90-29

Parallel Preconditioned Conjugate Gradient Methods for Elliptic Partial Differential Equations

by

Charles Hok-shun Tong

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

Financial support for this work has been provided in part by the National Science Foundation under contract NSF-DMS87-14612 and BBS87 14206, the Department of Energy under contract DE-FG03-87ER25037, the Army Research Office under contract DAAL 03-88-K-0085, the 1989 summer support from Research Institute for Advanced Computer Science (RIACS) under Cooperative Agreement NCC 2-387, and by the a DARPA-sponsored Graduate Research Assistantship (award no. 26947G) through UMIACS at the University of Maryland.

ABSTRACT

Parallel Preconditioned Conjugate Gradient Methods

for Elliptic Partial Differential Equations

by

Charles Hok-shun Tong

This thesis considers the development of multilevel preconditioners in conjunction with the conjugate gradient method for the solution of elliptic partial differential equations (PDEs) and their implementation on a massively parallel multiprocessor. Since there is global dependence inherent in the physical processes described by the elliptic PDEs, we believe that a 'good' preconditioner must account for such global coupling in order to give fast convergence rates. Moreover, with the advent of affordable massively parallel computers, a 'good' preconditioner should also exhibit high degree of parallelism. However, we show that many classical preconditioners in general do not possess both properties. Therefore there is a fundamental tradeoff between convergence rate and amount of parallelism to achieve optimal performance. The tradeoff is confirmed by our numerical experiments on the Connection Machine (CM).

We then study and develop multilevel preconditioners that offer high degree

of parallelism and account for global coupling. In particular, we develop a class of multilevel filtering (MF) preconditioners. This thesis presents both analytical and numerical results when these preconditioners are applied to some self-adjoint elliptic problems in two- and three-dimension. Fourier method is used as a tool to understand the preconditioners for model problems on uniform grids. However, it should be emphasized that the MF method can be applied to more general problems such as discontinuous coefficient problems and on irregular grids. Numerical experiments show the effectiveness of the MF preconditioners when applied to variable and discontinuous coefficient problems. We then show how to extend this class of preconditioners to more general elliptic problems which include anisotropic problems, biharmonic equation, second-order self-adjoint problem with local mesh refinement, interface operators in domain decomposition problems, etc. Again, Fourier method is used to explore insights into the behavior of the preconditioners on model problems and numerical experiments show that the preconditioners are indeed effective. A performance analysis of the MF on the CM is also included.

We also study another multilevel algorithm, the Fast Fourier Transform (FFT), which is useful in the solution of many PDEs. We describe an ordered FFT algorithm which optimizes the amount of interprocessor communication on massively parallel distributed-memory multiprocessors. In addition, a parallel method for calculating the trigonometric factors is proposed and an error analysis is included.

CHAPTER 1

Introduction

1.1 Statement of Problem

This research develops parallel numerical methods for solving linear systems arising from the discretization of elliptic partial differential equations (PDEs) on massively parallel computers. In particular, we are interested in solving second-order self-adjoint elliptic PDEs of the form

$$\sum_{i=1}^{d}(\frac{\partial}{\partial x_i}a_i\frac{\partial u}{\partial x_i} + b_i\frac{\partial u}{\partial x_i}) + cu = f \quad in \quad \Omega = [0,1]^d$$

where $d = 2$ and 3 for the two- and three-dimensional problems. Here a_i, b_i, and c can depend on $x_j, j = 1, \cdots, d$. For example, when $a_i = 1$, $b_i = 0$, $c = 0$ for all i we have the Poisson problem. We are also interested in solving the fourth order biharmonic equation in two- dimension, namely

$$\triangle^2 u = f(x,y) \text{ in } \Omega = [0,1]^2,$$

subject to boundary conditions

$$u = f(x,y) \quad and \quad \frac{\partial u}{\partial n} = g(x,y) \quad for \quad (x_1, x_2) \in \partial\Omega$$

where $\partial/\partial n$ denotes differentiation in an outward normal direction to the boundary $\partial\Omega$.

These problems find wide applications in many areas of science and engineering including oil reservoir simulation, semiconductor device simulation, computational fluid dynamics, etc. Moreover, these problems are computation-intensive and it is

not an overstatement to claim that the need to solve these problems fast is a main driving force for the development of supercomputers.

1.2 Linear Algebra Techniques for Elliptic PDE's

The discretization of the elliptic partial differential equations mentioned in the last section using finite difference or finite elements gives rise to linear systems of the form

$$Au = f$$

where A is a large, sparse, and symmetric positive definite matrix. Techniques for solving such linear systems can be classified as either direct methods or iterative methods.

1.2.1 Direct Methods

Many direct methods are based on Gaussian elimination. Examples of direct methods are the LU factorization, Cholesky factorization [18], fast Poisson solvers using fast Fourier transform (FFT), etc. If exact arithmetic is used (i.e. no round-off), the maximum number of arithmetic operations needed is known in advance as a function of the problem size, N.

Direct methods are most suitable for dense systems as well as banded systems, but they encounter the notorious "fill-in" problem when applied to general sparse systems. "Fill-in" arises when many more non-zero entries are created during the process of Gaussian elimination, so that the L and U factors are less sparse than A. The consequences are more storage needed and higher computational cost. For example, the operation counts for solving elliptic problems on two- and three-dimensional domains ($n \times n$ and $n \times n \times n$) using band elimination are $O(n^4)$ and

$O(n^7)$ respectively. Efforts have been made to alleviate this problem by exploring different types of orderings and techniques such as graph-theoretic approach and nested dissection [47].

1.2.2 Iterative Methods

Iterative methods start with an initial guess x^0 and algorithmically update x to obtain a sequence x^0, x^1, x^2, ... x^k, ..., of approximate solutions intended to be successively closer to the true solution x. Examples of iterative methods are Jacobi, Gauss-Seidel (GS), Successive Over-relaxation (SOR) [110, 105], alternate direction implicit (ADI) [105], Chebyshev semi-iterative (CSI) [110, 56], preconditioned conjugate gradient (PCG) [29], multigrid (MG) [24, 14], etc. Some of these methods can further be classified according to the types of orderings used, such as natural ordering, red/black ordering, multi-color ordering [1], and etc.

Iterative methods have the advantage that they do not incur the "fill-in" problems as seen in the direct methods, since the original matrices are not altered during the solution process. In addition, the number of arithmetic operations needed to solve a problem is typically $O(pq)$ where p is the number of iterations needed for convergence and q is the number of non-zero entries in the original matrix A, which means that these methods have the potential to be much more efficient than the direct methods when p and q are small (fast convergence rate and the matrix is sparse). In this dissertation, we will concentrate on one particular class of iterative method - the preconditioned conjugate gradient method (PCG). The PCG method has been known as an efficient technique for solving large sparse symmetric positive definite linear systems of equations. Many iterative methods have the drawback that they require the estimation of parameters

for fast convergence. For example, the SOR method requires the estimation of the relaxation parameter ω, and the CSI method requires the estimation of the largest and smallest eigenvalues of the problem matrix. The PCG method, which can be considered as a minimization method, does not require any parameter estimation. Moreover, the PCG method can be very competitive for computer implementation by the use of "good" preconditioners.

1.3 Parallel Scientific Computations

In recent years there has been increasing efforts in developing parallel computers for use in scientific applications. With the advent of VLSI technology, we are able to build more complex circuits at an affordable cost. This creates a favorable environment for exploring different design alternatives and we have seen many creative parallel architectural designs ranging from general purpose to special purpose architectures. Some of these novel parallel architectures are the pipeline or vector computers such as the CDC Cyber 205 and the CRAY computers, shared memory multiprocessors such as the Alliant Family, message passing multiple instruction multiple data (MIMD) multiprocessors such as the Intel iPSC and the NCUBE computers, massively parallel single instruction multiple data (SIMD) machine such as the ICL DAP and the Connection Machine, hybrid or cluster machines such as the CEDAR computers developed at the University of Illinois Center for Supercomputing Research and Development, special purpose systolic computers such as the WARP machine developed at Carnegie-Mellon University, dataflow computers, just to name a few. Ortega and Voigt [90] have an excellent survey on numerical methods on vector and parallel computers and several other books such as [63, 64] have a comprehensive description of parallel computers.

The challenge for the scientific computing community using parallel computers is to match algorithms to parallel computers in order to obtain the best performance. This task amounts to designing new parallel algorithms, and understanding thoroughly the architectural features of the machine so that the suitability of a particular algorithm on a particular machine can be assessed and the computations can be arranged to fully utilize the machine. For example, given a vector processor, it is desirable to design algorithms that have low arithmetic complexity and many long vector operations.

Some of the commonly used performance measures are MFLOPS (million floating point operations per second), efficiency, speedup, execution time, cost, etc. For example, speedup can be defined as the ratio of the execution time using the fastest sequential algorithm on one processor to that using the parallel algorithm on p processors [90]. Efficiency is defined as the ratio of speedup using p processor to p. It should be pointed out that a particular algorithm-architecture match may give high MFLOPS but long execution time. It is up to the user to perform tradeoffs and choose for himself the best performance measure.

This dissertation centers on developing "good" preconditioner on massively parallel computers such as the Connection Machine (CM). By "good" we mean we are interested in preconditioners that give the lowest execution time of the overall solution process. We will show that the conjugate gradient method without preconditioning is already very efficient on the CM. Many of the classical preconditioners such as incomplete factorization and polynomial preconditioners which improve performance on vector computers are shown to be unsuitable on the CM. We also investigate the class of multilevel preconditioners and show that the multilevel preconditioners are promising candidates on massively parallel machines which have

support for efficient global communication such as hypercube interconnection.

1.4 Contribution of this Research

The emphasis of this research is on the development of parallel preconditioners for the conjugate gradient method suitable for implementation on massively parallel computers. We begin with a performance study of a number of classical preconditioners on the CM. The conclusion to the initial study is that the class of multilevel preconditioners strikes a good balance on convergence rates and amount of parallelism so that good performance on massively parallel architectures such as the CM can be obtained.

We then develop the class of multilevel filtering (MF) preconditioners which we study both analytically and numerically on second-order self-adjoint problems. We also show the superior performance of this class of preconditioners on the CM as compared to many other classical preconditioners. Using the same MF framework we are able to extend this class of preconditioners to effectively solve more general elliptic problems such as anisotropic problems, biharmonic equation, positive-definite Helmholtz equation, convection-diffusion equations, problems with local mesh refinement, and interface operators arising from domain decomposition methods. Both Fourier analysis and numerical results are presented.

We also study the efficient implementation of ordered Fast Fourier Transform (FFT), an example of massively parallel multilevel algorithm, on massively parallel machines. FFT is useful in, for example, solving PDEs using the spectral method. We develop a new algorithm for ordered FFT which optimizes the amount of interprocessor communication on massively parallel distributed-memory multiprocessors. Moreover, a new parallel method for calculating the trigonometric factors

is proposed. Implementation of this ordered FFT on the CM gives performance close to 0.9 GFLOPS. An error analysis of such a scheme is also included.

The major research contributions are summarized in the following:

- performance studies of some classical preconditioners on the CM, the conclusion of which leads to the consideration of multilevel preconditioners for massively parallel computers,
- perform numerical experiments to verify the effectiveness of MF on some two- and three-dimensional second-order self-adjoint problems and compare the effectiveness of MF with other preconditioners,
- develop efficient MF preconditioners for the following classes of elliptic problems (Fourier analysis and numerical experiments):
  - anisotropic problems,
  - biharmonic equation,
  - problems with local mesh refinement,
  - positive definite Helmholtz equation,
  - convection-diffusion equations,
- develop an efficient multilevel preconditioners for interface operators arising from domain decomposition method,
- performance analysis of the MF preconditioners on the CM for some second- and fourth-order problems,
- develop a new ordered FFT algorithm:
  - implementation on the CM (performance close to 0.9 GFLOPS),

- develop a new parallel method for computing trigonometric factors,
- error analysis of the new method.

