and k_2, i.e., low frequency components, coarser grids can be used such that the products of $k_1 h$ and $k_2 h$ become larger and the spectral radius is reduced while the discretization error remains the same.

However, we should not neglect that both low and high frequency error components occur. Unless there is some splitting scheme which separates these into different frequency bands, the multigrid idea cannot work properly. One way to do the separation is very straightforward. Since high frequency errors are smoothed much faster than low frequency errors, we may perform relaxation several times on a fine grid until the convergence rate is slowed, which implies that the low frequencies become dominant. Then, a simple grid transfer is applied, say, by directly mapping the values from the finer grid to the coarser grid.

In a centralized sequential computing machine, this seems to be a very simple and effective choice, since it does not require too much overhead except the testing procedure, which examines whether the grid transfer criterion has been met. However, in parallel computing machines, this kind of splitting scheme wastes a lot of time in waiting for the results of relaxation on one specific grid. Therefore, other splitting schemes are needed, which will make the CMG method different from other MG methods.

In addition to the separation of low and high frequencies, there is another issue worth our attention. That is, the relaxation scheme may become more effective when the error function to be smoothed is confined to a small frequency band. This will be discussed in detail in Section 5.2.3.

5.2 Concurrent Multigrid Method (CMG)

The concurrent multigrid method (CMG), like other MG methods, consists of three components: error smoothing, grid transfer, and iteration between different grid levels. The most distinctive feature of the CMG method is that the grid transfer operator should be carefully chosen such that it may separate the low and high frequencies as much as possible.

5.2.1 Analysis and Synthesis of Grid Transfer Operator

There are two kinds of grid transfer operations: the finer grid can be transferred to the coarser grid and the coarser grid can be transferred to the finer grid. The former is called *projection* and the latter is called *interpolation*.

There are many ways to construct the multiple grid levels. Two important parameters are the grid size and the grid orientation. For example, let the spacing of the finest grid, also called the first level, be h, then the grid size of second level may be $2h$ or $3h$, the grid size of the third level may be $4h$ or $9h$, and so on. The second level may have the same orientation as the first level or may be rotated by some angle with respect to the first level. In order to simplify the following discussion, I only choose a simple, but typical, case here: the grid size of the mth level is $2^{m-1}h$ and the orientation of all grid levels is the same. In addition, the coarser grid points coincide with grid points of the finer level, i. e., there is no shifting between any two levels. If we denote the set of all grid points of the mth level by $G_{2^{m-1}h}$, then it is easy to see that $G_h \supset G_{2h} \supset G_{4h} \supset \cdots \supset G_{2^m h} \supset \cdots$.

Since the grid transfer between the grids G_h and G_{2h} has the same structure as that between $G_{2^{m-1}h}$ and $G_{2^m h}$, we may concentrate on the grid transfer between G_h and G_{2h}. The projection operator will be discussed first, and then the interpolation operator.

The projection from G_h to G_{2h}, denoted by $I_h^{2h} : G_h \rightarrow G_{2h}$, can be defined as

$$u_{2h}(x_1,x_2) = I_h^{2h}\, u_h\,(\vec{x}\,)\,|_{\vec{x}\,\in\,G_h} \tag{5.3}$$
$$= \sum_{-P\,\le\,t_1,t_2\,\le\,P} a(t_1,t_2)\, u_h(x_1+t_1h\,,x_2+t_{2d}h) \quad ,$$

where $(\,x_1\,,x_2\,) \in G_{2h}$, P is a positive integer, and $u_h\,(\,x_1\,,x_2\,)$ and $u_{2h}\,(\,x_1\,,x_2\,)$ are grid functions defined on G_h and G_{2h} separately. Take an example, if $P\ =\ 1$, the projection operator I_h^{2h} can be conveniently represented by a 2-D coefficient matrix

$$I_h^{2h} = \begin{bmatrix} a(-1,\,1) & a(\,0,\,1) & a(\,1,\,1) \\ a(-1,\,0) & a(\,0,\,0) & a(\,1,\,0) \\ a(-1,-1) & a(\,0,-1) & a(\,1,-1) \end{bmatrix}_h^{2h} \quad , \tag{5.4}$$

or in difference operator form

$$I_h^{2h} = \ a(0,0) + a(1,0)\,E_1 + a(-1,0)\,E_1^{-1} + a(0,1)\,E_2 + a(0,-1)\,E_2^{-1} \tag{5.5}$$
$$+\ a(-1,1)\,E_1^{-1}\,E_2 +\ a(1,1)\,E_1\,E_2 + a(-1,-1)\,E_1^{-1}\,E_2^{-1} + a(1,-1)\,E_1\,E_2^{-1}\ .$$

The properties of the projection operator I_h^{2h} are completely determined by the coefficients $a\ (\,t_1\,,t_2\,)$'s. The question of choosing a set of coefficients to separate the low and high frequency components is equivalent to a 2-D low pass FIR (finite impulse response) filter design [25]. Since 2-D FIR filter design has been studied for a long time in the digital signal processing field, we may apply the results of this investigation to our work directly.

Before going into the *design* problem, let us first concentrate on the *analysis* aspect. Fourier analysis is a convenient tool to analyze the the projection operator I_h^{2h}. We still use the complex sinusoidal function $e^{\,i(k_1x_1+k_2x_2)}$ as input and find out the corresponding eigenvalue of the projection operator. That is

$$I_h^{2h}\, e^{i(k_1x_1+k_2x_2)} = \sum_{-P\le t_1,t_2\le P} a\,(t_1,t_2)\, e^{i[k_1(x_1+t_1h)+k_2(x_2+t_2h)]} \tag{5.6}$$

$$= [\sum_{-P\le t_1,t_2\le P} a\,(t_1,t_2)\, e^{i(k_1t_1h+k_2t_2h)}\,]\, e^{i(k_1x_1+k_2x_2)} \quad ,$$

and therefore the eigenvalue of I_h^{2h} associated to $e^{i(k_1x_1+k_2x_2)}$ is

$\sum_{-P\le t_1,t_2\le P} a\,(t_1,t_2)\, e^{i(k_1t_1h+k_2t_2h)}$, which is also known as the *frequency response* of the operator I_h^{2h}. In most cases, there are some other constraints on the integer P and the coefficients $a(t_1,t_2)$'s so that the eigenvalue can be simplified. For example, assuming that $P = 1$ and the coefficients are symmetric, i.e.,

$$a\,(0\,,0) = a \quad , \tag{5.7.a}$$

$$a\,(1\,,0) = a\,(-1\,,0) = \frac{b}{2}\,, \quad a\,(0\,,1) = a\,(0\,,-1) = \frac{c}{2}\,, \tag{5.7.b}$$

$$a\,(1\,,1) = a\,(1\,,-1) = a\,(-1\,,1) = a\,(-1\,,-1) = \frac{d}{4} \quad . \tag{5.7.c}$$

and using (5.7.a) - (5.7.c), the equation (5.6) can be rewritten as

$$I_h^{2h}\, e^{i(k_1x_1+k_2x_2)} = [a + b\cos k_1h + c\cos k_2h + d\cos k_1h \cos k_2h\ ]\, e^{i(k_1x_1+k_2x_2)}. \tag{5.8}$$

and the eigenvalue becomes $a + b\cos k_1h + c\cos k_2h + d\cos k_1h\cos k_2h$.

One commonly used projection operator in MG methods is the *full weighting* (FW) projection, defined as

$$I_{h,FW}^{2h} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_h^{2h} \quad , \tag{5.9}$$

and from (5.7) and (5.8) we know the eigenvalue of $I_{h,FW}^{2h}$ associated to $e^{i(k_1x_1+k_2x_2)}$ is

$$\lambda_{FW}(k_1,k_2) = \frac{1}{4}\,[1+\cos k_1h + \cos k_2h + \cos(k_1h)\ cos\,(k_2h)\ ] \quad . \tag{5.10}$$

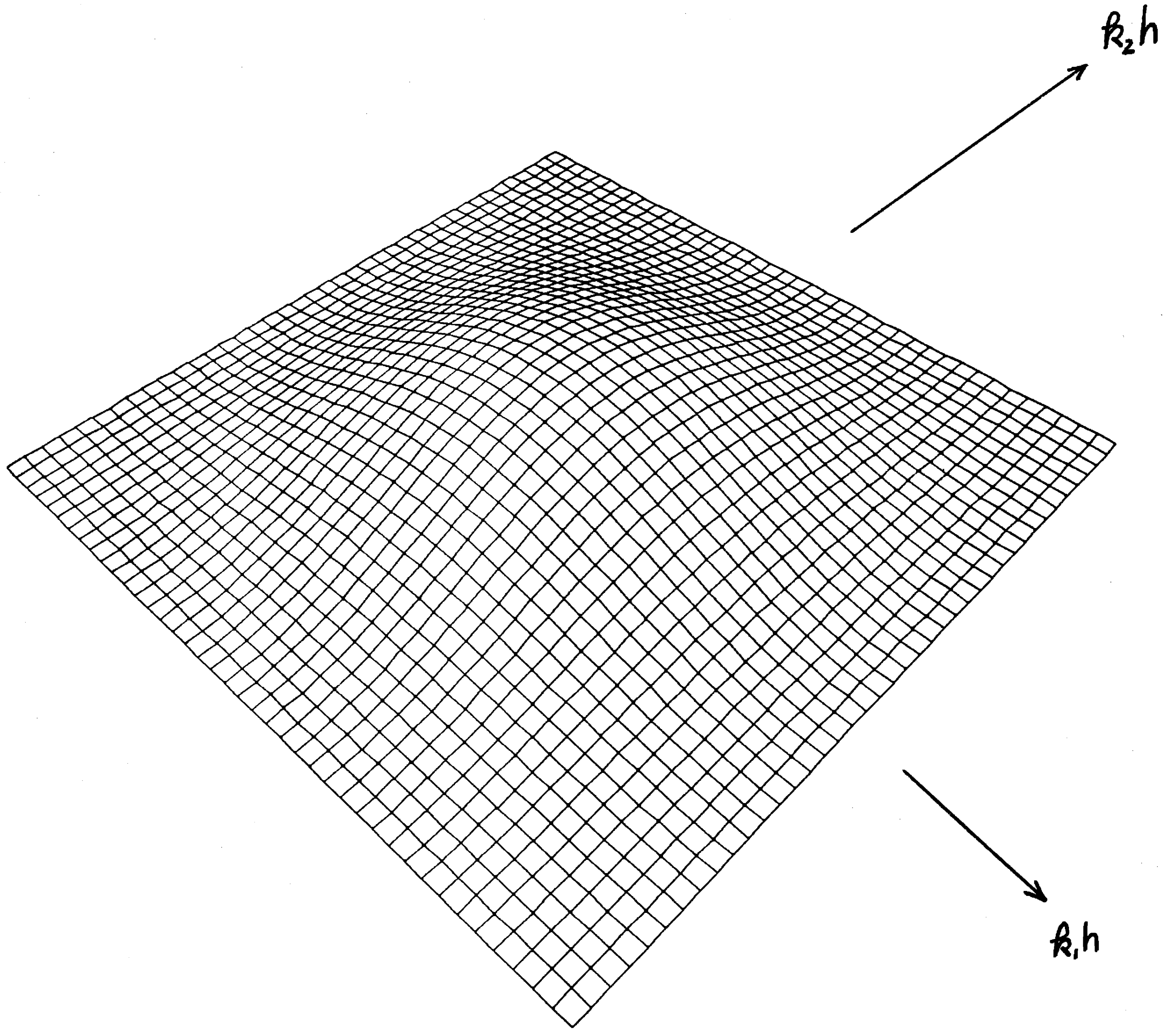

Fig 5.1 Frequency Response Function of the Full-Weighting Projection Operator

The full weighting projection has a low pass filtering characteristic, and the frequency response function $\lambda_{FW}\,(k_1,k_2)$ is drawn in Figure 5.1. However, this projection operator is not necessarily the best choice for the purpose of separating the low and high frequencies. To find a filter with better characteristics, we now discuss the filter design problem.