1.5 Organization of this Dissertation

Chapter 2 presents the initial set of experiments comparing the performances of some classical preconditioners on the CM. Chapter 3 first draws some conclusions for the motivation of the class of multilevel preconditioners and then surveys some of the previous works on multilevel preconditioners. Chapter 4 introduces the idea of multilevel filtering (MF) preconditioning. Analysis as well as numerical results are included for the two- and three-dimensional Poisson-like problems. Chapter 5 extends the idea of MF preconditioning to other more general elliptic problems such as those listed in the last section. Chapter 6 presents a new preconditioner based on the ideas of multilevel preconditioning and domain decomposition. Again analysis as well as numerical results are given to show the effectiveness of the new preconditioner. Chapter 7 describes the results of performance studies of the MF preconditioners on the CM. Timing models are developed to study the efficiency improvement of the MF preconditioners for some elliptic problems. Chapter 8 first overviews the class of massively parallel multilevel algorithms, and then presents an ordered Fast Fourier Transform algorithm that optimizes the communication overhead on massively parallel computers. This algorithm includes a new parallel method for the computation of the trigonometric factors. An error analysis of this method will also be included. Chapter 9 summarizes this research as well as lists some future research directions.

CHAPTER 2

Preconditioned Conjugate Gradient Methods and Their CM Performance

2.1 Introduction

The conjugate gradient method, coupled with "good" preconditioning, has been known as an efficient technique for solving large sparse symmetric positive definite linear systems of equations such as those generated by the discretization of elliptic partial differential equations in two or three dimensions. In the past, many preconditioners have been proposed which have helped to make the preconditioned conjugate gradient (PCG) methods very competitive for computer implementation. Some of these preconditioners offer condition number improvement of an order of magnitude, so that the overall operation count to achieve convergence is greatly reduced. For example, the modified incomplete Cholesky (MIC) factorization and its invariant with natural ordering was widely used on sequential computers. However, such preconditioners often have the property that they are very much sequential. For example, the maximum degree of parallelism of the MIC preconditioner for a $n \times n$ grids is $O(n)$. Therefore, these sequential preconditioners are unable to exploit efficiently the computational resources offered by massively parallel computers such as the CM, as it will be shown later from experimental results. In an effort to increase the amount of parallelism, one method is to reorder the sequence of operations to give, for example, the red/black ordering version of the MIC preconditioner. However, experiments and analyses have

shown that these more parallel preconditioners in general have considerably slower convergence rates compared to their more sequential counterparts. The question is whether this gain in the amount of parallelism can compensate for the extra iterations needed due to the loss in the convergence rates. The tradeoffs between convergence rates and the amount of parallelism in a preconditioner pose a tremendous challenge to researchers in search for better preconditioners. The paper [90] has an excellent survey on the performance of many preconditioners on vector and parallel computers. This chapter addresses the implementation of preconditioners on one particular parallel single instruction multiple data (SIMD) computer, namely, the CM.

A number of preconditioners have been implemented on the CM using *Lisp (one of the few languages supported on the CM) by the author and the results are presented in [102]. This chapter presents similar results but with more efficient implementation using the low level assembly language called PARIS. Among the preconditioners implemented are : Relaxed Incomplete LU (RILU) with natural ordering, m-step Neumann series and m-step least square polynomial preconditioners. The basic conclusion is that the time performance of these preconditioners on massively parallel computers such as the CM are no better than the conjugate gradient method without preconditioning. In section 2, the basic conjugate gradient method, different types of orderings, as well as the detailed formulation of the preconditioners are covered. In section 3, details of implementation are presented. In section 4, the iteration counts as well as CM execution times and MFLOPS counts achieved on the CM for different preconditioners will be presented and observations will be discussed.

2.2 Preconditioned Conjugate Gradient Methods

2.2.1 The Conjugate Gradient (CG) Algorithm

The CG algorithm for the solution of a large sparse symmetric positive definite linear system of equations

$$Au = f$$

where A is an $N \times N$ symmetric positive definite matrix and u and f are $N \times 1$ vectors, is given as follow :

$r = f - Au$; initial residual

$p = 0$

Repeat

$z = M^{-1}r$; preconditioning

$\beta = new < r, z > / old < r, z >$

$p = z + \beta p$; updating direction

$\alpha = new < r, z > / < p, Ap >$

$u = u + \alpha p$; updating solution

$r = r - \alpha Ap$; updating the residual

until $\| r \|_2 / \| r_0 \|_2 \leq tol$

where $< \cdot, \cdot >$ denotes the usual Euclidean inner product, and r and p are $N \times 1$ residual and search direction vectors respectively.

The matrix M is called the preconditioning matrix and the speed with which the algorithm converges depends strongly on the choice of M. It is desirable to have M approximating A so that the condition number $\kappa(M^{-1}A)$ is smaller than that of A alone, that M is also sparse, and that the computational overhead to

solve the system of equations

$$Mz = r$$

is relatively small. With the advent of massively parallel systems, it is also desirable to have high degree of parallelism inherent in the preconditioners. The parallel implementation issues for both the conjugate gradient iteration and different preconditioners are discussed in later sections.

2.2.2 Model problems and orderings

The model problems used in this experiment are the two- and three-dimensional Poisson equations as well as the following two-dimensional second-order self-adjoint variable coefficient boundary value problem:

$$-\frac{\partial}{\partial x}(a(x,y)\frac{\partial u}{\partial x}) - \frac{\partial}{\partial y}(b(x,y)\frac{\partial u}{\partial y}) = f(x,y) \quad in \quad \Omega = [0,1]^2$$

with $u(x,y) = g(x,y)$ for (x,y) on the boundary $\partial\Omega$. We assume $a(x,y)$, $b(x,y)$ and $f(x,y)$ are smooth functions and $a(x,y)$ and $b(x,y)$ are nonnegative. The 5-point finite difference approximation of this equation on a $n \times n$ uniform grid defined inside the domain gives rise to the difference equations

$$\alpha_1 u_{j+1,k} + \alpha_2 u_{j-1,k} + \alpha_3 u_{j,k+1} + \alpha_4 u_{j,k-1} - \alpha_0 u_{j,k} = f_{j,k} h^2 \quad j,k = 1,\cdots,n, \quad h = \frac{1}{n+1},$$

with

$$\alpha_1 = a_{l+1/2,m}, \quad \alpha_2 = a_{l-1/2,m} \quad \alpha_3 = b_{l,m+1/2}, \quad \alpha_2 = b_{l,m-1/2},$$

and

$$\alpha_0 = \sum_{i=1}^{4} \alpha_i$$

where $u_{j,k}$ is used to approximate the value of $u(jh,kh)$. A collection of $n \times n$ difference equations, together with the way that the grid points are ordered, form the coefficient matrix A.

The ordering of grid points on a two-dimensional grid determines the form of the coefficient matrix A and also that of the preconditioners. Using the natural ordering, grid points are ordered in row-wise (or column-wise) manner. And using the red/black ordering, grid points are first partitioned into red and black groups such that a grid point (j,k) is red if $j+k$ is even and black if it is odd. Then the grid points within the red group are ordered using natural ordering, followed by the ordering of the grid points within the black group. In the context of parallel computation, we are interested in maximizing the number of operations at grid points among which there is no data dependence. Consequently, these operations can be performed in parallel and the ordering for these grid points does not affect the final result. For example, in the solution of the linear system $Au = f$, it is appropriate to consider the following natural and red/black orderings:

Diagonal ordering (parallel version of natural ordering) :

$$(j,k) < (m,n) \quad if \quad j+k < m+n,$$

Parallel version of red/black ordering :

$$(j,k) < (m,n) \quad if \quad (j,k) \text{ is red and } (m,n) \text{ is black,}$$

where the order of updates during preconditioning is determined by the inequality condition (e.g. in ascending or descending order). These two orderings for the grid points on a uniform 6×6 square grid are illustrated in Figure 2.1. Note that the same ordering number is assigned to grid points (j,k) with the same $j+k$ in the natural ordering and grid points of the same color in the red/black ordering. This implies that operations at these grid points can be performed in parallel. The red/black ordering is more attractive than the natural ordering in parallel computation, as far as the computation time per iteration is concerned,

since it takes two steps to sweep all grid points while the natural ordering takes $O(n)$ steps. Nevertheless, the convergence rate of some iterative algorithms may be slowed down by changing from natural ordering to red/black ordering as analyzed in [76].

Figure 2.1: (a) Diagonal and (b) parallel red/black orderings

5	6	7	8	9		1	2	1	2	1
4	5	6	7	8		2	1	2	1	2
3	4	5	6	7		1	2	1	2	1
2	3	4	5	6		2	1	2	1	2
1	2	3	4	5		1	2	1	2	1
		(a)						(b)		

For the diagonal ordering, the grid points that have the same sum $j + k$ can be performed in parallel, and the same is true for the red/black ordering where the grid points are of the same color. This means that if this problem is solved on a parallel computer, then a sweep (or one iteration) using red/black ordering can at best be computed in constant time (independent of the number of grid points) while a sweep using diagonal ordering can at best be computed in $O(n)$ time on a $n \times n$ grid. Here we can see that the parallel red/black ordering offers higher degree of parallelism ($O(n^2)$ operations can be performed in parallel) than the diagonal ordering (which has degree of parallelism $O(n)$). Nevertheless, the convergence rates improvement of the preconditioners using diagonal ordering are usually better than that using parallel red/black ordering.

2.2.3 Survey of Preconditioners

The preconditioners implemented in this experiment are described in the following subsections. These include most of the point versions of the popular ones.

2.2.3.1 Incomplete Factorization and SSOR Preconditioners

This class of preconditioners uses an incomplete Cholesky factorization of the matrix A as a preconditioner [6, 7, 39, 52, 86, 89]. If LL^T is the Cholesky decomposition of the symmetric positive definite matrix A where A is sparse, then the factor L is generally much denser than A because of fill-in. By an incomplete Cholesky factorization we mean a relation of the form

$$A = LL^T + R$$

where L is lower triangular and $R \neq 0$. One way to obtain such an incomplete factorization is to suppress the fill-in, or part of it, that occurs during the Cholesky decomposition. Some examples in this class are the incomplete Cholesky (IC(k)), modified IC (MIC(k)), and relaxed IC (RIC(k)) preconditioners. The parameter k denotes the amount of fill-in allowed in the factorization. For example, if we desire not to compute any element of L in positions corresponding to zero elements of A, then we obtain the IC(0), MIC(0) and RIC(0) preconditioners. For the rest of this chapter, we limit ourselves to the case $k = 0$.

The idea of modifying an incomplete factorization to improve convergence dates at least back to the work of Dupont, Kendall and Rachford [39]. A related idea was given by Gustafsson [52], who called the method modified ICCG method (MICCG). The splitting $A = LL^T + R$ includes a residual matrix R. While IC preconditioner ignores the contribution of R, the modified IC (MIC) preconditioner

compensates for the contribution of R by adding them to the diagonal entries. The MIC preconditioner, in addition, has the property that the row sums of LL^T is equal to those of A. The modified method can be formulated in terms of a parameter α that ranges from zero (unmodified IC) to one (MIC). The rates of convergence are often optimal for an α slightly less than one [6, 11]. For α other than zero and one, we call the method relaxed IC (RIC) preconditioner. IC(0), MIC(0) and RIC(0) preconditioners can all be represented in the following matrix form:

$$M = LL^T$$

where

$$(M)_{ij} = (A)_{ij} \quad \forall i \neq j \quad and \quad (A)_{ij} \neq 0.$$

Thus, the L's for these three preconditioners have the same sparsity and off-diagonal elements. The only difference is in the diagonal entries.

The algorithm to compute L for the class of incomplete factorization with $k = 0$ can be summarized as follow [89]:

$l_{11} = a_{11}^{1/2}$

For $i = 1 \quad to \quad N$

$s_i = 0$

end for

For $i = 2 \quad to \quad N$

For $j = 1 \quad to \quad i - 1$

if $a_{ij} = 0$ **then** $l_{ij} = 0$ **else**

$l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}\right) / l_{jj}$

$s_i = s_i + \sum_{l_{jk} \neq 0} \mid l_{ij} l_{jk} \mid$

end if

end for

$l_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 + \beta s_i\right)^{1/2}$

end for

The parameter β is the relaxation parameter. The values of β are 0, 1, and otherwise for IC(0), MIC(0) and RIC(0) preconditioners respectively. There are other equivalent forms of these incomplete factorization methods. For example, instead of using the incomplete Cholesky factorization, one can use the incomplete LU or LDL^T factorization. The incomplete Cholesky decomposition has an advantage that it requires less storage compared to the others. However, it also has the disadvantages that the square root operation is required during factorization and an additional arithmetic operation is required during preconditioning. Therefore, it is sometimes more desirable to use the root-free forms. We use the incomplete LU factorization in our implementation.

The symmetric successive overrelaxation (SSOR) preconditioner [7] has the following matrix form:

$$M = \frac{1}{2-\omega}(\frac{1}{\omega}D - C_L)(\frac{1}{\omega}D)^{-1}(\frac{1}{\omega}D - C_U)$$

where C_L and C_U are lower and upper triangular matrices respectively and

$$A = C_L + C_U,$$

and ω is the relaxation parameter. The SSOR preconditioner can be written in the form:

$$M = LU$$

where L and U have the same sparsity as the L and U matrices in the LU form of the incomplete factorizations with $k = 0$. This means that ILU(0), MILU(0),

RILU(0) and SSOR exhibit the same data communication pattern given a linear system. For the two-dimensional Poisson problem on a uniform grid, for example, we can represent the local operators $L_{j,k}$, $U_{j,k}$ and $M_{j,k} = L_{j,k}U_{j,k}$ for SSOR preconditioner on grid point (j,k) in stencil form as found in Figure 2.2.

Each of the preconditioners described above consists of a forward solve (L^{-1}) followed by a backward solve (U^{-1}). As shown in Figure 2.2 for the Laplacian operator, since every grid point updates its data based on the data from grid points to its west and south for the L-solve, the amount of parallelism ranges from 1 to n (refer to the definition of diagonal ordering given previously). The U-solve also has $O(n)$ degree of parallelism. Due to the sequential nature of the these diagonally-ordered preconditioners, the forward and backward solves can at best be performed in $O(n)$ time on parallel computers.

For certain problems arising from the discretization of partial differential equations, we can obtain more parallelism by using multicolor orderings of the points [91]. For the two-dimensional problems we consider in this chapter, for example, we can reorder the equations using red/black ordering so that

$$\hat{A} = \begin{bmatrix} D_R & C^T \\ C & D_B \end{bmatrix}$$

where D_R and D_B are diagonal and C is sparse. Consider the incomplete factorization of $\hat{A}$ with $k = 0$ denoted by:

$$M = \begin{bmatrix} I_R & 0 \\ L_{BR} & I_B \end{bmatrix} \begin{bmatrix} D_R & U_{RB} \\ 0 & \hat{D}_B \end{bmatrix} = \begin{bmatrix} D_R & U_{RB} \\ L_{BR}D_R & L_{BR}U_{RB} + \hat{D}_B \end{bmatrix}$$

where I_R and I_B are identity matrices for the red and black points, $\hat{D}_B$ is a diagonal matrix, and L_{BR} and U_{RB} have the same sparsity as C and C^T respectively. Again,

the incomplete factorization preconditioners M's are defined such that

$$(M)_{ij} = (A)_{ij} \quad \forall i \neq j \quad and \quad (A)_{ij} \neq 0.$$

Equating the entries of M and $\hat{A}$ based on the definition of different incomplete factorization preconditioners, we obtain

1. red/black ILU(0)

$$L_{BR} = CD_R^{-1}, \quad U_{BR} = C^T, \quad and \quad diag[\hat{D}_B] = diag[D_B - L_{BR}U_{RB}].$$

2. red/black MILU(0)

$$L_{BR} = CD_R^{-1}, \quad U_{BR} = C^T, \text{ and } diag[\hat{D}_B] = \text{ row sums of } D_B - L_{BR}U_{RB}.$$

3. red/black RILU(0)

$$L_{BR} = CD_R^{-1}, \quad U_{BR} = C^T, \text{ and}$$

$$diag[\hat{D}_B] = diag[D_B] + \beta \times \text{ row sums of } L_{BR}U_{RB}.$$

The red/black SSOR preconditioner has the same matrix form as before, namely,

$$M = \frac{1}{2-\omega}(\frac{1}{\omega}D - C_L)(\frac{1}{\omega}D)^{-1}(\frac{1}{\omega}D - C_U)$$

but now the matrices D, C_L and C_U have been permuted to conform to the red/black ordering.

Again, this preconditioning consists of a forward solve (L^{-1}) followed by a backward solve (U^{-1}). In this case, however, since all the red points can be updated in parallel and so can all the black points, the solves can at best be done in constant time. Again, for the two-dimensional Poisson problem on a uniform grid, for example, we can represent the local operators $L_{j,k}$, $U_{j,k}$ and $M_{j,k} = L_{j,k}U_{j,k}$ for SSOR preconditioner with red/black ordering on grid point (j,k) in stencil form as found in Figure 2.2.

Figure 2.2: Stencils of local operators for SSOR for 2D Poisson Problem

$$\begin{array}{ccc} & -\frac{\omega}{4} & \\ -\frac{\omega}{4} & 1 & \\ & -\frac{\omega}{4} & \end{array} \qquad \begin{array}{ccc} & 1 & \\ -\frac{\omega}{4} & & -\frac{\omega}{4} \\ & 1 & \end{array} \qquad \begin{array}{ccc} \frac{\omega^2}{16} & -\frac{\omega}{4} & \\ -\frac{\omega}{4} & 1+\frac{\omega^2}{8} & -\frac{\omega}{4} \\ & -\frac{\omega}{4} & \frac{\omega^2}{16} \end{array}$$

$$L_{j,k} \qquad U_{j,k} \qquad M_{j,k}$$

(a) diagonal ordering

(j,k) red :

$$\begin{array}{ccc} & -\frac{\omega}{4} & \\ 1 & & -\frac{\omega}{4} \\ & -\frac{\omega}{4} & \end{array} \qquad \begin{array}{ccc} & -\frac{\omega}{4} & \\ -\frac{\omega}{4} & 1 & -\frac{\omega}{4} \\ & -\frac{\omega}{4} & \end{array}$$

$$L_{j,k} \qquad U_{j,k} \qquad M_{j,k}$$

(j,k) black :

$$\begin{array}{ccc} & -\frac{\omega}{4} & \\ -\frac{\omega}{4} & 1 & -\frac{\omega}{4} \\ & -\frac{\omega}{4} & \end{array} \qquad 1 \qquad \begin{array}{ccc} \frac{\omega^2}{8} & -\frac{\omega}{4} & \frac{\omega^2}{8} \\ \frac{\omega^2}{16} & -\frac{\omega}{4} & 1+\frac{\omega^2}{4} \\ \frac{\omega^2}{8} & -\frac{\omega}{4} & \frac{\omega^2}{8} \end{array} \qquad \begin{array}{ccc} & \frac{\omega}{16} & \\ -\frac{\omega}{4} & & \frac{\omega^2}{16} \\ & \frac{\omega}{16} & \end{array}$$

$$L_{j,k} \qquad U_{j,k} \qquad M_{j,k}$$

(a) parallel red/black ordering

2.2.3.2 Polynomial preconditioners

1. m-step Jacobi preconditioner [5, 2, 35, 65, 89, 92]

   Let A be a square nonsingular matrix and $A = P - Q$ a splitting matrix of A such that P is nonsingular and the largest eigenvalue in magnitude of $P^{-1}Q$ is less than 1. Then, with $B = P^{-}Q$ [89],

   $$A^{-1} = (\sum_{k=1}^{\infty} B^k)P^{-1}.$$

   We can derive a preconditioner which is an approximation to A by

   $$M_{P(m)}^{-1} = (I + B + B^2 + \cdots + B^{m-1})P^{-1},$$

   which gives the m-step Jacobi PCG method. For the point Jacobi method, P is the diagonal of A.

2. Parametrized polynomial preconditioner

   In general, we can consider the polynomial preconditioner as

   $$M_{GP(m)}^{-1} = \sum_{l=0}^{m-1} \gamma_l B_l,$$

   so that the coefficients $\gamma_l, 0 \leq l \leq m$, can be chosen to minimize a certain objective function. Examples of such preconditioners are the least-squares polynomial preconditioner and the min-max polynomial preconditioner [2, 65]. Only the least-squares preconditioners with $m = 2, 3$ and 4 will be implemented here.

3. Other polynomial preconditioners [2]

   The same idea for parametrized polynomial preconditioners can be applied to a different splitting - for example, the SSOR splitting. This gives rise

to the m-step SSOR preconditioners. A variation of this is the multi-color m-step SSOR preconditioner. No implementation for this preconditioner is included here.

The polynomial preconditioners formulated above are very good candidates for parallel computation. If the P matrix is the identity matrix, then the $m-1$ steps for the m-step Jacobi preconditioning amount to $m-1$ iterations of the basic Jacobi method followed by accumulating the results of the iterations. As the basic Jacobi method offers a very high degree of parallelism (all grid points can be updated at the same time), so does this m-step Jacobi preconditioner. Thus, on massively parallel computer systems such as the CM, the m-step Jacobi preconditioning takes $O(m)$ time.

2.2.3.3 Other preconditioners

The preconditioners we have discussed so far represents the more popular ones. There are still other preconditioners which we have not discussed here. Examples are the use of fast Poisson solvers based on the fast Fourier Transform and the ADI iteration as preconditioners [83]. A few other preconditioners which has gained some attention in recent years is diagonally-scaled or incomplete Cholesky factorization preconditioners applied to reduced systems [16]. When a given matrix has property A [110], it can be reduced to a matrix which involves about half of the original unknowns and has better condition number. The description of these methods as well as their performance on the Alliant have been reported in [16]. The use of the reduced system method on the Connection Machine has the potential of speeding up the solution time. However, it also introduces some issues concerning slightly irregular communication pattern during the matrix vector mul-

tiplication. For this reason, we have not implemented these methods on the CM and we hope to assess the efficiency of these method on single instruction multiple data machines in the future.

2.2.3.4 Convergence Rates

The convergence rates of the conjugate gradient method with different preconditioners for the model Poisson problem, which depend on both the corresponding condition number as well as the distribution of the eigenvalues of the preconditioned system, can be studied either by matrix iterative analysis [10, 89, 39] or by Fourier analysis [26, 28, 75]. A summary of the Fourier analysis results is listed in Table ch2:tb0 which can also be found in [28] (in the tables h is grid size and the number of grid points is related to h via $N \approx h^{-2}$ for on 2D grids).

Table 2.1: Comparison of Condition Numbers

preconditioner	Natural	red/black
none(Laplacian)	$O(h^{-2})$	$O(h^{-2})$
ILU	$O(h^{-2})$	$O(h^{-2})$
MILU	$O(h^{-1})$	$O(h^{-2})$
SSOR	$O(h^{-1})$	$O(h^{-2})$
polynomial	$O(h^{-2})$	$O(h^{-2})$

Table 2.1 basically shows that while the red/black and polynomial preconditioners have high degree of parallelism, they often do not improve the order of the condition numbers, compared with the preconditioners using natural ordering. The effective preconditioners such as the MILU and SSOR with natural ordering

are, however, much more sequential in nature.