The first step in filter design is the *filter specification*. In Chapter 3, it has been shown that the relaxation operation behaves like a notch filter which filters out the middle error frequencies most effectively. Therefore, the projection operator at the m th level, $I_{2^{m-1}h}^{2^m h} : G_{2^{m-1}h} \to G_{2^m h}$, should be designed such that the low frequency components are projected to the coarser $(m+1)$ th level, while the middle frequency components remain at the m th level.

To state it more rigorously, let us use k_1 and k_2 to represent the spatial frequencies without discretization and $K_{1,m}$ and $K_{2,m}$ to represent the spatial frequencies at the m th level, corresponding to a spacing $2^{m-1}h$. Then, the relation between $K_{1,m}$, $K_{2,m}$ and k_1, k_2 can be shown [18] to be

$$K_{1,m} = k_1\, 2^{m-1}\, h \;\;,\;\; -\pi \le K_{1,m} \le \pi \;\;,\;\; m = 1\,,2\,,3\,,\,.\,.\,.\,.\,,\, M \;,\qquad (5.11.a)$$

$$K_{2,m} = k_2\, 2^{m-1}\, h \;\;,\;\; -\pi \le K_{2,m} \le \pi \;\;,\;\; m = 1\,,2\,,3\,,\,.\,.\,.\,.\,,\, M \;,\qquad (5.11.b)$$

where M is the index of the highest level. We also know that $K_{1,m}$ and $K_{2,m}$ are periodic functions with periods 2π for all m's. From (5.11), it is easy to find the relation satisfied by frequencies at two successive levels,

$$K_{1,m+1} = k_1\, 2^m\, h = 2\, K_{1,m} \;,\quad m = 1\,,2\,,\,\cdots\, M-1 \;,\qquad (5.12.a)$$

$$K_{2,m+1} = k_2 2^m h = 2 K_{2,m} , \quad m = 1 , 2 , \cdots M-1 . \qquad (5.12.a)$$

Define K_m to be the radius from the origin in the $K_{1,m}$-$K_{2,m}$ plane, then we have

$$K_m = \sqrt{K_{1,m}^2 + K_{2,m}^2} , \qquad m = 1 , 2 , 3 , \ldots , M . \qquad (5.13)$$

As a consequence of equations (5.12) and (5.13), the circular region $0 \leq K_m \leq \frac{\pi}{2}$ in the $K_{1,m}$-$K_{2,m}$ plane is equivalent to the larger circular region $0 \leq K_{m+1} \leq \pi$ in the $K_{1,m+1}$-$K_{2,m+1}$ plane.

If we choose our ideal low pass filter at the mth level to be

$$\begin{aligned} \lambda (K_m) &= 1 \qquad 0 \leq K_m \leq \frac{\pi}{2} , \\ \lambda (K_m) &= 0 \qquad \textit{elsewhere} , \qquad m = 1 , 2 , 3 , \ldots , M-1 , \end{aligned} \qquad (5.14)$$

then at the mth level, only the error frequency components in the region $0 \leq K_m \leq \pi$ can be provided from the $(m-1)$th level by the projection operator $I_{2^{m-2}h}^{2^{m-1}h}$. And, furthermore, the lower frequencies $0 \leq K_m \leq \frac{\pi}{2}$ should be projected to the $(m+1)$th level by the projection operator $I_{2^{m-1}h}^{2^m h}$. Therefore, the remaining portion at the mth level is the donut-shaped region $\frac{\pi}{2} \leq K_m \leq \pi$, which can be filtered out very efficiently using the mth level relaxation operator.

In summary, assume every mth level projection operator has the ideal low pass filtering characteristic as indicated in equation (5.14), then we may cascade all projection operators together as shown in Figure 5.2 and separate the frequencies in the finest grid, $K_{1,1}$'s and $K_{2,1}$'s, into several frequency bands. That is,

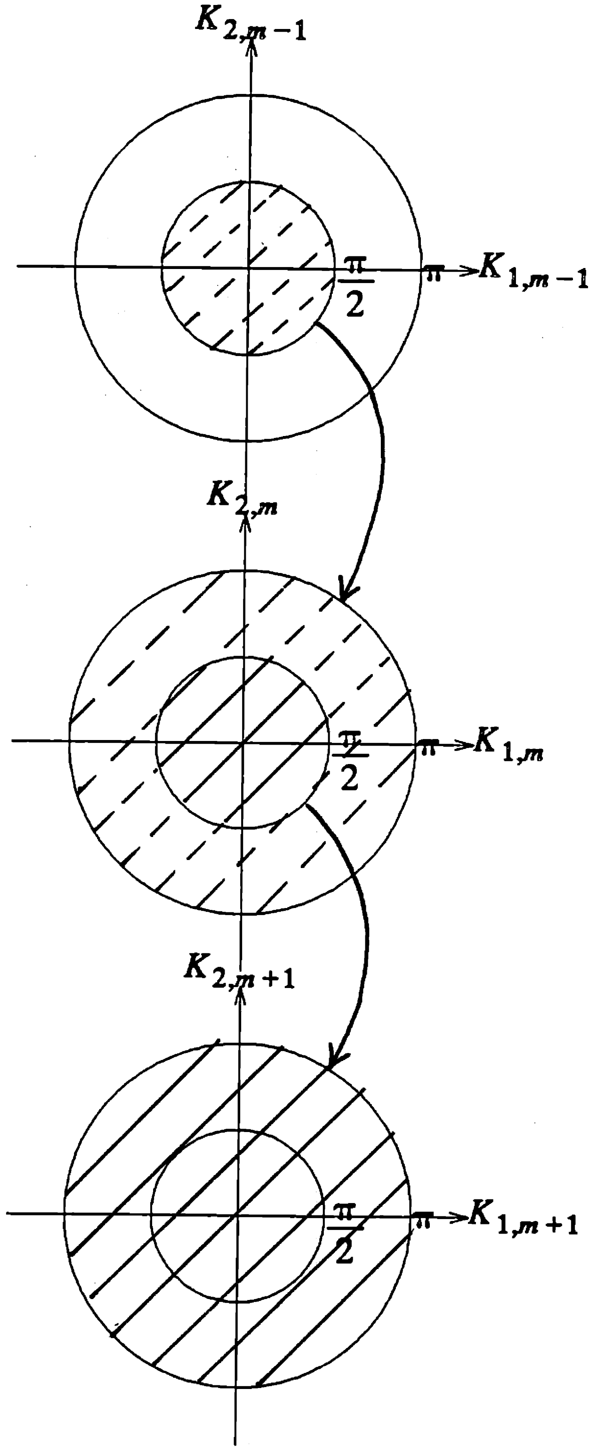

Fig 5.2 Frequency Bands of the $m-1$th, mth, and $m+1$th Levels

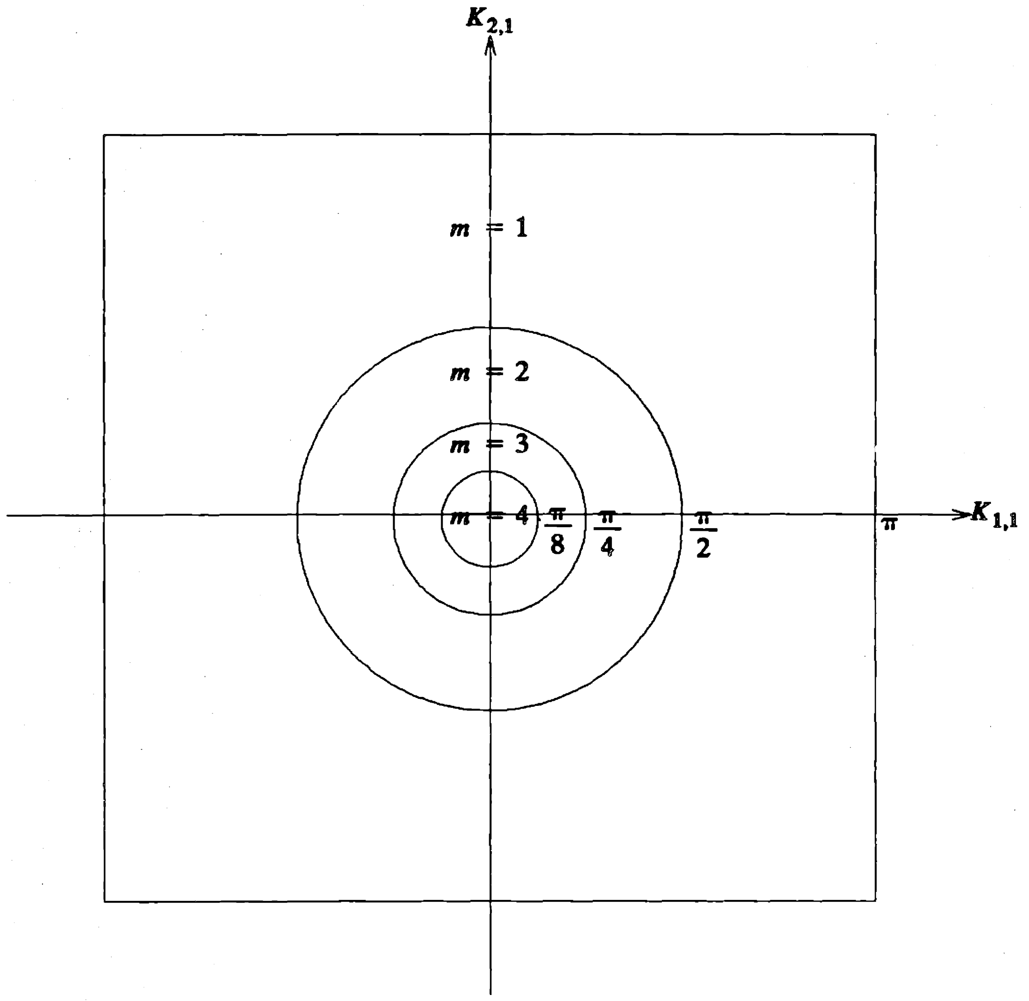

Fig 5.3.a Frequency Bands of Multiple Levels Represented in the $K_{1,1} - K_{2,1}$ Plane ($M = 4$)