2.3 Implementation

2.3.1 The Connection Machine

The detailed description of the Connection Machine can be found in [61, 80]. The CM used in the present experiment is a 16k-node CM-2. The language used for program development was PARIS, an assembly language for the CM. During the experimental phase, it was found that if single-precision floating point numbers were used, there was considerable discrepancy between the recursively computed residual and the actual residual. One possible explanation is the loss of the orthogonality of the computed Krylov basis vectors due to finite precision arithmetic [41]. However, for the purpose of better performance, majority of the experiments use single-precision.

2.3.2 Processor mapping

The 2-D model problem can be nicely mapped onto the 2-D NEWS grid of the CM using binary reflected Gray code when n is a power of two. Depending on the ratio of the number of grid points to the number of available physical processors, each physical processor simulates one or more grid points. To run a 128×128 grid problem, the following configuration command in PARIS can be used:

```
dimension_array(1:2)=128
geometry_id = cm_create_geometry(dimension_array,2)
vpset = cm_allocate_vp_set(geometry_id)
call cm_set_vp_set(vpset)
```

In this case, since there are altogether $128 \times 128 = 16384$ grid points and we have 16384 (16k) physical processors, each physical processor performs the computations for a single grid point. Suppose 65336 (256×256) grid points are to be simulated but only 16k physical processors are available, then each physical processor has to take the computation load of 4 grid points. By using the virtual processing capability of the CM, this mapping (the mapping of 4 grid points to one physical processor) is transparent to the users and is performed automatically after the geometry has been set up.

An advantage of mapping the problem on the 2D NEWS grid is that the neighboring grid points are mapped to neighboring processors; and since the communication speed between neighboring processors using the NEWS communication is very fast, and that most of the communications are between local grid points, good execution time performance can be expected.

One major concern is how the boundary grid points are handled. Since no computation is needed for the boundary grid points other than providing data to their neighbors, one way is not to map them to any processors. During computation, the processors that simulate the grid points which are located next to the boundary grid points will be performing slightly different tasks from the other interior processors. An example is the local operation $A_{j,k}$ which requires fetching data from four neighbors (north, south, east, and west). Since these next-to-boundary grid points have one or two neighbors missing (e.g. the grid point at the northeast corner does not have east and north neighbors), they have to execute this operator a little differently. Because the CM is a SIMD machine and cannot execute two different active operations simultaneously, the updating of interior and next-to-boundary grid points have to be done in two separate steps, resulting in

longer execution time. Another way is to map boundary grid points also to actual processors. The drawback to this scheme is that these boundary processors will be idle most of the time. However, since the operations to be performed on all interior processors will be identical, the updating takes only one step, resulting in shorter execution time compared to the first scheme. This latter scheme is chosen for our implementation for reasons that it is simple to implement and it will probably give better performance.

2.3.3 The Preconditioned Conjugate Gradient Method on the CM

2.3.3.1 Implementation of a Conjugate Gradient iteration

One iteration of the PCG method requires 3 inner products (including the residual calculation), 3 multiply-and-add operations, 1 matrix-vector product calculation, 2 scalar divisions, one comparison, plus the computations required for preconditioning. Let's look at how each of these operations is performed on the CM.

1. Linked Triad (SAXPY)

   This operation is in the form of $y = ax + b$ where x and b are vectors and a is a scalar. If each processor takes care of one element in the vector and the scalar a is supplied by the host system, then the following PARIS code can be used

   call cm_f_mult_const_add_1l(y,x,a,b,ml,el).

   Here 'ml' and 'el' are integers specifying the lengths of the mantissa and exponent for the floating point operands. The time to perform this operation depends only on the virtual processor ratio (number of grid points per phys-

ical processor) and this ratio depends on the total number of grid points and the total number of available processors. For a particular virtual processor ratio (VP ratio), this operation takes constant time.

2. Matrix-vector product

   The matrix in this case is A, which, when operating on a vector, is equivalent to the parallel execution of the local operator $A_{j,k}$ (defined previously) on each element of the vector. For the Poisson problem with 5-point discretization, each processor adds the data fetched from the processors to its north, south, east, and west, divides the sum by 4, and subtracts the quotient from its own data. In PARIS code, it can be represented as

   ```
   call cm_f_get_from_news_1l(tmp,src,0,0,wl)
   call cm_f_news_add_2_1l(tmp,src,0,1,ml,el)
   call cm_f_news_add_2_1l(tmp,src,1,0,ml,el)
   call cm_f_news_add_2_1l(tmp,src,1,1,ml,el)
   call cm_f_multiply_constant_3_1l(dest,tmp,0.25,ml,el)
   ```

   Here 'wl' is the length of the data (number of bits). The third and fourth arguments to the 'cm_f_news_add_1l' function specify which dimension and direction to get the data. For example, if both are 0, then processor (i, j) will get the data from processor $(i-1, j)$. Again, this operation can be done in constant time for a particular VP ratio.

3. Inner Product

   The inner product has been known as a bottleneck to the performance of the PCG method. It is important that this inner product operation can be done

efficiently. It can be observed that the hypercube configuration of the CM helps to speed up this computation. It allows multiplication to be done in parallel in all processors, and the the partial sums are accumulated in the form of a binary tree. The PARIS code for inner product calculation is

```
call cm_f_multiply_3_1l(tmp,src,src,ml,el)
innerproduct = cm_global_f_add_1l(tmp,ml,el)
```

4. Others

   Other computational needs include 2 scalar divides and 1 comparison for convergence. These computations are performed on the front-end computer and require constant time.

In summary, without considering the preconditioning, the overall computation time for each iteration on the CM is dominated by the inner product operation. Even though the parallel computational complexity of the inner product computation is $O(\log N)$ where N is the number of grid points, it can be seen later that the performance of the inner product calculation on the CM is comparable to the other operations such as the linked triad operation.

2.3.3.2 Implementation of preconditioners

For preconditioning, depending on whether diagonal ordering or red/black ordering is used, the way to map the preconditioning algorithms on the CM and the order of computation times can be quite different.

1. Implementation of preconditioners with diagonal ordering[17, 89, 93]

Assuming that grid points (1,1) and (n,n) are located at the lower left and upper right corners of the domain respectively, then the L-solve starts at the grid point (1,1) and updates one diagonal at a time until grid point (n,n) is reached, where n is the number of interior grid points in each dimensions. This constitutes a wavefront moving from the lower left corner to the upper right corner and is called a forward sweep. During the forward sweep, each active processor (those processors located on the wavefront) updates its grid point value by averaging with the corresponding variables fetched from its south and west neighbors. The PARIS code is (assume the context flag has been set for the active processors) :

```
call cm_f_news_mult_3_1l(tmp,src,west,0,0,ml,el)
call cm_f_subtract_3_1l(dest,src,tmp,ml,el)
call cm_f_news_mult_3_1l(tmp,src,south,1,0,ml,el)
call cm_f_subtract_2_1l(dest,tmp,ml,el)
call cm_f_multiply_2_1l(dest,center,ml,el)
```

where 'west', 'south' and 'center' are weighting factors corresponding to the entries in the lower triangular L matrix (recall $M = LU$). The backward sweep (or the U-solve) can be performed similarly except that now the sweep starts at grid point (n,n) and proceeds to grid point (1,1).

The way to select a diagonal of grid points to be updated and leave the other processors idle can be achieved by the use an address variable. This variable is initially set to the sum of the x and y coordinates for each processor. During preconditioning, the processors with address variable equal to 1 is updated first, followed by 2, 3 and so on. For diagonal ordering, since there

are $O(n)$ diagonals in a 2-D grid, the corresponding preconditioning takes $O(n)$ time.

It should also be mentioned that the Eisenstat trick [40, 89] , when applied to a preconditioner having the form:

$$M = ST^{-1}S^T,$$

can help to reduce the amount of computation. Since the incomplete factorization and the SSOR preconditioners all have the above form, the Eisenstat trick can be applied. We have not implemented the Eisenstat trick in our experiment. The savings due to the use of the Eisenstat trick is in the matrix vector multiplication of the conjugate gradient iteration, and the L- and U-solve are still required. As we shall see, for incomplete factorization and SSOR preconditioners with diagonal ordering on the CM, most of the time is spent in preconditioning. Therefore, we conclude that the use of the Eisenstat trick should not improve the performance on the CM significantly.

2. Implementation of preconditioners with red/black ordering

   The preconditioners with red/black ordering can be implemented using two parallel boolean flags, namely the red and the black flags, which indicate whether the corresponding grid point is red or black point. For red/black ordering, since all the red points can be updated in parallel, and so are the black points, the corresponding preconditioning only takes constant time. This technique is used on the implementation of ILU, MILU and SSOR preconditioners with red/black ordering.

3. Implementation of polynomial preconditioners

During each step of the m-step Jacobi preconditioning (for the Poisson problem), all processors corresponding to the interior grid points are active and they all fetch data from their north, south, east and west neighbors by performing

```
call cm_get_from_news_1l(dest,src,0,0,wl)
call cm_f_news_add_2_1l(dest,src,0,1,ml,el)
call cm_f_news_add_2_1l(dest,src,1,0,ml,el)
call cm_f_news_add_2_1l(dest,src,1,1,ml,el)
call cm_f_multiply_constant_2_1l(dest,0.25,ml,el)
```

and this is to be performed $m-1$ times and the 'dest' is to be accumulated. With a fixed VP ratio, the polynomial preconditioning takes $O(m)$ time.

2.3.4 Experiments

The PCG methods are applied to the 2D and 3D Poisson problem and also a 2D variable coefficient problem. Since an extensive set of numerical experiments have been reported in [102], only a few of them are to be repeated here. The ones selected represent the ones that give the most amount of parallelism (namely, the polynomial preconditioners) and the one that gives the best convergence rate (namely the RILU with natural ordering). The preconditioners used are :

- Conjugate gradient method without preconditioning (CG)
- RILU with diagonal ordering (RILU diagonal)
- 2-step Jacobi preconditioner (Jacobi-2)
- 4-step Jacobi preconditioner (Jacobi-4)

- 2-term least-squares polynomial preconditioner ($\gamma_0 = 7/6, \gamma_1 = 5/6$) (LS2)
- 3-term least-squares polynomial preconditioner ($\gamma_0 = 35/32, \gamma_1 = 50/32, \gamma_2 = 35/32$) (LS3)
- 4-term least-squares polynomial preconditioner ($\gamma_0 = 37/40, \gamma_1 = 49/40, \gamma_2 = 91/40, \gamma_3 = 63/40$) (LS4)

For each of the experiments, the following things are to be observed :

- the iteration counts to achieve convergence,
- the execution time on the CM, and
- MFLOPS achieved on the CM.

2.3.4.1 Experiment 1

A 2D Poisson problem is solved with $f = 0$ and initial guess $u_0 = 1$. Various preconditioners listed above are used using $n = 128, 256, 512, 1024$. The stopping criterion used was $\| r_i \|_2 / \| r_0 \|_2 \leq 10^{-6}$.

2.3.4.2 Experiment 2

The 3D Poisson equation is solved with $f = 0$ and initial guess $u_0 = 1$ using $n = 32, 64$. The stopping criterion is the same as the one in experiment 1.

2.3.4.3 Experiment 3

The 2D variable coefficient problem used here is

$$\frac{\partial}{\partial x}\left(e^{-xy}\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(e^{xy}\frac{\partial u}{\partial y}\right) = f, \Omega = [0,1]^2,$$

where $u = xe^{xy} \sin \pi x \sin \pi y$

Zero initial condition is used and the stopping criterion is the same as in experiment 1.

2.4 Results and Discussion

2.4.1 Results

The iteration counts for experiment 1 are shown in Table 2.2. The corresponding CM cpu times are shown in Table 2.3.