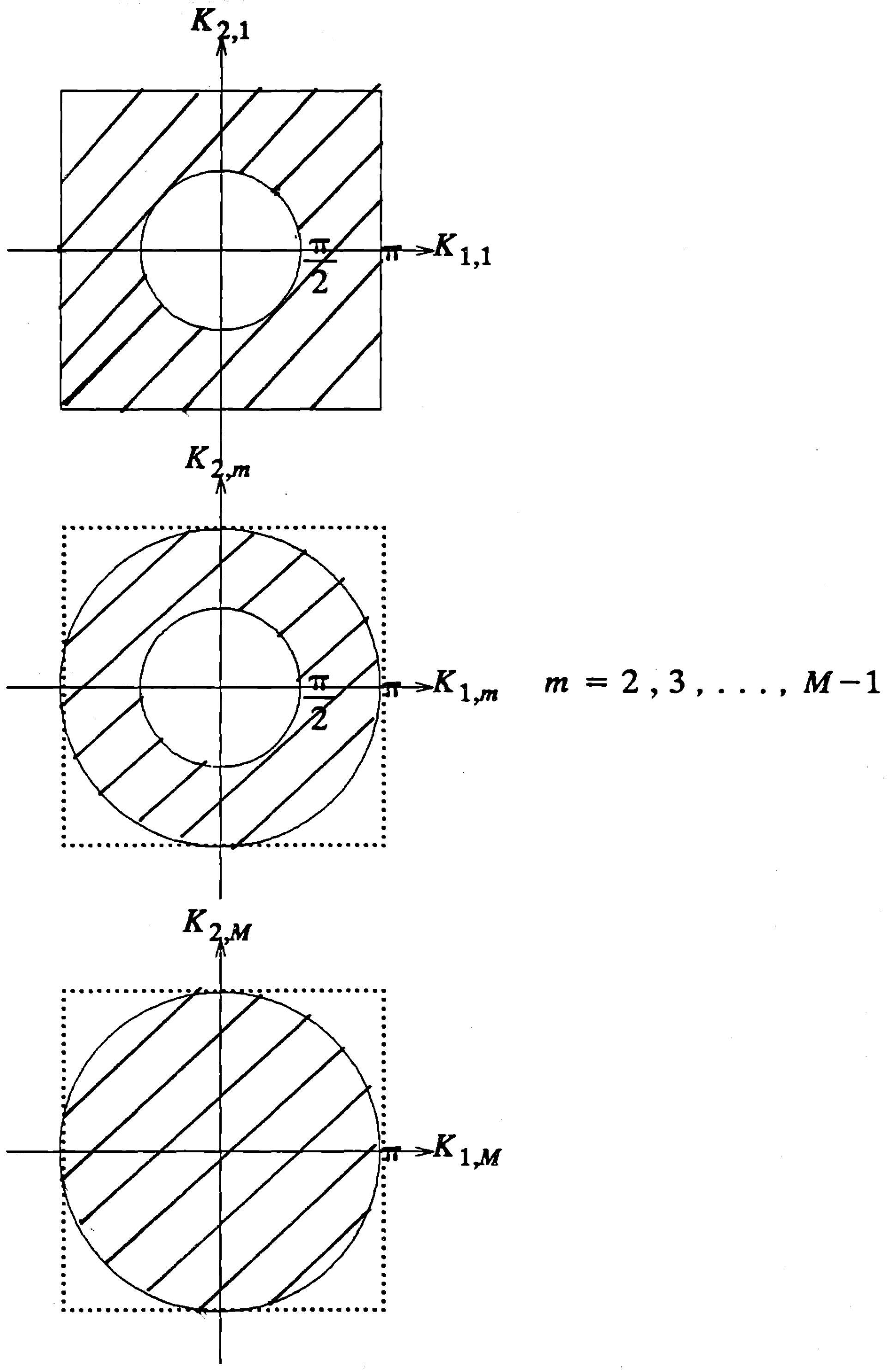

Fig 5.3.b Frequency Bands of Multiple Levels Represented in the $K_{1,m} - K_{2,m}$ Plane

$$\frac{\pi}{2} \le K_1 \text{ and } -\pi \le K_{1,1}, K_{2,1} \le \pi \qquad \textit{band} : 1 , \tag{5.15.a}$$

$$\frac{\pi}{2^m} \le K_1 \le \frac{\pi}{2^{m-1}} \qquad \textit{bands} : 2 , 3 , \ldots , M-1 , \tag{5.15.b}$$

$$K_1 \le \frac{\pi}{2^{M-1}} \qquad \textit{band} : M . \tag{5.15.c}$$

These different bands can be mapped into $K_{1,m} - K_{2,m}$ plane. It appears that they are lying in the regions which can be easily smoothed out with relaxation in these different grids, i.e.,

$$\frac{\pi}{2} \le K_1 \text{ and } -\pi \le K_{1,1}, K_{2,1} \le \pi \qquad m = 1 , \tag{5.16.a}$$

$$\frac{\pi}{2} \le K_m \le \pi \qquad m = 2 , 3 , \ldots , M-1 , \tag{5.16.b}$$

$$K_m \le \pi \qquad m = M . \tag{5.16.c}$$

Equations (5.15) and (5.16) are illustrated in Figures 5.3.a and 5.3.b.

Although the characteristic of an ideal low pass filter is given by equation (5.14), in practice, we are not able to construct such a filter. Instead, the following specifications are used to approximate the low pass property,

$$\begin{array}{lll} 1 - \delta_p \le \lambda (K_m) \le 1 + \delta_p & 0 \le K_m \le \omega_p & \textit{pass band} \\ \delta_s \le \lambda (K_m) \le 1 - \delta_p & \omega_p \le K_m \le \omega_s & \textit{transition band} \\ -\delta_s \le \lambda (K_m) \le \delta_s & \omega_s \le K_m & \textit{stop band} \end{array} , \tag{5.17}$$

which is shown in Figure 5.4. For a fixed set of parameters δ_p, δ_s, ω_p, and ω_s, the task of finding a corresponding P and $a(t_1,t_2)$'s is known as a *2-D FIR filter design* mentioned before. The details of the design are described in [25].

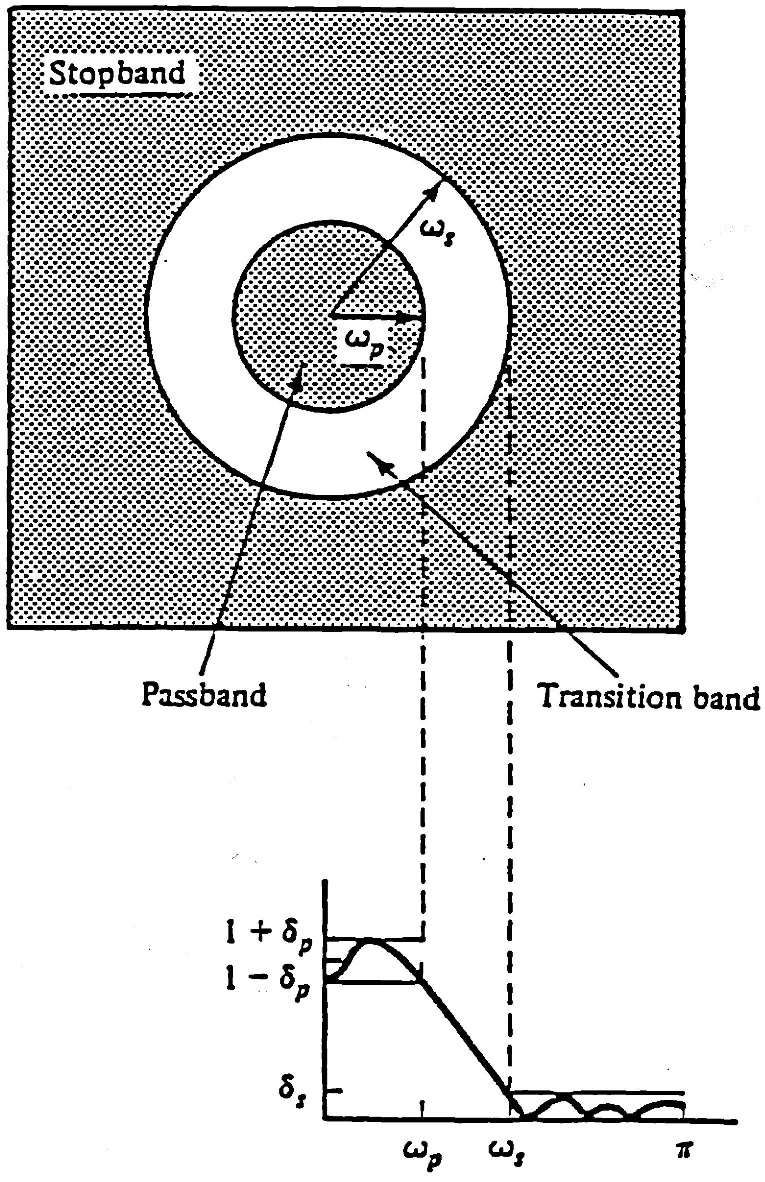

Fig 5.4 Specification of 2-D Low Pass Filter

Here, I would like to point out that the performance of the designed filter heavily depends on its size, $(2P+1) \times (2P+1)$ as well as suitable coefficients. A better low pass filter usually requires larger P, which increases the communication cost in performing the projection operation. However, this constraint is not very severe for two reasons. First, the projection only happens when a grid transfer is required, which occurs less frequently than basic relaxation steps. Secondly, usually $P = 4$ or 5 gives a satisfactory result, so that the communication cost in this case is only a constant multiple (four or five times) of that of the nearest neighborhood communication scheme.

After the discussion of the projection operator, we come to the second grid transfer operation : interpolation. The main issue here is how to preserve the low frequencies obtained in the coarser grid faithfully in the finer grid. Once this has been done, these low frequency components can be combined with the high frequency components obtained in the finer grid level to provide a more complete solution.

The interpolation operators from G_{2h} to G_h, represented by $I_{2h}^h : G_{2h} \rightarrow G_h$, can be classified into four classes. As shown in Figure 5.5, the grid points are partitioned into four groups: A, B, C, and D. In position A, the coarse grid points coincide with the finer grid points, so we have

position A interpolation :

$$u_h\,(x_1\,,x_2)\,|_{(x_1,x_2)\in G_{2h}} = I_{2h,A}^h\,u_{2h}\,(\vec{x}\,)\,|_{\vec{x}\in G_{2h}} = u_{2h}\,(x_1\,,x_2) \quad .(5.18)$$

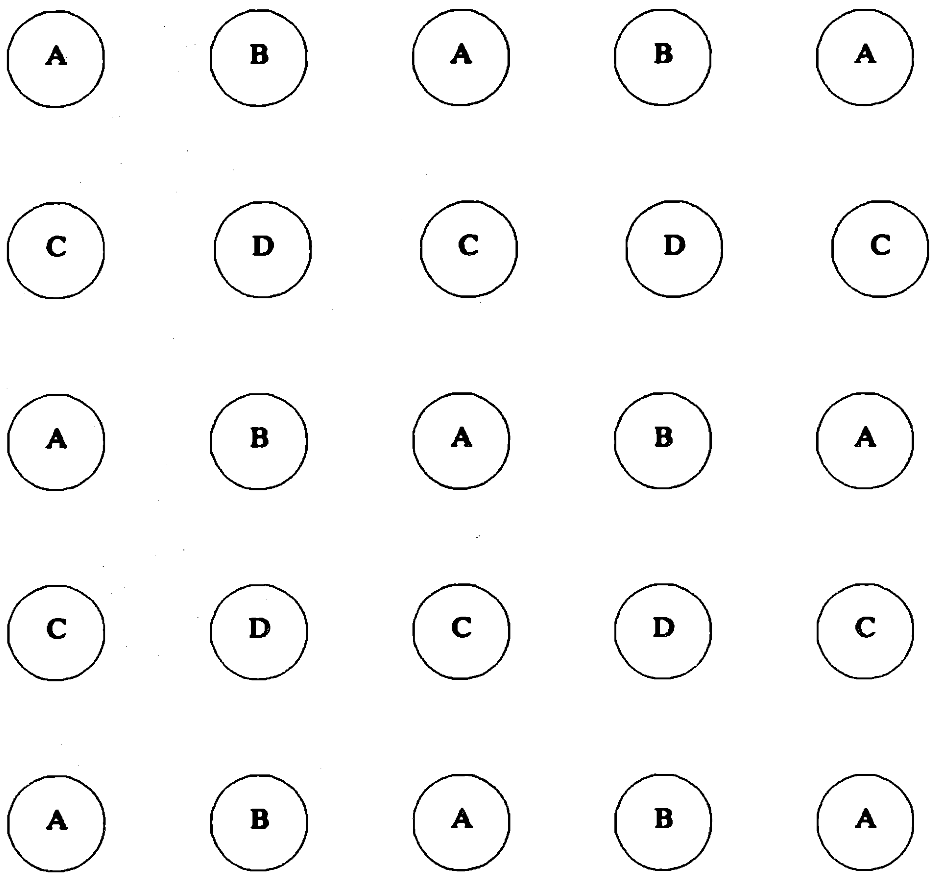

Fig 5.5 Grid Points for Different Interpolation Operators

The interpolation $I_{2h,A}^{h}$ is in fact the identity operator with the eigenvalue one for all frequencies, so it behaves like an all pass filter, which certainly preserves the low frequency components.

In positions B and C, let the interpolations be defined as

position B interpolation :

$$u_h(x_1 + h, x_2)\,|_{(x_1, x_2) \in G_{2h}} = I_{2h,B}^{h}\, u_{2h}(\vec{x})\,|_{\vec{x} \in G_{2h}} \tag{5.19}$$
$$= a(1,0)\, u_{2h}(x_1, x_2) + a(-1,0)\, u_{2h}(x_1 + 2h, x_2) \;,$$

position C interpolation :

$$u_h(x_1, x_2 + h)\,|_{(x_1, x_2) \in G_{2h}} = I_{2h,C}^{h}\, u_{2h}(\vec{x})\,|_{\vec{x} \in G_{2h}} \tag{5.20}$$
$$= a(0,-1)\, u_{2h}(x_1, x_2 + 2h) + a(0,1)\, u_{2h}(x_1, x_2) \quad .$$

If we choose $a(1,0) = a(-1,0) = a(0,1) = a(0,-1) = \frac{1}{2}$ and use Fourier analysis again, it is easy to find that the eigenvalues of $I_{2h,B}^{h}$ and $I_{2h,C}^{h}$ are $cos(k_1 h)$ and $cos(k_2 h)$ separately. This tells us that these two simple interpolation operators indeed do not influence the low frequency components too much. Of course, better low pass performance can be achieved if we increase the size of the interpolation operator and pay higher communication cost.

Last, assume the interpolation in position D is

position D interpolation :

$$u_h(x_1 + h, x_2 + h)\,|_{(x_1, x_2) \in G_{2h}} = I_{2h,D}^{h}\, u_{2h}(\vec{x})\,|_{\vec{x} \in G_{2h}} \tag{5.21}$$
$$= a(1,-1)\, u_{2h}(x_1 - h, x_2 + h) + a(-1,-1)\, u_{2h}(x_1 + h, x_2 + h)$$
$$+ a(-1,1)\, u_{2h}(x_1 + h, x_2 - h) + a(1,1)\, u_{2h}(x_1 - h, x_2 - h) \quad .$$

A good choice for these coefficients is

$$a(1,-1) = a(-1,-1) = a(-1,1) = a(1,1) = \frac{1}{4} . \qquad (5.22)$$

Under the assumptions of (5.21) and (5.22), the eigenvalue of $I_{2h,D}^{h}$ is $\frac{cos(k_1 h) + cos(k_2 h)}{2}$; therefore, $I_{2h,D}^{h}$ looks like a low pass filter.

There is a simplified notation to represent these four operations together, i.e.,

$$I_{2h}^{h} = \left]\begin{matrix} a(-1,1) & a(0,1) & a(1,1) \\ a(-1,0) & a(0,0) & a(1,0) \\ a(-1,-1) & a(0,-1) & a(1,-1) \end{matrix}\right[_{2h}^{h} = \left]\begin{matrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{matrix}\right[_{2h}^{h} \qquad (5.23)$$

which means the weighted contributions from $u_{2h}(x_1, x_2)$ to its neighboring points. This 3×3 interpolation operator is usually used in most current MG methods. As in the projection case, more general $(2P+1) \times (2P+1)$ interpolation operators can be designed such that the low frequency components of the coarser grid can be less influenced by the interpolation operation, at the expense of increased communication cost.

5.2.2 Motivation for the Concurrent Multigrid Methods (CMG) - 1-D case

Consider a discretized equation on a single grid with spacing h,

$$\hat{L}_{h,i,j}\, \hat{u}_{h,i,j} = \hat{f}_{h,i,j} \qquad (ih\ , jh) \in G_h \qquad (5.24.a)$$

$$\hat{B}_{h,i,j}\, \hat{u}_{h,i,j} = \hat{g}_{h,i,j} \qquad (ih\ , jh) \in \Gamma_h \qquad (5.24.b)$$

The goal of the CMG method is to obtain a system of discretized equations on multiple grids with spacings $2^{m-1}h\ , m = 1, 2, \ldots, M-1$,

$$L_{2^{m-1}h,i,j}\, u_{2^{m-1}h,i,j} = f_{2^{m-1}h,i,j} \qquad (ih\ , jh) \in G_{2^{m-1}h}\ , \qquad (5.25.a)$$

$$B_{2^{m-1}h,i,j}\, u_{2^{m-1}h,i,j} = g_{2^{m-1}h,i,j} \qquad (ih\ , jh) \in \Gamma_{2^{m-1}h}\ , \qquad (5.25.b)$$

so that these equations can be solved simultaneously and the low and high frequencies are separated in different grids. We also require that the solutions on different grids should be able to be combined in an easy way to become the single grid solution $\hat{u}_h$, i.e.,

$$\hat{u}_h = C\ (\ u_h\ , u_{2h}\ , \dots\ , u_{2^{M-1}h}\ )\ , \qquad (5.26)$$

where C represents some kind of combination.

For a given multigrid structure, i.e., for a fixed number M of grids, fixed $G_{2^{m-1}h}$'s, and $\Gamma_{2^{m-1}h}$'s, the CMG includes three parts:

(1) how to choose the operators $L_{2^{m-1}h,i,j}$, $B_{2^{m-1}h,i,j}$ and the driving functions $f_{2^{m-1}h,i,j}$, $g_{2^{m-1}h,i,j}$ for each grid,

(2) how to get the solution $u_{2^{m-1}h,i,j}$ on each grid,

(3) how to combine these solutions together.

In order to understand these issues, let us use a simple 1-D problem to illustrate the basic idea of the CMG method.

Consider a 1-D second order ordinary differential equation (ODE) with homogeneous boundary conditions in the region [0 , 1],

$$a\ (x)\ \frac{d^2\, u\ (x)}{d\ x^2} + b\ (x)\ \frac{d\ u\ (x)}{d\ x} + c\ (x)\, u\ (x) = d\ (x)\ , \qquad (5.27.a)$$

$$u\ (0) = 0\ , \qquad u'\ (1) = 0\ . \qquad (5.27.b)$$

We may use Fourier series to expand all functions given above and get

$$\left(\sum_{n=-\infty}^{\infty} a_n e^{i2\pi nx}\right)\left(\sum_{n=-\infty}^{\infty} -4\pi^2 n^2 u_n e^{i2\pi nx}\right) + \left(\sum_{n=-\infty}^{\infty} b_n e^{i2\pi nx}\right)\left(\sum_{n=-\infty}^{\infty} i2\pi n u_n e^{i2\pi nx}\right)$$

$$+ \left(\sum_{n=-\infty}^{\infty} c_n e^{i2\pi nx}\right)\left(\sum_{n=-\infty}^{\infty} u_n e^{i2\pi nx}\right) = \sum_{n=-\infty}^{\infty} d_n e^{i2\pi nx} \qquad (5.28.a)$$

and the boundary conditions become

$$\sum_{n=-\infty}^{\infty} u_n = 0\ , \qquad \sum_{n=-\infty}^{\infty} i2\pi n u_n e^{i2\pi n} = 0\ . \qquad (5.28.b)$$

Comparing the coefficients of (5.28.a), we obtain

$$\sum_{q=-\infty}^{\infty} -4\pi^2(n-q)^2 a_q u_{n-q} + \sum_{q=-\infty}^{\infty} i2\pi(n-q) b_q u_{n-q}$$

$$+ \sum_{q=-\infty}^{\infty} c_q u_{n-q} = d_n \ \ , n = 0,\pm 1,\pm 2,\ldots\ . \qquad (5.29)$$

The equations (5.28.b) and (5.29) constitute an infinite dimensional system which needs to be solved for the infinite number of unknowns u_n. At a first glance, the transformation of the original space domain problem into a new frequency domain problem does not appear to be helpful. But this is *not* true.