Table 2.2: Iteration counts for experiment 1

Preconditioner	n = 128	n = 256	n = 512	n = 1024
CG	206	401	783	1525
RILU(diagonal)	29	40	54	not done
Jacobi-2	101	197	384	748
Jacobi-4	71	139	270	527
LS2	95	187	364	711
LS3	68	132	258	503
LS4	59	116	226	411

The MFLOPS for the PCG procedures in experiment 1 can be found in Table 2.4. To calculate the MFLOPS, only the standard floating-point arithmetic operations such as addition and multiplication are counted. Operations such as data copying, logic evaluations and data communication operations are ignored.

Also the iteration counts, the CM times and MFLOPS for experiments 2 are shown in Table 2.5 to 2.7. For experiment 3, only iteration counts and execution

Table 2.3: CM Execution Time (in seconds) for experiment 1

Preconditioner	n = 128	n = 256	n = 512	n = 1024
CG	0.29	1.23	6.6	40
RILU (diagonal)	8.69	57.1	447	not done
Jacobi-2	0.27	1.15	6.2	38.7
Jacobi-4	0.32	1.42	7.5	43.8
LS2	0.34	1.49	8.0	46.6
LS3	0.3	1.34	7.1	41.5
LS4	0.33	1.43	7.6	43.6

Table 2.4: MFLOPS on CM for the preconditioners for experiment 1

Preconditioner	n = 128	n = 256	n = 512	n = 1024
CG	221	406	589	760
RILU(diagonal)	1.6	1.4	1.0	not done
Jacobi-2	184	337	491	641
Jacobi-4	174	308	451	606
LS2	146	263	384	512
LS3	156	271	398	534
LS4	152	276	407	552

times are shown in Table 2.8.

Table 2.5: Iteration counts for experiment 2

Preconditioner	n = 32	n = 64
CG	66	130
RILU (natural)	17	22
Jacobi-2	33	65
Jacobi-4	24	46
LS2	30	60
LS3	22	43
LS4	18	37

Table 2.6: CM cpu time(in seconds) for experiment 2

Preconditioner	n = 32	n = 64
CG	0.35	3.43
RILU (natural)	8.61	65.8
Jacobi-2	0.34	3.32
Jacobi-4	0.47	4.35
LS2	0.45	4.36
LS3	0.43	4.05
LS4	0.43	4.29

Table 2.7: MFLOPS on CM for the preconditioners for experiment 2

Preconditioner	n = 32	n = 64
CG	142	229
RILU (natural)	2.5	3.3
Jacobi-2	121	195
Jacobi-4	107	177
LS2	87	144
LS3	91	150
LS4	93	154

Table 2.8: Iteration counts and CM times for experiment 3 (256x256)

Preconditioner	iteration count	CM time (sec)
CG	763	2.4
RILU (natural)	49	70
Jacobi-2	373	2.7
Jacobi-4	338	3.4

2.4.2 Connection Machine Statistics

To evaluate the performance of the PCG method on the CM, the execution time statistics are gathered on some basic operations of the PCG algorithm. These statistics are taken by averaging the CM cpu times from a few sample runs with each run performing the corresponding operations (on parallel variables) 1000 times. Table 2.9 shows that MFLOPS counts of the operations using single-precision floating point numbers while Table 2.10 shows the MFLOPS count when double-precision floating point numbers are used. The CM used to gather the statistics is the CM-2 with 16k nodes and with single-precision floating point hardware. The operations to be examined are:

- two-operand addition (ADD)
- two-operand multiplication - both operands are parallel variables (MULT)
- linked triad (SAXPY)
- inner product calculation (INNER)
- data communication using NEWS grid when distance = 1 (COMM)

It can be observed that single-precision floating point operations are much faster than the double-precision floating point operations. The reason is that single-precision floating point operations are performed in the floating-point hardware which is much faster than the single-bit processors. Another observation from the tables is that the inner product operation is relatively efficient compared to the other arithmetic operations. Also, the increase in VP ratio improves the MFLOPS performance most of the time. This behavior demonstrates the positive effect of virtual processing.

Table 2.9: CM MFLOPS for different types of operations (single-precision)

operation	n = 128	n = 256	n = 512	n = 1024
ADD	328	437	460	456
MULT	410	504	535	524
SAXPY	655	1008	1070	1003
INNER	172	437	718	846
COMM	3.5 Gb/s	6.0 Gb/s	7.8G b/s	11.6 Gb/s

Table 2.10: CM MFLOPS for different types of operations (double-precision)

operation	n = 128	n = 256	n = 512	n = 1024
ADD	8.2	13.1	14.6	15.4
MULT	5.5	6.0	6.2	6.3
SAXPY	7.4	8.9	9.0	9.0
INNER	4.3	5.8	6.3	6.5
COMM	3.9 Gb/s	6.3 Gb/s	9.3G b/s	13.2 Gb/s

2.4.3 Discussion

- The first observation about the running time performances is that there is little improvement in execution using the best preconditioner (2-step Jacobi, as of Table 2.3) over using CG alone while the other preconditioners take more time than the CG. There are a few reasons for this. One reason is that the performance of the inner product calculation is relatively efficient and so a basic conjugate gradient iteration becomes much more efficient than the preconditioners on the CM. As a result, for the grid sizes we look at on this machine, the gain in time performance due to faster convergence cannot compensate for the time loss due to preconditioning. Another reason is that the CG method requires only two inner product calculations as opposed to three for the preconditioned CG methods. In addition, the use of the polynomial preconditioners helps to reduce the iteration counts only by a small factors and consequently the saving in time due to lower iteration counts is not good enough to compensate for the overhead time involved in preconditioning.

- From Table 2.2, we can see that although the RILU preconditioner using diagonal ordering are very effective in reducing the iteration counts, their performance on the CM is very poor. The obvious reason is that the CM spends most of its time in preconditioning. The performance improves for 3D problems since the amount of parallelism for RILU is increased [17]. However, it is still not good enough. Thus, we can conclude that such preconditioners with diagonal ordering are not good candidates on fine-grain massively parallel computers such as the CM. However, it should be mentioned that if the CM is having mesh-connected network instead of the hypercube network,

the preconditioners using natural ordering may turn out to be competitive, since then the inner product time will be quite significant. Therefore, the choice of a "good" preconditioner for a machine depends also very much on the architecture of the machine.

- Even though the least-squares polynomial preconditioners give better convergence rate improvement, their performances on the CM are worse than the m-step Jacobi preconditioners. Comparing the 4-step Jacobi(Jacobi-4) preconditioner with the 4-step least-squares (LS4) preconditioner in Table 2.2, it can be observed that LS4 requires slightly lower iteration count. However, LS4 also requires more multiplications than Jacobi-4. It turns out that it is better to leave out the extra multiplications and perform more a few more iterations.

CHAPTER 3

Survey of Multilevel Preconditioners

3.1 Classification of Preconditioners

What we have concluded so far is that both convergence rate and degree of parallelism are essential factors for efficient implementation of preconditioners on massively parallel computers. This behavior can be traced back to the physical processes underlying the elliptic problems. As elliptic problems exhibit global coupling between different points in the domain, it is not surprising to know that the preconditioners that use only local updating will not be effective in improving convergence rates. Unfortunately, the more global preconditioners such as the RILU do not give high enough degree of parallelism. In this section, we summarize some common characteristics of the existing preconditioners. The result is a classification of all the preconditioners into three types as described in the following sub-sections.

3.1.1 Global Preconditioners Using Local Updates

This class of preconditioners consists of the MILU and its variants (ILU and RILU) as well as SSOR using diagonal ordering. After one iteration of preconditioning step, the update at a given point depends on every other points (even the far-away points) in the domain. Because of this global data exchange, this class of preconditioner is thus effective in speeding up the convergence. However, the overall process is achieved through many local updates (updates depending on

values of neighbors), and so this class is limited in the amount of parallelism.

3.1.2 Local Preconditioners Using Local Updates

This class of preconditioners consists of the MILU and its variants using parallel red/black ordering and the class of polynomial preconditioners. After one iteration of preconditioning step, the update at a given point depends only on its neighboring points, and as a result there is only little improvement in the convergence rates. Since the preconditioning step allows updating every point (or half of the points for red/black ordering) simultaneously, the amount of parallelism is high.

3.1.3 Global Preconditioners Using Global Updates

This class of preconditioners consists of the hierarchical basis [111], multigrid [24] and other multilevel preconditioners [22, 12, 13]. This class is characterized by a hierarchy of grids ranging from coarse to fine grids. The preconditioning involves data transfer between different grid level (for example, restriction and interpolation in multigrid methods) and/or local updates within each grid level. Because of the use of many grid levels with different coarseness, at the end of one iteration of preconditioning step, the update at a given points also depends on the points far away. Moreover, since local updates can be performed in parallel at each grid level, there is also relatively high degree of parallelism.

The condition number improvement (which is related to convergence rate) and the degree of parallelism for the different classes of preconditioners are summarized in Table 3.1 and 3.2 for two- and three-dimension problems (in the tables n is the number of unknowns in each dimension). Here the degree of parallelism is a measure of the amount of computations that can be performed concurrently

given unrestricted hardware resources. Hence, this measure depends on the data dependency inherent in the preconditioners. Parallel complexity measures how the total execution time depends on the number of unknowns n given unrestricted hardware resources. Lastly, sequential complexity measures the total amount of work to perform the preconditioning.

Table 3.1: **Characteristics of Different Classes of Preconditioners (2D)**

preconditioner	condition number	degree of parallelism	parallel complexity	sequential complexity
local-local	$O(n^2)$	$O(n^2)$	$O(n \log n)$	$O(n^3)$
global-local	$O(n)$	$O(n)$	$O(n^{1.5})$	$O(n^{2.5})$
global-global	$O(log^2 n)$	$O(n^2/\log n)$	$O(\log^2 n)$	$O(n^2 \log n)$

Table 3.2: **Characteristics of Different Classes of Preconditioners (3D)**

preconditioner	condition number	degree of parallelism	parallel complexity	sequential complexity
local-local	$O(n^2)$	$O(n^3)$	$O(n \log n)$	$O(n^4)$
global-local	$O(n)$	$O(n^2)$	$O(n^{1.5})$	$O(n^{3.5})$
global-global	$O(log^2 n)$	$O(n^3/\log n)$	$O(\log^2 n)$	$O(n^3 \log n)$

It is the excellent convergence rate and the relatively high degree of parallelism of the global preconditioners with global updates that motivate us to look into the class of multilevel preconditioners. However, it should be pointed out that efforts by other researchers in developing multilevel preconditioners are not

necessarily motivated by the same reason. In this and later chapters, we investigate the multilevel-type preconditioners and show that they outperform the other preconditioners on massively parallel machines such as the CM. Here we want to emphasize that even though the multilevel-type preconditioners give excellent convergence rates and high degree of parallelism, their performance depends also very much on the architecture of the machine. In particular, in order to obtain high degree of efficiency, an efficient interconnection network for global communication is required, such as the hypercube interconnection network on the CM.

3.2 Previous Work on Multilevel Methods

In this section, we briefly survey many multilevel methods that have been proposed in the past. We borrow the classification from Tuminaro's thesis [104] and groups the multilevel methods into three types : multigrid preconditioners, concurrent iteration multilevel methods and convergence acceleration multilevel methods.

3.2.1 Multilevel Preconditioners

These methods use preconditioners (within a conjugate gradient algorithm) that resemble a multigrid iteration. Specifically, these preconditioners mimic the grid levels by projections onto multiple grids or basis functions with different scales. One advantage of this approach is that since preconditioners need only roughly approximate A^{-1} to be effective, there is more freedom in constructing preconditioners.

3.2.1.1 Multigrid (MG) preconditioner

A natural choice for a multilevel preconditioner is to use a fixed number of cycles of a conventional multigrid method. This approach was explored early on in the development of multigrid methods [73, 74] to, for example, enhance the convergence rates for discontinuous coefficient problems [74] and to avoid the locking effect for elasticity problems [19]. The basic operations on each grid are interpolation, projection and smoothing operations, each of which can be easily designed to be highly parallelizable. For example, in the V-cycle strategy, each grid is visited exactly twice in each preconditioning step, once going from fine to coarse grids and once coming back from coarse to fine.

3.2.1.2 Hierarchical basis preconditioner (HB)

Another preconditioning technique of multilevel type is the hierarchical basis method [15, 111]. The name refers to the space of hierarchical basis functions defined on a grid hierarchy. Let the hierarchical basis functions be denoted by ψ_j^l, where l denotes the grid level and j the index of the basis function on that level ($\psi_j^l = 1$ at the jth grid point and 0 on the other grid points on level l). Then, the action of the inverse of the hierarchical basis preconditioner M on a function v can be written as

$$M^{-1}v = \sum_l \sum_j (v, \psi_j^l)\psi_j^l$$

which takes the discretized form $SS^T v_h$ and can be computed by a V-cycle with the matrix S^T (a change of basis from nodal to hierarchical) corresponding to a fine-to-coarse grid traversal and S (a change of basis from hierarchical to nodal) to a coarse-to-fine traversal. On each level, only local operations are performed. In 2D, the condition number of the preconditioned system can be shown to grow

like $O(\log^2 h^{-1})$, which is very slow. Unfortunately, this nice property is lost in 3D where the growth is $O(h^{-1})$ [87, 111]. However, these theoretical results are proven under much weaker regularity assumptions than for the multigrid methods. Moreover, the computational work per step is $O(h^{-1})$ even for highly nonuniform and refined meshes. For numerical experiments on parallel computers, see [3, 51].

3.2.1.3 Parallel multilevel preconditioner by Bramble-Pasciak-Xu (BPX)

Very recently, Bramble-Pasciak-Xu [22, 107] proposed the following preconditioner for second-order elliptic problems in R^d:

$$M^{-1}v = \sum_l h_l^{2-d} \sum_j (v, \phi_j^l)\phi_j^l,$$

where $\phi_j^l, l = 1, 2, \cdots$ are the nodal basis functions at grid level l ($\phi_j^l = 1$ at the jth grid point and 0 at the other points on level l) and h_l is measure of the mesh size at grid level l. Since the form of their preconditioner is similar to that for the hierarchical basis preconditioner, the computations can be arranged in a similar way via a V-cycle. They proved that the condition number of the preconditioned operator can be bounded by $O(\log h^{-1})$ for problems with smooth solutions, by $O(\log^2 h^{-1})$ for problems with crack type singularities, and by $O(\log^3 h^{-1})$ for problems with discontinuous coefficients. In 3D, this is a significant improvement over the hierarchical basis preconditioner.

3.2.1.4 Algebraic multilevel preconditioners (AMP)

Vassilevski [106] proposed a different approach to derive multilevel preconditioners. He used the standard nodal basis functions and a multilevel ordering of the nodes of the discretization, in which nodes at a given level belonging to a coarser grid are ordered after the other nodes. He then considered an approximate block

factorization of the stiffness matrix in this ordering, in which the Schur complement at a given grid level is approximated by iteration with the preconditioner of the stiffness matrix recursively defined at the previous level. He showed that, with one iteration at each level, the condition number of the preconditioned system can be bounded by $O(\log h^{-1})$. A similar method has been proposed by Kuznetsov [77]. Later, Axelsson-Vassilevski [12, 13] improved this bound to $O(1)$ by carrying out recursively more (Chebychev) iterations with the preconditioner at each level. Axelsson [9] also showed that the same technique can be applied when hierarchical basis functions are used instead of the nodal basis. Note that when the number of iterations at each level exceeds 1, the grid traversal differs from (and requires more work than) all the previously mentioned V-cycle based methods.

3.2.1.5 Wavelet Preconditioners (WP)

Recently a lot of attention has been paid to the development of wavelet-based numerical methods for solving partial differential equations [48, 96, 23]. Wavelets are new families of basis functions that yields the representation of a function

$$f(x) = \sum b_{jk} W(2^j x - k)$$

where $W(2^j x - k)$ generates the wavelet basis functions by translation and dilation. The construction of $W(2^j x - k)$ begins with the solution of $\phi(x)$ to a dilation equation

$$\phi(x) = \sum c_k \phi(2x - k).$$

Then W comes from ϕ, and the basis comes by translation and dilation of W. Some of the solution to the dilation equation are :

- box function

$$\phi(x) = \begin{cases} 1 & 0 < x \leq 1 \\ 0 & otherwise \end{cases}$$

with $c_0 = c_1 = 1$.

- hat function

$$\phi(x) = \begin{cases} x & 0 < x \leq 1 \\ 2 - x & 1 < x \leq 2 \end{cases}$$

with $c_0 = c_2 = 0.5$ and $c_1 = 1$.

- Daubechies function (D4) [34] with c_0 to c_3 as $0.25(1 \pm \sqrt{3})$, $0.25(3 \pm \sqrt{3})$.

Finally the wavelet is defined as

$$W(x) = \sum (-1)^k c_{1-k} \phi(2x - k).$$

The object of the wavelets is to localize as far as possible in both time and frequency, with efficient algorithms. It turns out that the "multiresolution analysis" properties of the wavelet basis are similar to the idea of multilevel methods.

So far emphases in the literature have been on the use of wavelet basis functions to accurately represent the solution functions. However, we can consider the wavelet basis as a particular form of multilevel basis and we propose to use some suitable wavelet bases as multilevel preconditioners. Such preconditioners, like the hierarchical basis and the multilevel nodal basis preconditioners, can be represented as

$$M_w^{-1} v = \sum_k c_k \sum_l (u, \psi_k^{(l)}) \psi_k^{(l)}$$

where c_k is a properly chosen scaling function for level k. The wavelet basis preconditioners for second-order self-adjoint elliptic problems are currently being studied.

3.2.1.6 Relationship among multilevel preconditioners

As can be seen from the discussion above, there are similarities among various multilevel preconditioners. Most of the multilevel preconditioners are in the form of a multigrid V-cycle (MG, HB, BPX, WP and MF, but not AMP). The MF preconditioner is very similar to the BPX method. The MF method allows some flexibility in the choice of filters (basically any multigrid residual averaging operator can be used) and does not depend on the use of a finite element discretization with nested nodal basis functions. It also allows a single grid (i.e. non-multigrid) version which may better suit massively parallel architecture computers. On the other hand, the finite element framework allows an elegant proof of the asymptotic convergence behavior for rather general problems as is done in [22, 107] whereas the filtering framework is rigorously provable for constant coefficient model problems only (although much more detailed information such as eigenvalue distribution can be obtained for them).

Finally, it is interesting to compare these preconditioners with the conventional multigrid method. Several of the preconditioners have the same form of a conventional multigrid cycle, except that the smoothing operations are omitted. For less regular problems where a good smoothing operator is hard to derive and could be quite expensive, one step of these preconditioners can be substantially less expensive than a corresponding step of the multigrid iteration. In a sense, one can view these preconditioners as efficiently capturing mesh size dependent part of the ill-conditioning of the elliptic operator and leaves the other sources of ill-conditioning (e.g. discontinuous coefficients) to the conjugate gradient iteration. The combination of multigrid and conjugate gradient holds the promise of being both robust and efficient. However, to get a spectrally equivalent preconditioner, it seems that

one must go beyond the V-cycle and perform more iterations on each grid as in the AMP method.

3.2.2 Concurrent Iteration Multilevel Methods

These methods are conceptually similar to conventional multigrid methods in that the original problem is approximated on a hierarchy of grids. The major difference is that instead of traversing all the levels sequentially, now all levels can be processed concurrently. This is accomplished by first distributing the original problem over all grids, performing relaxation sweeps on all levels concurrently, and then recombining the solutions. Examples in this type of multilevel methods are due to Gannon and Van Rosendale [46] and Greenbaum [50].

Since relaxations can be computed simultaneously, one iteration is faster than a single iteration of a standard multigrid method when many processors are available. Unfortunately, they have convergence rates somewhat slower than the corresponding standard multigrid method. Again, tradeoff has to be made to find which approach is more efficient on a particular parallel computer.

3.2.3 Convergence Acceleration Multilevel Methods

These methods accelerate the convergence of multigrid method by introducing more work executed concurrently with the conventional method. On parallel processors when this additional work is performed by the otherwise idle processors, the time per iteration is then about the same as a conventional multigrid iteration. The advantage of these methods is that fewer iterations are required for convergence. Thus, these methods are potentially faster than the standard multigrid method on parallel computers. Examples of this type of multilevel methods are

due to Chen/Sameh [30], Chan/Tuminaro [104], Douglas/Miranker [36], Frederickson/McBryan [45] and Hackbusch [54].

The primary emphasis in this dissertation is on the multilevel preconditioners. In particular, we emphasize on the multilevel filtering preconditioners by Kuo, Chan and Tong [76] as well as the parallel multilevel preconditioners by Bramble, Pasciak and Xu [22]. In our experiments on the Connection Machine, however, we also implement the parallel superconvergent multigrid (PSMG) algorithm by Frederickson/McBryan [45]. In addition, we should point out that the single grid multilevel filtering preconditioner proposed in this dissertation can be classified as a convergence acceleration multilevel method.

CHAPTER 4

Multilevel Filtering Preconditioning - analysis and experments

This chapter introduces a preconditioning technique which we call multilevel filtering (MF) preconditioning. In the next section, we use Fourier analysis to study the effect of MF preconditioners on !D, 2D, and 3D Poisson problems with zero boundary condition and on uniform grids. Fourier analysis is used here mainly as a tool to obtain insights into how to design different components (filtering, scaling) of our MF preconditioners in order to get better convergence rates. Even though the MF method is very restrictive from the analysis point of view, it can be applied to many general second-order self-adjoint problems. For example, in our numerical experiments, we have quite effectively applied the MF preconditioners to variable and discontinuous coefficient problems. In next chapter, we will also show how to apply the MF method to grids with local refinement as well as to higher order problems. Moreover, the MF method is very close to the multilevel nodal basis method proposed by Bramble, Pasciak, and Xu [22, 107] so that in some cases we can borrow the finite element theory from [22] to derive a condition number bound for our method.

4.1 Idea of MF Preconditioning

This section describes the basic ideas as well as a step-by-step design of the MF preconditioner. To simplify the analysis, we apply the preconditioner to a one-dimensional (1D) Poisson equation and analyze the condition number of the

preconditioned system. However, it should be emphasized that the excellent convergence rates offered by the MF hold also for higher dimensional cases (we have included a sub-section on the results of condition number analysis for two- and three-dimension cases).

4.1.1 Eigendecomposition of the 1D Laplacian

Consider the following 1D Poisson equation on $\Omega = [0, 1]$

$$-\triangle u = f(x) \tag{4.1}$$

subject to zero Dirichlet boundary conditions. The discretization of the above equation on rectangular grids with grid size h gives rise to a linear system equation denoted by $Au = f$ where A, u and f correspond to the discrete Laplacian, the solution and the forcing functions respectively. Clearly, A is a tridiagonal matrix with diagonal elements $-\frac{1}{h^2}, \frac{2}{h^2}, -\frac{1}{h^2}$. It is well known that the matrix A can be diagonalized as

$$A = W\Lambda W^T$$

where

$$(W)_{ij} = \sqrt{2}h \sin ij\pi h,$$

$$\Lambda = diag(\lambda_k) \ , \ \lambda_k = \frac{4}{h^2} \sin^2 \frac{k\pi h}{2}.$$

Here $\{\lambda_k\}$ is known as the spectrum of the discrete Laplacian. Using the above form, we can find the solution u by

$$u = A^{-1}f = W\Lambda^{-1}W^T f.$$

This procedure serves as the general framework for the fast Poisson solvers in higher dimensional (> 1) cases. The motivation of the multilevel filtering preconditioning

is to avoid the use of FFT but instead use a sequence of cheaper filtering operations to approximately achieve the desired spectral decomposition.