We may rewrite equations (5.29) and (5.28.b) as

$$\sum_{|q|\le P} -4\pi^2(n-q)^2 a_q u_{n-q} + \sum_{|q|\le P} i2\pi(n-q) b_q u_{n-q} + \sum_{|q|\le P} c_q u_{n-q}$$

$$+ \sum_{|q|>P} -4\pi^2(n-q)^2 a_q u_{n-q} + \sum_{|q|>P} i2\pi(n-q) b_q u_{n-q} + \sum_{|q|>P} c_q u_{n-q} \qquad (5.30.a)$$

$$= d_n \ \ , n = 0,\pm 1,\pm 2,\ldots,$$

and

$$\sum_{|q|\le Q} u_q + \sum_{|q|>Q} u_q = 0\ , \qquad \sum_{|q|\le Q} q u_q + \sum_{|q|>Q} q u_q = 0\ , \qquad (5.30.b)$$

where P and Q are some positive integers. It will be shown below that the equations

(5.30.a) and (5.30.b) can be simplified under some assumptions.

Although Fourier series may contain an infinite number of terms, the high frequency components are usually so small that we may neglect them and still obtain some reasonably good approximations. If the coefficient functions $a(x)$, $b(x)$, and $c(x)$ are smooth, their frequency bands should be quite narrow so that beyond some range, say $|q|>P$, the Fourier components of these functions become negligible. Therefore, we may separate the convolution terms in equation (5.29) into two parts, shown in equation (5.30.a), and consider the second part very small compared with the first part. Similarly, the equations of boundary conditions can also be separated into two parts and the second part is negligible. Notice that since the coefficient functions are usually more smooth than the solution function, we have $P < Q$.

Therefore, we may approximate equations (5.30) by

$$\sum_{|q|\le P} -4\pi^2(n-q)^2 a_q u_{n-q} + \sum_{|q|\le P} i2\pi(n-q) b_q u_{n-q} + \sum_{|q|\le P} c_q u_{n-q} = d_n \quad , n = 0,\pm 1,\pm 2,\ldots, \tag{5.31.a}$$

and

$$\sum_{|q|\le Q} u_q = 0 \,, \qquad \sum_{|q|\le Q} q u_q = 0 \,, \tag{5.31.b}$$

under the assumptions $a_q = b_q = c_q = 0 \,, |q| > P$ and $u_q = 0 \,, |q| > Q$. Because there are $(2Q + 1)$ unknowns u_q's in the above system, we need $(2Q + 1)$ equations to obtain a set of unique solutions. There are already two constraints for the boundary conditions, and we can have $(2Q - 1)$ more equations by choosing the equations (5.31.a) with the index n , $|n| \leq Q - 1$. The assumption that $u_q = 0$, $|q| > Q$ can be interpreted as *boundary conditions in the frequency domain.* The equations (5.31.a) originally include $2(Q+P)+1$ variables, u_q for $|q| \leq Q + P$. Therefore, the boundary conditions required in the transformed domain are $u_q = 0$ for $Q < |q| \leq Q + P$.

In summary, we have the following system in the frequency domain,

$$\sum_{|q| \leq P} -4\pi^2(n-q)^2 a_q u_{n-q} + \sum_{|q| \leq P} i2\pi(n-q) b_q u_{n-q} +$$

$$\sum_{|q| \leq P} c_q u_{n-q} = d_n \quad , n = 0, \pm 1, \pm 2, \ldots, \pm(Q-1), \tag{5.32.a}$$

$$\sum_{|q| \leq Q} u_q = 0 \, , \qquad \sum_{|q| \leq Q} q u_q = 0 \, , \tag{5.32.b}$$

with the frequency domain boundary conditions

$$u_q = 0 \quad , Q < |q| \leq Q + P \quad . \tag{5.32.c}$$

In all previous Chapters, we are focused on parallel algorithms for solving systems of equations arising from space domain discretization. Now, given the above equations in the frequency domain, it seems interesting to look for a parallel algorithm for solving them too. If equations (5.32) are written in matrix form, the matrix is almost a banded matrix with bandwidth $(2P + 1)$ except for the last two rows, which represent the space domain boundary conditions.

As mentioned before, for a smooth operator, the coefficient functions contains only very few significant Fourier expansion terms, so it becomes a narrow band matrix. In an extreme case, where the PDE operator has constant coefficients, equation (5.32.a) can be reduced to be

$$-4\pi^2 q^2 a_0 u_n + i2\pi q b_0 u_n + c_0 u_n = d_n \quad , n = 0, \pm 1, \pm 2, \ldots, \pm(Q-1) \ . \quad (5.33)$$

The band matrix becomes a diagonal matrix. Since there is no coupling between the different frequencies, they can be solved in parallel and the solutions are

$$u_n = \frac{d_n}{-4\pi^2 n^2 a_0 + i2\pi n b_0 + c_0} \ , \quad n = 0, \pm 1, \pm 2, \ldots, \pm(Q-1) \ . \quad (5.34)$$

Finally, u_Q and u_{-Q} can be determined by solving (5.31.b). One feature of the above system is that these frequency components, indicated in (5.34), are not a function of Q. They are exactly the same as those found without cutting out any high frequency components. If we let Q go larger and larger, we will be able to calculate all possible u_n's, which converges to zero as $O\left(\frac{1}{n^2}\right)$. As a consequence, the space domain boundary condition equations (5.32.b) are not important any longer. Since the Fourier analysis approach separates the coupling of different frequencies entirely, it is commonly used in solving 1-D linear constant coefficient PDEs with homogeneous boundary conditions, and in analyzing the linear time-invariant or space-invariant systems. On the other hand, because of the coupling effect shown in (5.32.a), the result is not as simple for varying coefficient PDEs. In order to get a parallel computational algorithm for the general system (5.32), we can try the following system decomposition scheme. Then, let each system be solved by a single computer. Find a sequence of positive integers $Q_1, Q_2, \ldots, Q_M$, which satisfy

$$Q_1 < Q_2 < \cdots < Q_M = Q \text{, and } P < \mid Q_{i+1} - Q_i \mid \ i = 1, 2, \ldots, M-1. \quad (5.35)$$

Then we may decompose the system of equations in (5.32) as

System 1 : $\sum_{|q|\le P} -4\pi^2(n-q)^2 a_q u_{n-q} + \sum_{|q|\le P} i2\pi(n-q) b_q u_{n-q}$

$$+ \sum_{|q|\le P} c_q u_{n-q} = d_n \ , \qquad |n| \le Q_1 - 1 \ ,$$

$$\sum_{|q|\le Q} u_q = 0 \ , \qquad \sum_{|q|\le Q} q u_q = 0 \ , \qquad (5.36.a)$$

with boundary conditions $u_q = u_q^{(0)}$, $Q_1 < |q| \le Q$, and with unknown variables u_q , $0 \le |q| \le Q_1$

System 2 : $\sum_{|q|\le P} -4\pi^2(n-q)^2 a_q u_{n-q} + \sum_{|q|\le P} i2\pi(n-q) b_q u_{n-q}$

$$+ \sum_{|q|\le P} c_q u_{n-q} = d_n \ , \qquad Q_1 < |n| \le Q_2 - 1 \ ,$$

$$\sum_{|q|\le Q} u_q = 0 \ , \qquad \sum_{|q|\le Q} q u_q = 0 \ , \qquad (5.36.b)$$

with boundary conditions $u_q = u_q^{(0)}, |q| \le Q_1 \ or \ Q_2 < |q| \le Q$, and with unknown variables u_q, $Q_1 + 1 < |q| \le Q_2$

. . .

. . .

. . .

System M : $\sum_{|q|\le P} -4\pi^2(n-q)^2 a_q u_{n-q} + \sum_{|q|\le P} i2\pi(n-q) b_q u_{n-q}$

$$+ \sum_{|q|\le P} c_q u_{n-q} = d_n \ , \qquad Q_{M-1} < |n| \le Q_M - 1 \ ,$$

$$\sum_{|q|\le Q} u_q = 0 \ , \qquad \sum_{|q|\le Q} q u_q = 0 \ , \qquad (5.36.c)$$

with boundary conditions $u_q = u_q^{(0)}$, $|q| \le Q_{M-1}$, and with unknown variables u_q, $Q_{M-1} + 1 < |q| \le Q_M = Q$

Since the above M systems are independent of each other, they can be solved in parallel. The initial boundary conditions can be chosen arbitrarily. For given boundary conditions, each system obtains a unique set of solutions, which can be used as the new boundary conditions of the other systems for the next iteration. We may repeat the same procedure until the solutions converge to some final values. This iterative algorithm is similar to the *block relaxation method* in the *space* domain, except that constraints arising from the space domain boundary conditions makes all systems coupled together. We can see that the coupling in the *space* domain depends on the order of the differential operators and the discretization scheme we choose for the operator $\frac{d^n}{dx^n}$ while the coupling in the *frequency* domain depends on the the property of the coefficient functions as well as on the space domain boundary conditions.

Of course, the system decomposition may be done in some other ways. For example, two adjacent systems may have an overlapping region. After each iteration, the values of the frequency components in the overlapping region can be obtained as a convex combination of the solutions provided by these two systems, which is called *overlapping block relaxation* in the frequency domain.

Although the above parallel algorithm in the frequency domain is interesting, there is an important difficulty associated with this method. It is that the *problem formulation* in the frequency domain, i.e., equation (5.32), for real world problems is not realistic. For a general 2-D problem, the geometry of the problem domain may be irregular. This property makes the Fourier transforms of the coefficient functions $a(x)$, $b(x)$, $c(x)$ and $d(x)$ extremely difficult, if not impossible. In addition, the space domain boundary conditions, homogeneous or nonhomogeneous, mix all frequency components together and, therefore, increase the complexity of applying the frequency domain approach. So, it is necessary to go back to problems formulated in the space domain.

The CMG method is basically a space domain iterative algorithm. However, the concept of decomposing a system according to its frequency bands' structure suggests a new methodology to achieve the same effect in the space domain. That is, in order to allow good convergence in all frequency bands, we may use a multigrid discretization scheme, in which each grid solves for a limited band of frequency components for the solution. In addition to getting parallel computation, there is another advantage in the multigrid concept. Since different frequency components can be discretized by grids of different sizes, the smoothing of the low frequency errors, performed by the coarser grid, turns out to be much faster than the smoothing of them with a single fine grid.

5.2.3 Overview of the Concurrent Multigrid Methods (CMG) - 2-D case

To analyze and design a CMG algorithm, we need the help of frequency domain analysis, or Fourier analysis. The above simple example gives us an idea of how to choose reasonable operators and driving functions in different grids required by the CMG method. The overview of the CMG method is shown in Figure 5.6. The details will be explained below.