4.1.2 Spectral Band Decomposition of the 1D Laplacian

The construction of the MF preconditioner begins with grouping the eigenfunctions of A into subsets corresponding to different bands of frequencies. In matrix form we partition W so that

$$W = [W_1, W_2, \cdots, W_L], n = 2^L - 1$$

where W_1 and W_L correspond to the lowest and highest frequency bands respectively with

$$W_l = [W_{2^{l-1}}, \cdots, W_{2^l-1}].$$

(This band decomposition is the only decomposition allowed by the multilevel grid structure.)

Using the notations introduced above, we can rewrite

$$A = \sum_{l=1}^{L} W_l \Lambda_l W_l^T$$

where

$$\Lambda = diag(\Lambda_l), \Lambda_l = diag(\lambda_{2^{l-1}}, \cdots, \lambda_{2^l-1}).$$

Figure 4.1 shows the spectrum of the Laplacian divided into bands for $n = 256$. To eliminate the need for the use of FFT, we seek to approximate A as described in the following subsections.

4.1.3 Approximation 1 : Constant Band Eigenvalues

The first approximation comes in when we replace within each band the corresponding eigenvalues (Λ_l) by a constant c_l. Thus, we have a preconditioner M

such that

$$M^{-1}v = \sum_{l=1}^{L} \frac{P_l v}{c_l} \text{ where } P_l = W_l W_l^T.$$

Here $P_l = W_l W_l^T$ is a mapping from space domain to space domain. In addition, P_l can be derived from the original W by

$$P_l = W I_{P_l} W^T$$

where I_{P_l} is a diagonal matrix whose $(k,k)$the element is

$$I_{P_l} = \begin{cases} 1 & k \in band_l \\ 0 & otherwise. \end{cases}$$

and

$$band_l = \{k : 2^{l-1} \leq k < 2^l \text{ and } k \in I\}$$

Since P_l passes Fourier components in band $band_l$ and blocks components in other bands, it is called a bandpass filter.

Thus applying the preconditioner to a vector (i.e. $M^{-1}v$) consists of three phases : projection of v into the subspace corresponding to each band (operator B_l), scaling by the corresponding approximate eigenvalues c_l, and synthesizing the scaled components (summation). In the context of multirate signal processing, the decomposition of a function into several components, each of which is confined to a narrow wavenumber band (i.e. bandpass filtering) , is known as the filter bank analyzer and the reverse process is the filter band synthesis [76].

The preconditioned eigenspectrum can then be described by

$$\Lambda_{M^{-1}A} = \Lambda_M^{-1} \Lambda_A = diag(\frac{\lambda}{c_1}, \frac{\lambda_2}{c_2}, \frac{\lambda_3}{c_2}, \cdots, \frac{\lambda_{2^{l-1}}}{c_l}, \cdots, \frac{\lambda_n}{c_l}).$$

The question is to choose appropriate c_l's to reduce the condition number $\kappa(M^{-1}A)$. Suppose we can find c_l's so that

$$C_1 \leq \frac{\lambda_k}{c_l} \leq C_2, \ k \in band_l, \ 1 \leq l \leq L$$

where C_1 and C_2 are positive constants independent of h, then M and A are spectrally equivalent. The c_l's can be chosen based on the distribultion of eigenvalues for the problem. For the Laplacian considered in this discussion, we recall that

$$\lambda_k = \frac{4}{h^2}\sin^2\frac{k\pi h}{2} \quad k = 1, \cdots, n \quad n \approx \frac{1}{h}.$$

We can approximate λ_k's by eliminating the sin term, giving:

$$\lambda_k \approx \frac{4}{h^2}\frac{k\pi h}{2} = k^2\pi^2.$$

Using this relationship and the wavenumber band partitioning defined previously, we obtain the following approximate eigenvalue distribution for $band_l$:

$$\Lambda_l \approx \pi^2[2^{2l-2}, \cdots, (2^l - 1)^2].$$

We observe here that the eigenvalues in $band_l$ behaves approximately like 2^{2l} and so we can construct c_l's such that

$$c_l = c2^{2l}.$$

In general, we use

$$c_l = c2^{\beta l}$$

where β is the order of the partial differential equation (We will see in next chapter that the MF preconditioners can be applied to fourth-order problems by changing this scaling factor accordingly.)

The condition number of the preconditioned spectrum for each band is then bounded by

$$4^{L-l}\left[1 - \cos(2^{-L+l-1}\pi)\right] \leq \frac{\Lambda_l(k,k)}{c_l} < 4^{L-l}\left[1 - \cos(2^{-L+l}\pi)\right],$$

for $k \in band_l$. The largest and smallest eigenvalues of $M^{-1}A$ are bounded respectively

$$\Lambda_{max}(M^{-1}A) < \max_{1 \le l \le L} 4^{L-l} \left[1 - \cos(2^{-L+l}\pi)\right] < \frac{\pi^2}{2},$$

and

$$\Lambda_{min}(M^{-1}A) \ge \min_{1 \le l \le L} 4^{L-l} \left[1 - \cos(2^{-L+l-1}\pi)\right] \ge 1,$$

Note that the last inequalities in above equations hold independent of L, or equivalently, the grid size h. Thus, the condition number $\kappa(M^{-1}A)$ is bounded by a constant

$$\kappa(M^{-1}A) < \frac{\pi^2}{2} \approx 4.93.$$

Figure 4.1 also shows the scaling factors c_l's for each band and Figure 4.2 shows the spectrum of the preconditioned system for $n = 256$. We can observe that the condition number has dropped from ~ 27000 to about 5.

4.1.4 Construction of Ideal Bandpass Filters P_l's

Having shown the effectiveness of MF preconditioning (namely $\kappa(M^{-1}A) = O(1)$), the question now is how to construct the space domain to space domain bandpass filters. In practice, a bandpass filter is usually constructed by taking the difference of two lowpass filters. Using the mathematical framework developed in the previous sections, a lowpass filter can be represented by

$$Q_l = W I_{Q_l} W^T$$

where I_{Q_l} is a diagonal matrix whose (k,k)th element is

$$I_{Q_l}(k,k) = \begin{cases} 1 & k \in band_j, \forall j \le l \\ 0 & otherwise. \end{cases}$$

Then a bandpass filter P_l is represented by

$$P_l = Q_l - Q_{l-1}$$

where $Q_L = I$, the identity and $Q_0 = 0$.

4.1.5 Construction of Ideal Lowpass Filters Q_l's

In order to form the MF preconditioner, we need to generate all lowpass filters $Q_l, l = 1, \cdots, L-1$. However, as we go to lower and lower frequency filters (as l goes to 1), the bandwidths are also getting narrower. It is well known that when a signal has compact support in frequency domain, it will have large support in the space domain (uncertainty principle). What this means is that the low-frequency lowpass filters are relatively expensive to construct, since it has to average over a larger number of grid points. This motivates the use of the cascade of a sequence of *elementary filters* $H_L, H_{L-1}, \cdots, H_2$ for the design of lowpass filters, with H_i's being simple and cheap averaging operators over grid points separated by spacing proportional to the wavelength of the band. However each H_i also lets through higher frequency aliases and the cascades are used to filter these out in order to achieve the lowpass effect. In mathematical terms, we define

$$H_l = W I_{H_l} W^T$$

where I_{H_l} is a diagonal matrix with the (k,k)th element

$$I_{H_l}(k,k) = \begin{cases} 1 & k \in band_1 \cup \cdots \cup band_{l-1} \\ 0 & k \in band_l \ . \end{cases}$$

The filter Q_l's are related to the filter H_l's via

$$Q_L = I,$$

$$Q_l = \prod_{p=l+1}^{L} H_p, 1 \leq l \leq L-1,$$

It is easy to verify that Q_l's satisfy the desired bandpass characteristics by pre- and post-multiplying the above equations with W and W^T respectively. Note also that the values of $I_{H_l}(k,k)$ for $k \cup band_{l+1} \cup \cdots \cup band_L$ do not influence the bandpass feature of Q_l's. This observation simplifies the design of H_l's.

4.1.6 Approximation 2 : Replace Bandpass by Lowpass Filters

Using the formulation from the previous sections, we can form the MF preconditioner mathematically in terms of Q_l's as

$$\begin{aligned}
M^{-1}v &= \sum_{l=1}^{L} \frac{P_l v}{c_l} \\
&= \sum_{l=1}^{L} \frac{(Q_l - Q_{l-1})v}{c_l} \\
&= \sum_{l=1}^{L} \frac{Q_l v}{c_l} - \sum_{l=1}^{L} \frac{Q_{l-1} v}{c_l} \\
&= \sum_{l=1}^{L} \frac{Q_l v}{c_l} - \sum_{l=1}^{L-1} \frac{Q_l v}{c_{l+1}} \\
&= \frac{Q_L v}{c_L} + \sum_{l=1}^{L-1} (\frac{1}{c_l} - \frac{1}{c_{l+1}}) Q_l v \\
&= \sum_{l=1}^{L} \frac{Q_l v}{d_l}
\end{aligned}$$

where $d_L = c_L$ and $\frac{1}{d_l} = \frac{1}{c_l} - \frac{1}{c_{l+1}}, l = L-1, \cdots, 1$. Note that the bandpass filters P_l's in the preconditioner M have been replaced by the more efficient lowpass filters Q_l's. We observed from numerical experiments that the following modified preconditioner

$$\hat{M}^{-1}v = \sum_{l=1}^{L} \frac{Q_l v}{c_l}$$

gives about the same convergence rate as M. This can also be proved by the following theorem.

Theorem 4.1 *If* $\exists \sigma > 0$ *such that* $c_{l+1} \geq (1+\sigma)c_l, \forall k = 1, 2, \ldots, L$, *then*

$$v^T \hat{M} v \leq v^T M v \leq (1+\sigma) v^T \hat{M} v.$$

Proof :

$$\begin{aligned} v^T M v &= \sum_{l=1}^{L} d_l v^T Q_l v \\ &= c_L v^T Q_L v + \sum_{l=1}^{L} \left[(c_l + \frac{c_l^2}{c_{l+1} - c_l}) v^T Q_l v \right] \\ &\leq \sum_{l=1}^{L} c_l (1 + \sigma^{-1}) v^T Q_l v - c_L \sigma v^T Q_l v \\ &\leq \sum_{l=1}^{L} c_l (1 + \sigma^{-1}) v^T Q_l v \\ &= (1 + \sigma^{-1}) v^T \hat{M} v. \end{aligned}$$

Also,

$$\begin{aligned} v^T \hat{M} v &= \sum_{l=1}^{L} c_l v^T Q_l v \\ &\leq \sum_{l=1}^{L} c_l v^T Q_l v + \sigma^{-1} \sum_{l=1}^{L-1} c_l v^T Q_l v \\ &= \sum_{l=1}^{L} c_l (1 + \sigma^{-1}) v^T Q_l v - c_L \sigma v^T Q_l v \\ &= v^T M v. \end{aligned}$$

This completes the proof.

4.1.7 Approximation 3 : The Use of Nonideal Elementary Filters

Consider the design of the filter H_L appearing at the first stage. The H_L have the following ideal lowpass characteristic,

$$I_{H_l}(k,k) = \begin{cases} 1 & 0 \leq k < 2^{L-1} \\ 0 & 2^{L-1} \leq k \leq 2^L. \end{cases}$$

We find that H_L is an $n \times n$ full matrix. Thus, the operation $H_L v$ for an arbitrary vector v has a complexity proportional to $O(n^2)$. This is too expensive to perform. Therefore, we seek the approximation of the ideal elementary filter H_L with a nonideal filter $H_{L,J}$ which is a symmetric band matrix of bandwidth $O(J)$ with the spectral property $I_{H_{L,J}}(k,k) \approx I_{H_L}(k,k)$ for $1 \leq k \leq n$. Consequently, the operation $H_{L,J} v$ only has a complexity proportional to $O(Jn)$ and the corresponding MF preconditioner has a complexity proportional to $O(LJn)$.