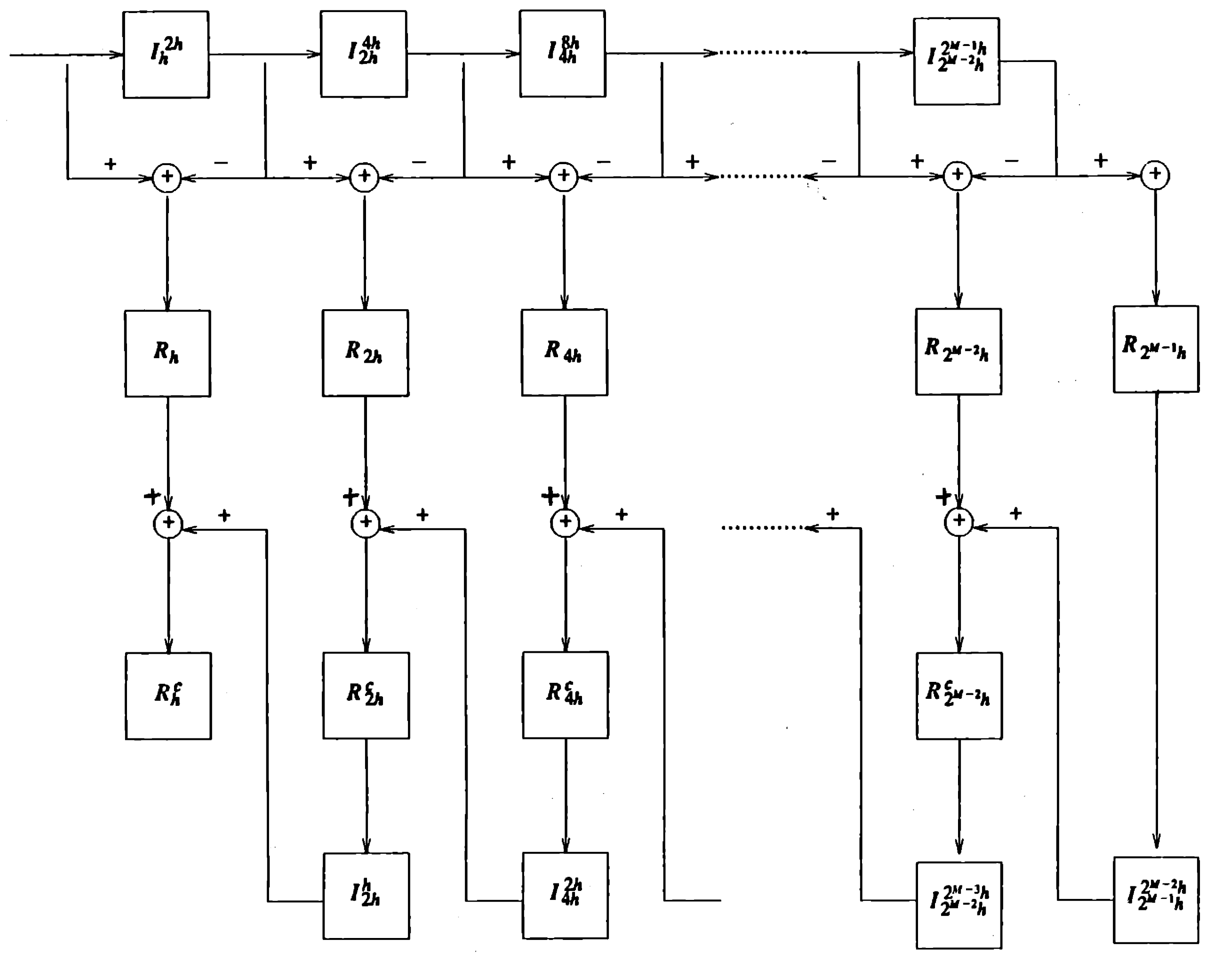

Fig 5.6 Diagram of a Simple Concurrent Multigrid (CMG) Scheme

(1) How to choose the operators $L_{2^{m-1}h,i,j}$, $B_{2^{m-1}h,i,j}$ and the driving functions $f_{2^{m-1}h,i,j}$, $g_{2^{m-1}h,i,j}$ for each grid

For a linear constant coefficient PDE, it is quite natural that operators at all grid levels should have the same form. For example, the discretized Laplacian operator at all grid levels should be

$$L_{2^{m-1}h,i,j} = \frac{E_1+E_1^{-1}+E_2+E_2^{-1}}{4} \;, \quad m = 1, 2, \cdots, M \;, \tag{5.37.a}$$

and a Dirichlet type boundary operator should be

$$B_{2^{m-1}h,i,j} = 1 \;, \qquad m = 1, 2, \cdots, M \;. \tag{5.37.b}$$

Note that although these operators are in the same difference forms, the difference operators for different grids operate on different grid points.

For a linear varying coefficient PDE, the situation becomes more difficult. The linear varying coefficient PDEs are so broad that it is not easy to give a precise rule to generate the coarser grid operators which is applicable for all cases. Instead of giving a concrete method, I will explain the motivation for getting coarser grid operators, then propose a general methodology to obtain the object we want. The main purpose of discretization and computation with coarser grids is to provide us the low frequency components of the solution more economically. So, we would like to neglect the unnecessay details contained in the finer grid driving functions and operators, as long as the *crude* operators and driving functions can give us good low frequency approximations of the solution.

A reasonable choice for a crude operator at some space point in the coarser grid is the average of this operator near the same point in the finer grid. Here, to average an operator can be interpreted as averaging its coefficients. This is equivalent to letting the coarser grid contain only low frequency components of the coefficient functions while the finer grid contains both low and high frequency components of the coefficient functions.

As regards the driving function, we may use the projection operator mentioned in Section 5.2.1 to separate the driving functions between different grid levels. The justification of this splitting is well explained by the above simple example.

Although the coarse grid cannot even provide us a very accurate problem formulation of the low frequency bands, because it neglects the coupling between the low frequency bands of the solution and the high frequency bands of coefficient functions, it gives us a good initial approximation to the low frequency modes and saves us lots of computations at finer grid levels. More refinement can be obtained by feeding this initial approximation to a finer level.

(2) How to get the solution $u_{2^{m-1}h,i,j}$ on each grid

The error smoothing procedure can be enhanced by using more effective relaxation schemes such as the LASOR algorithm proposed in Chapter 3. However, the lowest frequency components for different grid layers are no longer the same. As a consequence, the optimal relaxation factor ω is not only a function of space but also a function of the grid level.

Let us first summarize the LASOR result with one-grid discretization. In Chapter 3, it has been shown that given a local Jacobi relaxation operator $J_{i,j}$, the optimal relaxation factor with respect to *a single frequency* $e^{i(k_1x_1 + k_2x_2)}$ is

$$\omega_{i,j,opt}(\lambda_{i,j}) = \frac{2}{1 + \sqrt{1 - \lambda_{i,j}^2(J_{i,j})}} \quad , \tag{5.38}$$

where $\lambda_{i,j}$ is the eigenvalue of $J_{i,j}$ with the input frequency $e^{i(k_1x_1 + k_2x_2)}$. Furthermore, if only one grid level is allowed, the optimal relaxation factor $\omega_{i,j}^*$ for *all* frequencies becomes

$$\omega_{i,j}^* = \frac{2}{1 + \sqrt{1 - \rho_{i,j}^2(J_{i,j})}} \quad , \tag{5.39}$$

where $\rho_{i,j}$ is the spectral radius of $J_{i,j}$. And, $\mu_{i,j}$, the spectral radius of the one-grid LASOR operator $G_{i,j}$ is

$$\mu_{i,j}(G_{i,j}(\omega_{i,j}^* , J_{i,j})) = \omega_{i,j}^* - 1 \quad . \tag{5.40}$$

Now, since a multiple-level discretization scheme is used, the optimal relaxation factor of node (i,j) and grid level m, $\omega_{i,j,m}^*$, is given by

$$\omega_{i,j,m}^* = \frac{2}{1 + \sqrt{1 - \rho_{i,j,m}^2(J_{i,j,m})}} \quad , \tag{5.41}$$

where $J_{i,j,m}$ is the Jacobi relaxation operator in the mth grid and $\rho_{i,j,m}(J_{i,j,m})$ is the spectral radius of $J_{i,j,m}$. Finally, $\mu_{i,j,m}$, the spectral radius of the LASOR operator $G_{i,j,m}$, can be derived as

$$\mu_{i,j,m}(G_{i,j,m}(\omega_{i,j,m}^* , J_{i,j,m})) = \omega_{i,j,m}^* - 1 \quad . \tag{5.42}$$

In the following, I will use the model problem as an example to illustrate the difference between one-grid and multi-grid LASOR methods. From the result of

Chapter 3, we know

$$\rho_{i,j}\,(J_{i,j}) = \frac{\cos(\hat{k}_1 h) + \cos(\hat{k}_2 h)}{2} \quad , \tag{5.43}$$

where $\hat{k}_1$ and $\hat{k}_2$ are the lowest frequencies in the x_1 and x_2 directions. Now, in the multi-grid case, the spectral radius of the m th level is

$$\rho_{i,j,m} = \max \ \frac{\cos(k_1 2^{m-1}h) + \cos(k_2{}^{m-1}h)}{2} \quad , \tag{5.44}$$

under the constraint that k_1 and k_2 appear at the m th level. From (5.11) and (5.16), we know that the above expression is equivalent to

$$\rho_{i,j,m} = \max_{\frac{\pi}{2} \le \sqrt{K_{1,m}^2 + K_{2,m}^2} \le \pi} \frac{\cos K_{1,m} + \cos K_{2,m}}{2} = \frac{\cos(\frac{\sqrt{2}}{4}\pi) + \cos\cos(\frac{\sqrt{2}}{4}\pi)}{2}$$

$$= 0.444016 \qquad \text{for } m = 2,3,\ldots,M-1 \ , \tag{5.45.a}$$

and the spectral radius is

$$\mu_{i,j,m} = \omega^*_{i,j,m} - 1 = \frac{2}{1 + \sqrt{1 - (0.444016)^2}} - 1$$

$$= 0.054842 \qquad \text{for } m = 2,3,\ldots,M-1 \ . \tag{5.45.b}$$

For the finest level ($m = 1$), although the constraint given by equation (5.16.a) is

$$\frac{\pi}{2} \le \sqrt{K_{1,1}^2 + K_{2,1}^2} \ \text{ and } \ -\pi \le K_{1,1}\,, K_{2,1} \le \pi \ , \tag{5.46}$$

we still have to consider another constrain arising from the fact that the discretization error should not be too large. As a consequence, the relaxation performed in the finest level is not aimed at smoothing such high frequency components. The function of the relaxation in the finest grid is to further refine the solution, when the solutions of different levels are combined together and transferred to the finest grid. This point will be mentioned in (3). For the coarsest level ($m = M$), we have

$$\rho_{i,j,M} = \frac{\cos(\hat{k}_1 2^{M-1}h) + \cos(\hat{k}_2 {}^{M-1}h)}{2} \ . \tag{5.47}$$

Based on (5.43) and (5.47), we may compare the convergence rates of the one-grid/LASOR algorithm and the M-grid/LASOR algorithm with respect to the lowest frequencies $\hat{k}_1 = \hat{k}_2 = \pi$, under the assumption $\pi\, 2^{M-1}\, h$ is much less than 1.

one-grid/LASOR algorithm :

$$\mu_{i,j} = \frac{2}{1 + \sqrt{1 - \cos^2(\pi h)}} - 1 = \frac{1 - \sin \pi h}{1 + \sin \pi h} = 1 - 2\pi h \tag{5.48.a}$$

$$R_\infty[\mu_{i,j}] = -\ln(\mu_{i,j}) = 2\pi h \tag{5.48.b}$$

and

M-grid/LASOR algorithm :

$$\mu_{i,j,M} = \frac{2}{1 + \sqrt{1 - \cos^2(\pi\, 2^{M-1}h)}} - 1 = 1 - 2\pi\, 2^{M-1}\, h \tag{5.49.a}$$

$$R_\infty[\mu_{i,j,M}] = -\ln(\mu_{i,j,M}) = 2\pi\, 2^{M-1}\, h \tag{5.49.b}$$

Lastly, we list a table to compare the spectral radii and convergence rates of different relaxation methods combined with one-grid and multi-grid discretization schemes.

	JACOBI	*G-S*	*LASOR*
ONE-GRID	$\rho = 1 - \frac{\pi^2 h^2}{2}$ $R = \frac{\pi^2 h^2}{2}$	$\mu = 1 - \pi^2 h^2$ $R = \pi^2 h^2$	$\mu = 1 - 2\pi h$ $R = 2\pi h$
M-GRID	$\rho = 1 - \frac{\pi^2 (2^{M-1} h)^2}{2}$ $R = \frac{\pi^2 (2^{M-1} h)^2}{2}$	$\mu = 1 - \pi^2 (2^{M-1} h)^2$ $R = \pi^2 (2^{M-1} h)^2$	$\mu = 1 - 2\pi 2^{M-1} h$ $R = 2\pi 2^{M-1} h$

The M-grid/LASOR algorithm converges faster than the one-grid/LASOR by a factor 2^{M-1} while the M-grid/Jacobi method or the M-grid/G-S method converges faster than the one-grid/Jacobi or the one-grid/G-S method by a factor 4^{M-1}. We may also note that the advantage of the LASOR method over the Jacobi and Gauss-Seidel methods will be more substantial for the finer grids than with coarser grids. However, as long as $2^{M-1} h$ is much less than 1, the LASOR algorithm is still preferred no matter what kind of discretization scheme is used.

(3) How to combine these solutions together

In stage (2), we separated the problem into different frequency bands, neglected the coupling effects among these bands, and obtained the solutions for different frequency bands by relaxation in different grids. In order to get a correct final solution, we need schemes to combine the solutions of different bands together and to refine the combined solution so that the coupling effect can be added back.

An easy combination scheme is to transfer the coarser grid solution to the finer grid and add the transferred solution (low frequency components) to the solution already in the finer grid (high frequency components), then perform several relaxation steps in the combined solution to obtain the refinement effect. On the other hand, if the convergence rate of the relaxation for the combined solution is too slow, we may separate the low and high frequency components again, transfer the low frequency components to the coarser grid, and perform relaxation for these split solutions. The splitting and recombination procedures can occur between any pair of levels and in any reasonable sequence. However, these procedures should not be too complicated, because they require extra control signals and mechanisms to coordinate all processors in the same grid to do the same thing.

A straightforward splitting and recombination scheme is shown in Figure 5.6 where the splitting occurs at the beginning stage of the sequence from the finest grid to the coarsest grid and the recombination occurs at the final stage of the sequence from the coarsest grid to the finest grid.

5.3 Computer Architectures for Concurrent Multigrid Methods

The supporting computer architectures for multigrid methods have been studied and proposed by Brandt [26] and Gannon [27]. The most intuitive and ideal parallel architecture for the concurrent multigrid method described above is the pyramid multi-processor array, which has also been studied in other contexts such as parallel computing machines for image processing [28]. Other possible alternatives are embedding the concurrent multigrid method in some fixed computer architectures, say, mesh-connected array, mesh-shuffle connected system, and so on [27].

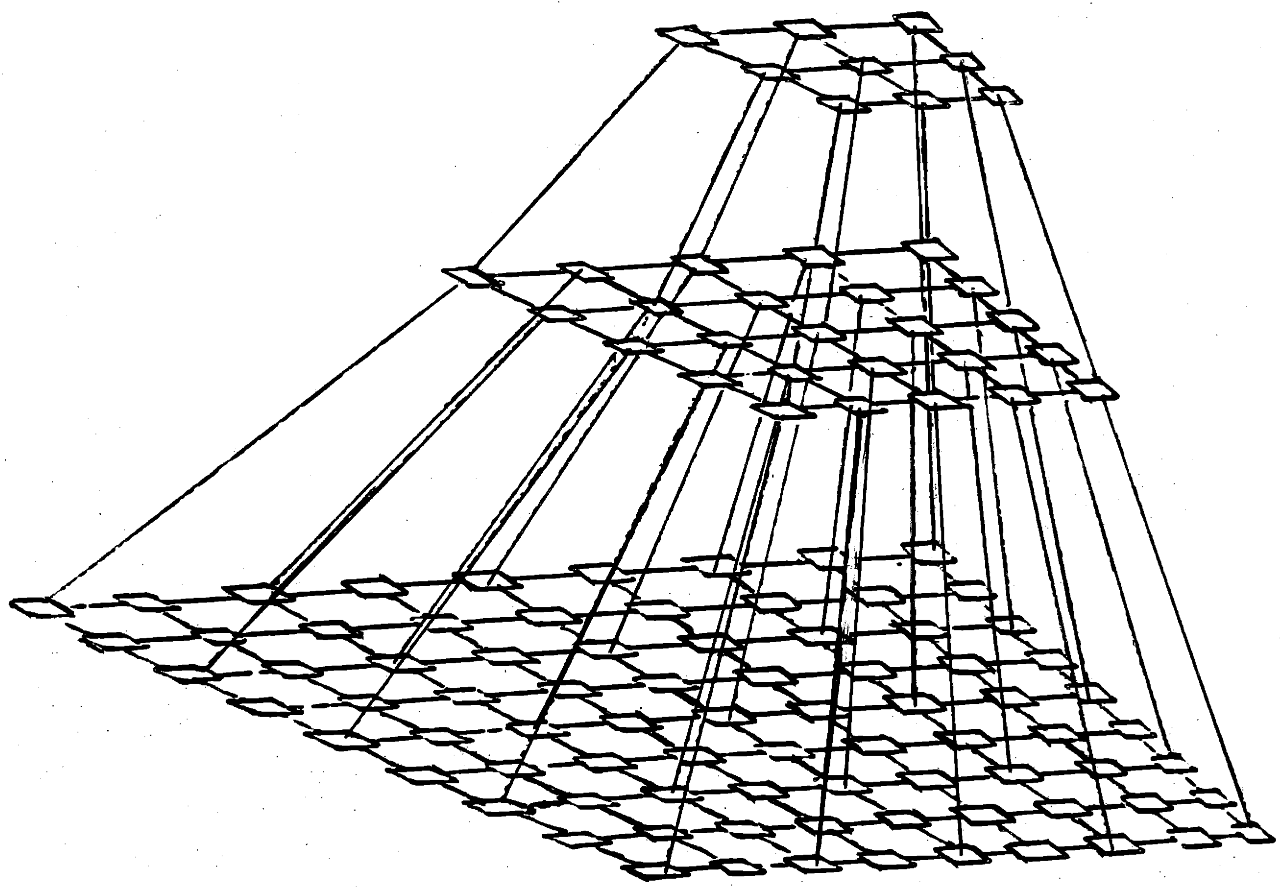

Fig 5.7 Pyramid Multi-processor Array

Generally speaking, the area of parallel computer architectures is still an active research field and many questions remain. In addition, it is not easy to consider all possible details without implementing a real system. So, instead of giving a perfect solution, I would like to mention results discussed by others and give some my own opinions to make this Chapter complete.

The interconnection of a pyramid multi-processor array is shown in Figure 5.7. These processors are configured into M levels where the i_th level is a $(2^{n-i+1} + 1) \times (2^{n-i+1} + 1)$ square grid. Each processor is connected to its 8 or 4 nearest neighboring processors at the same level. In addition, the processor (p, q) at the ith level is connected to the processor $(\frac{p-1}{2}, \frac{q-1}{2})$ at the $(i+1)$th level.

There are two different classes of operators in the CMG algorithm : the grid transfer operators, including the projection and the interpolation operators, and the relaxation operator. The grid transfer operators require the data to flow between two adjacent processors in different layers while the relaxation operator requires the data to flow between the nearest neighbor processors in the same layer. Both of them satisfy the local communication constraint. Consequently, communication cost is lowest if the CMG algorithm is implemented by this architecture. Another advantage of the pyramid structure is that the architecture reflects the data flow pattern required by the algorithm, and it is, therefore, easier to program each processor.

Although the pyramid structure is promising, there are still issues remaining. It has been shown that relaxation can be performed in an asynchronous way so that we do not have to coordinate all processors to do one relaxation step at the same time. However, the grid transfer operation needs synchonization. In order to coordinate all processors to perform the projection or interpolation, extra control signals and global communication at one level are necessary. Another difficulty arises when we try to implement *adaptive* projection or interpolation schemes, by which we mean controlling the grid transfer operator based on the current convergence rate of the iterative solutions. It usually requires global information to judge whether the convergence rate is slow or fast. Therefore, there is a tradeoff between efficiency and implementation complexity.

The pyramid structure can be composed of many small processors each of which is integrated in a chip. However, it also seems possible to implement many processors in a single chip using the wafer scale integration (WSI) technology. Although the WSI technology is not quite mature at present, we may still be interested in embedding the 3-D pyramid structure in a 2-D plane. This problem has been studied by Gannon [27] where several possibilities are mentioned: mesh-connected array, mesh-shuffle connected system, permutation network, and direct VLSI embedding. One way to implement the pyramid structure in the mesh-connected array is illustrated by Figure 5.8.

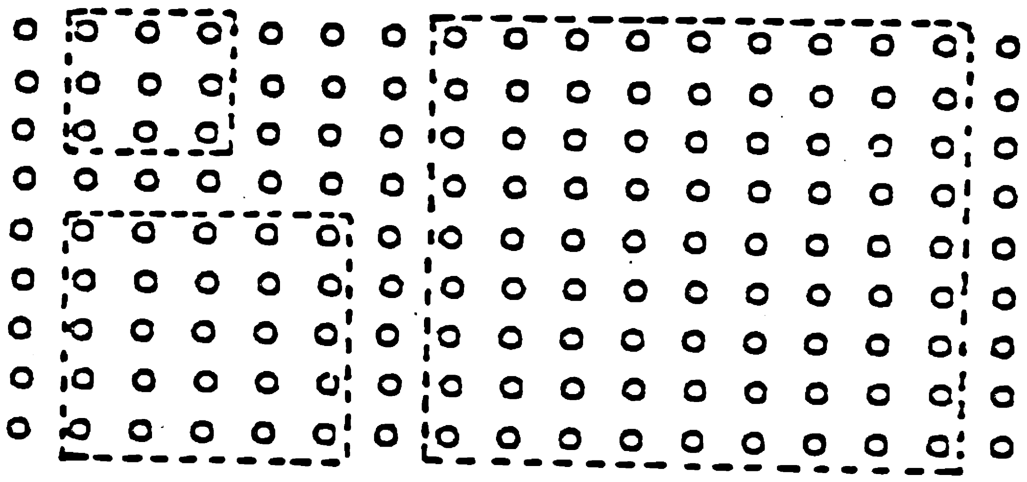

Fig 5.8 Embedding a Pyramidal Architecture in a 2-D Mesh-connected Processor Array

Chapter 6. Extensions and Conclusion

In this thesis, two distributed and parallel numerical PDE algorithms have been proposed. The LASOR (Locally Accelerated Successive Over-Relaxation) algorithm and its performance were discussed in Chapters 3 and 4 while the CMG (Concurrent Multi-Grid) algorithm was presented in Chapter 5. This last chapter will describe some extensions of research along these two directions and will present some conclusions.

6.1 Extensions

The LASOR algorithm includes two important steps. The first is to determine the admissible lowest frequencies using boundary condition information. The second is to approximate the PDE operator by a linear constant coefficient difference operator locally, divide the nodes into red and black points, and form a locally accelerated successive over-relaxation (LASOR) operator. In previous discussions, some ideal assumptions were made so that the analysis and design of the LASOR algorithm become very simple. However, we may encounter several difficulties in applying the LASOR method to solving real world problems directly.

Under the assumption that the problem domain is a unit square and that the boundary condition operator is constant along each edge, the procedure for determining the lowest admissible frequencies is straightforward. These assumptions make the basis functions separable and easy to analyze. However, in practice, the above assumptions may not hold. The problem domain is usually of irregular shape and the boundary condition operators may have space-varying coefficients. As a consequence, to find the lowest frequency component is not an easy task as shown before.

The second difficulty is related to the construction of the LASOR operator. If the coefficients of a PDE operator have some discontinuities in some region, the Jacobi relaxation operator is not smooth over the region with discontinuous coefficients. Then, how to determine the local optimal relaxation factors based on these abruptly changing operators is still an open question.

It has been mentioned that to map the irregular domain problem into a regular processor domain is important in practice. In addition, we also need some schemes to partition the grid points evenly between all processors, when the number of grid points is larger than that of processors. The mapping and partitioning problems have been studied recently [29] [30], but not too many results are known.

It also seems interesting and challenging to see whether the local acceleration concept can be generalized to PDE problems of other types, such as hyperbolic PDEs and nonlinear PDEs, and other discretization schemes, say, the finite-element method.

The discussion of the CMG method in Chapter 5 can be applied to smooth PDE operators with simple boundary conditions and square problem domain. However, if we have irregular problem domains, complicated boundary conditions, and PDE operators with discontinuous coefficients, then there are some difficulties in applying the CMG method directly. How to adapt the CMG method to these more general situations is an important topic for further research. The data transfer among grids close to boundary regions and discontinuous-coefficient regions requires more careful considerations.

The performance of the CMG method highly depends on the supporting architectures and should be studied further either by some analytical approach or by computer simulation.

I also look forward to seeing how the CMG method can be adapted to more general PDEs including the parabolic, hyperbolic, and nonlinear PDEs.

6.2 Conclusion

We may note that *distributed* computational PDE algorithms (LASOR) are different from traditional *central* computational methods (SOR) in several ways.

First, distributed computation provides a natural way to achieve highly parallelism. Second, distributed algorithms suggest a space-adaptive acceleration scheme, which is not feasible in central computation. Third, although global information is required in determining the local optimal acceleration factors, it seems that only very little global information is relevant. Last, we benefit a lot in designing the local acceleration algorithm from the *simple* structure of the local operator and the minimum global information while the determination of the central acceleration factor is very complicated and time-consuming.

These nice properties are believed to be closely related to the special structure of PDE. Partial differential equations are formulated to describe local interactions in the physical world. As a consequence, no interaction can happen between two different space points without influencing the region between these points. The locality property is very similar to the local communication constraint imposed by VLSI computation. Therefore, although this constraint is critical for other types of problems, it is not as severe in numerical PDE problems.

The thesis has also demonstrated the use of the local Fourier analysis approach, or frequency domain approach, to analyze both the LASOR algorithm and the CMG algorithm. This methodology sets a bridge between numerical analysis for solving PDEs and digital signal processing. This new method seems more informative than traditional matrix iterative methods, which usually hide information in a huge matrix through ordering. In addition, the local Fourier analysis approach provides a way to analyze distributed numerical algorithms while matrix iterative methods only can be applied to central numerical algorithms. A closer relationship between numerical analysis techniques and Fourier analysis is expected in the future.

What we have done is only the beginning of a new field, combining VLSI technology, local Fourier analysis, numerical analysis, and distributed computation. If VLSI processor arrays become available in the near future and parallel distributed numerical algorithms can be better understood and explored, then it is likely that the traditional central numerical algorithms will be revolutionized in such a way that parallel and distributed numerical algorithms will become the main approach to solve large-scale scientific problems arising from PDEs. Due to the revolution both in hardware technology (VLSI) and in parallel and distributed algorithms, it is not surprising that the computers of next generation will be hundreds of times faster than today's most powerful computers.

References

1. G. Birkhoff, "Solving Elliptic Problems : 1930-1980," in *Elliptic Problem Slovers*, ed. M. H. Schultz, pp. 17-38, Academic Press, Inc., New York, N. Y., 1981.
2. T. L. Jordan, "A Guide To Parallel Computation and Some Cray-1 Experiences," in *Parallel Computations*, ed. G. Rodrigue, pp. 1-50, Academic Press, Inc., New York, N. Y., 1982.
3. D. Heller, "A Survey of Parallel Algorithms in Numerical Linear Algebra," *SIAM Review*, vol. 20, no. 4, pp. 740-777, Oct. 1978.
4. A. H. Sameh and D. J. Kuck, "On Stable Parallel Linear System Solvers," *Journal of ACM*, vol. 25, no. 1, pp. 81-91, Jun. 1978.
5. H. T. Kung and C. E. Leiserson, "Systolic Array (for VLSI)," in *Sparse Matrix Proc. 1978*, pp. 256-282, SIAM, 1979.
6. H. T. Kung, "Why Systolic Architectures?," *Computer*, vol. 15, no. 1, pp. 37-46, Jan. 1982.
7. L. S. Haynes, R. L. Lau, D. P. Siewiorek, and D. W. Mizell, "A Survey of Highly parallel Computing," *Computer*, vol. 15, no. 1, pp. 9-24, Jan. 1982.
8. R. Vichnevetsky, *Computer Methods for Partial Differential Equations, Vol. 1, Elliptic Equations and the Finite-Element Method,* Prentice-Hall, Inc. , Englewood Cliffs, N.J. , 1981.
9. D. k. Faddeev and V. N. Faddeeva, *Computational Methods of Linear Algebra,* W. H. Freeman and Company , San Francisco, CA , 1963.
10. B. R. Musicus , "Levinson and Fast Choleski Algorithms for Toeplitz and almost Toeplitz Matrices," MIT Technical Report, 1982.
11. S. Y. Kung, K. S. Arun, R. J. Gal-ezer, and D. V. Bhaskar Rao, "Wavefront Array Processor: Language, Architecture, and Applications," *IEEE Trans. on Computer*, vol. 31, no. 11, pp. 1054-1066, Nov. 1982.
12. H. M. Ahmed , J. Delosme , and M. Morf , "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing," *Computer*, vol. 15 , no. 1 , pp. 65-82.
13. S. Y. Kung and Y. H. Hu, "A Highly Concurrent Algorithm and Pipelined Architecture for Solving Toeplitz Systems," *IEEE Trans. on ASSP*, vol. 31 , no. 1, Feb. 1983.
14. R. S. Varga, *Matrix Iterative Analysis,* Prentice-Hall, Inc. , Englewood Cliffs, N.J. , 1962.
15. L. A. Hageman and D. M. Young, *Applied Iterative Methods,* Academic Press, Inc. , New York, N.Y. , 1981.
16. B. Friedman, *Principles and Techniques of Applied mathematics,* John Wiley & Sons, Inc., New York, N.Y. , 1956.
17. G. Dahlquist, A. Björck, and N. Anderson, *Numerical Methods,* Prentice-Hall, Inc. , Englewood Cliffs, N.J. , 1974.
18. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing,* Prentice-Hall, Inc. , Englewood Cliffs, N.J. , 1975.
19. L. Adams and J. M. Ortega, "A Multi-Color SOR Method for Parallel Computation," ICASE report, 82-9, Apr. 1982.

20. D. P. Bertsekas , "Distributed Asynchronous Computation of Fixed Points," *Mathematical Programming*, vol. 27, pp. 107-120, 1983.

21. R. P. Fedorenko, "The Speed of Convergence of an Iterative Process," *U.S.S.R. Comp. Math. and Math. Phys.*, vol. 4 , no. 3 , pp. 227-235 , 1964.

22. N. S. Bakhvalov , "On the Convergence of a Relaxation Method with Natural Constraints on the Elliptic Operator," *U.S.S.R. Comp. Math. and Math. Phys.*, vol. 6 , no. 5, pp. 101-135, 1966.

23. A. Brandt , "Multi-level Adaptive Solutions to Boundary-value Problems," *Math. of Comp.*, vol. 31, no. 138, pp. 333-390, Apr. 1977.

24. K. Stuben and U. Trottenberg, "Multigrid Methods : Fundamental Algorithms, Model Problem Analysis, and Applications," in *Multigrid Methods*, ed. U. Trottenberg, Springer-Verlag, Berlin Heidelberg New York, 1982.

25. D. E. Dudgeon and R. M. Mersereau, *Multidimensional Digital Signal Processing,* Prentice-Hall, INC., Englewood Cliffs, New Jersey, 1984.

26. A. Brandt, "Multigrid Solvers on Parallel Computers," in *Elliptic Problem Slovers*, ed. M. H. Schultz, pp. 39-83, Academic Press, Inc., New York, N. Y., 1981.

27. D. Gannon and J. V. Rosendale, "Highly Parallel Multigrid Solvers for Elliptic PDEs: An Experimental Analysis," ICASE Report 82-36, 1982.

28. L. Uhr, "Pyramid Multi-computer Structures and Augumented Pyramids," in *Computing Structures for Image Processing*, ed. M.J.B. Duff, pp. 95-112, Academic Press, London, 1983.

29. D. Gannon, "On Mapping Non-uniform PDE Structures and Algorithms onto Uniform Array Architectures," in *Proceeding of International Conference on Parallel Processing*, 1981.

30. S. H. Bokhari, "On the Mapping Problem," *IEEE trans. on Computers*, vol. C-30, no. 3, pp. 207-214, Mar. 1981.

31. J. R. Rice, "ELLPACK : Progress and Plans," in *Elliptic Problem Slovers*, ed. M. H. Schultz, pp. 135-162, Academic Press, Inc., New York, N. Y., 1981.