Let us write the nonideal elementary filter of the form

$$H_{L,J} = a_0 + \sum_{j=1}^{J} a_j (E^j + E^{-j}).$$

where the coefficients a_0 and a_j's are to be determined and the E^j and E^{-j} are shift operators of distance of 2^j in the x and -x directions respectively. In order to define the operation

$$H_{L,J} v_n = a_0 + \sum_{j=1}^{J} a_j (v_{n+j} + v_{n-j})$$

for any vector v_n appropriately, the odd-periodic extension of v_n is assumed,

$$v_{-m} = -v_m \text{ and } v_{m+2pn} = -v_m, \text{ for integer } p.$$

This implies that $H_{L,J}$ corresponds to a circulant matrix. The above odd-periodic assumption is only used for analyzing and designing $H_{l,J}$'s in this section. The actual implementation of the MF preconditioner with a multigrid discretization described in later section does not rely on this assumption.

There are numerous ways to determine the coefficients a_0 and a_j's depending what approximation criteria to be used. The operator $H_{L,J}$ has the eigenfunction $\sin(k\pi nh)$ with the eigenvalue

$$I_{H_{L,J}}(k,k) = a_0 + 2 \sum_{j=1}^{J} a_j \cos(k\pi jh).$$

Here we consider a class of filters based on the following two criteria:

1. $I_{H_{l,J}}(\frac{n}{2},\frac{n}{2}) = \frac{1}{2}$ and $I_{H_{L,J}}(k,k) - \frac{1}{2} = -\left[I_{H_{l,J}}(n-k,n-k) - \frac{1}{2}\right]$,

2. $I_{H_{l,J}}(k,k) = 1$ when $k \to 0$ and the first jth derivatives $(1 \leq j \leq J)$ of $I_{H_{l,J}}(k,k)$ are all zero as $k \to 0$.

The first criterion implies that the function $I_{H_{l,J}}(k,k) - \frac{1}{2}$ is odd symmetric with respect to $k = \frac{n}{2}$. A direct consequence of this criterion is that

$$a_0 = \frac{1}{2} \text{ and } a_j = 0, j \text{ positive even.}$$

The second criterion, called the *maximally flat* criterion [59], requires the approximation at the origin to be as accurate as possible. It is used to determine a_j with odd j. In Table 4.1, we list the coefficients a_j for $J = 1, 3, 5$ obtained according to criteria (1) and (2). The larger J becomes, the better the approximation is.

Table 4.1: Coefficients of a class of nonideal lowpass filters

J	a_0	a_1	a_3	a_5
1	$\frac{1}{2}$	$\frac{1}{4}$	0	0
3	$\frac{1}{2}$	$\frac{9}{32}$	$\frac{-1}{32}$	0
5	$\frac{1}{2}$	$\frac{150}{512}$	$\frac{-25}{512}$	$\frac{3}{512}$

The filter $H_{L-1,J}$ can be constructed with the same set of coefficients used by $H_{L,J}$, i.e.

$$H_{l-1,J} = a_0 + \sum_{j=1}^{J} a_j(E^{2j} + E^{-2j}).$$

The only difference between $H_{L,J}$ and $H_{L-1,J}$ is the position of grid points used for averaging. For the first-stage filter $H_{L,J}$, local averaging is used. For the second-stage filter $H_{L-1,J}$, we consider averaging between points separated by $2h$. This

design is due to the following reason. From the above equation, we see that the filter $H_{L-1,J}$ has the spectrum

$$I_{H_{L-1,J}}(k,k) = a_0 + 2\sum_{j=1}^{J} a_j \cos(k\pi j 2h),$$

and that $I_{H_{l-1,J}}(k,k)$ is related to $I_{H_{l,J}}(k,k)$ via

$$I_{H_{l-1,J}}(k,k) = I_{H_{l,J}}(2k,2k).$$

Consequently, for functions consisting only of components in low wavenumber region $1 \leq k < 2^{l-1}$, $H_{L-1,J}$ behaves like a lowpass filter which preserves components in the region $1 \leq k < 2^{L-2}$ and filters out components in the region $2^{L-2} \leq k < 2^{L-1}$. However, note that $H_{l,J}$, $l < L$ is not a lowpass filter with respect to the entire wavenumber band.

By applying the same procedure recursively, we can approximate the general elementary filter H_l on a uniform infinite grid as

$$H_{l,J} = a_0 + \sum_{j=1}^{J} a_j (E^{2^{l-1}j} + E^{-2^{l-1}j}), 2 \leq l \leq L,$$

where the coefficients a_j's are listed in Table 4.1. The spectrum of $H_{l,J}$ is

$$I_{H_{l,J}}(k,k) = a_0 + 2\sum_{j=1}^{J} a_j \cos(k\pi j 2^{(l-1)h}), 2 \leq l \leq L.$$

We can construct nonideal lowpass filters $Q_{l,J}$ with nonideal elementary filters $H_{l,J}$,

$$Q_{L,J} = I,$$

$$Q_{l,J} = \prod_{p=l+1}^{L} H_{p,J}, 1 \leq l \leq L-1.$$

The elementary filter $H_{l,J}$ is symmetric and so is the bandpass filter $Q_{l,J}$. Finally, we obtain the nonideal MF-preconditioner

$$\hat{M}_J^{-1} r = \sum_{l=1}^{L} \frac{Q_{l,J}}{c_l} r.$$

which approximates the ideal MF-preconditioner M.

It is worthwhile to summarize the similarities and differences between the fast Poisson solver and the MF preconditioning. They are both based on spectral decomposition. The fast Poisson solver decomposes a function into its Fourier components through the FFT, whereas the MF preconditioner approximately decomposes it into a certain number of bands through filtering. The filtering operations, which correspond to local averaging processes, can be easily adapted to irregular grids and domains and variable coefficients as will be shown later. In contrast, the FFT is primarily applicable to constant coefficient problems with regular grids and domains. Besides, for the fast Poisson solver we usually require detailed knowledge of the spectrum. But for the MF preconditioner we only have to estimate how the spectrum varies from one band to another.

4.1.8 Fourier analysis and higher dimensional cases

Since the MF preconditioner $\hat{M}_J$ and the Laplacian A share the same eigenvectors, i.e. Fourier sine functions, the spectrum and condition number of the MF-preconditioned Laplacian can be analyzed conveniently by Fourier analysis. From the last section, we have the following spectral relationship

$$I_{Q_{L,J}}(k,k) = 1,$$

$$I_{Q_{l,J}}(k,k) = \prod_{p=l+1}^{L} I_{H_{p,J}}(k,k), 1 \leq l \leq L-1.$$

We can then determine the eigenvalues of $\hat{M}_J^{-1}A$,

$$\lambda(\hat{M}_J^{-1}A) = \sum_{l=1}^{L} \frac{1}{c_l} I_{Q_{l,J}} I_{Q_{l,J}}^T \Lambda.$$

The eigenvalues of $\hat{M}_J^{-1}A$ are plotted as function of k with $J = 1, 3$ and $h^{-1} = 256$ in Figure 4.3 and 4.4. We should compare these spectra with that in Figure 4.2

based on the ideal filtering assumption. All of them have one common feature. That is, eigenvalues are redistributed in such a way that there exist many local maxima and minima. The condition numbers for $J = 1, 3$ are 8.67 3.29 respectively. Note that one of these numbers is in fact smaller than the condition number 4.58 obtained with ideal filtering. The precise reason for this phenomena is still not clear to us. It might be related to the smoothness of the eigenvalue distribution curves. The eigenvalue distribution for $\hat{M}_J^{-1}A$ in Figure 4.2 has many keen edges. However, these edges are smoothed by nonideal digital filters as shown in Figure 4.3 and 4.4.

The generalization of the MF preconditioner to two- or three-dimensional problems on square or cube domains can be done straightforwardly. For example, we may construct the two-dimensional elementary filter by the tensor product of one-dimensional elementary filters along the x- and y-directions,

$$H_{l,J} = \left[a_0 + \sum_{j=1}^{J} a_j (E_x^{2^{L-l}j} + E_x^{-2^{L-l}j}) \right] \times \left[a_0 + \sum_{j=1}^{J} a_j (E_y^{2^{L-l}j} + E_y^{-2^{L-l}j}) \right],$$

which can further be simplified by using operator algebra. For example, the coefficients for $H_{L,1}$ can be written in stencil form as

$$H_{L,1} : \frac{1}{16} \begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix}.$$

Similarly, the three-dimensional elementary filter can be obtained by the tensor product of three one-dimensional filters along the x-, y- and z-directions.

The condition numbers of one-, two- and three-dimensional MF-preconditioned Laplacians with two types of nonideal filters ($J = 1$ and $J = 3$) are computed and plotted as function of the grid size h in Figures 4.5 and 4.6. These figures

show that $\hat{M}_J^{-1}A$ for $J = 1$ behaves like $O(\log h^{-1})$ while that for $J = 3$ behaves asymptotically like $O(1)$ for two- and three-dimensional cases.

Earlier we discussed the odd-periodic property of the sequence v_n. However, this may not be easily implementable for general multidimensional problems with nonrectangular domains. The difficulty arises when the size of $H_{l,J}$ is so large that it operates on points outside the domain. There are two possible solutions. It may be preferable to construct filters of larger size by the repeated application of filters of smaller size. For example, we can apply the filter $H_{L,J}$ with $J = 1$ twice. This is equivalent to a filter of size 5,

$$H_{L,1}^2 = (\frac{1}{4}E^{-1} + \frac{1}{2} + \frac{1}{4}E)^2 = \frac{1}{16}E^{-2} + \frac{1}{4}E^{-1} + \frac{3}{8} + \frac{1}{4}E + \frac{1}{16}E^2.$$

Another possibility is to apply smaller filters at points close to boundaries and larger filters at points far away from boundaries. Note also that, for fixed J, the size of the elementary filter $H_{l,J}$ increases as l decreases. However, this problem can be resolved by incorporating the multigrid discretization structure into the above multilevel filtering framework as described in the next subsection.

4.1.9 Multigrid multilevel filtering (MGMF) preconditioners

In previous sections, we discussed the construction of the MF preconditioner for the model Poisson problem based on a single discretization grid (let's call it single grid multilevel filtering SGMF preconditioner). This section will discuss the generalization of this preconditioning technique so that it can be implemented more efficiently and applied to more general self-adjoint elliptic PDE problems.

The filtering operation described above is performed at every grid point at all levels $2 \leq l \leq L$. Since there are $O(\log n)$ levels and $O(Jn)$ operations per level, where n and J denote the order of unknowns and the filter size respectively, the

total number of operations required is proportional to $O(Jn \log n)$. However, since waveforms consisting only of low wavenumber components can be well represented on coarser grids, we can use the multigrid philosophy [24, 55] and incorporate the multigrid discretization structure into the filtering framework described in the previous sections. That is, we construct a sequence of grids Ω_l of sizes $h_l = O(2^{-l})$, $1 \leq l \leq L$, to represent the decomposed components. Then, the total number of unknowns is $O(n)$ and consequently the total number of operations per MF preconditioning step is $O(Jn)$. Note that J is a constant independent of h.

The multigrid multilevel filtering (MGMF) preconditioner is obtained by inserting down-sampling (I_l^{l-1}) and up-sampling (I_{l-1}^l) operators into the SGMF preconditioner. With the notation commonly used in the multigrid literatures, the down-sampling and up-sampling operators for grids Ω_l ($h_l = 2^{L-l}h$) and Ω_{l-1} ($h_{l-1} = 2^{L-l+1}h$) can be defined as

$$I_l^{l-1} : \left| \begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix} \right|_l^{l-1}, \quad I_{l-1}^l : \left| \begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix} \right|_{l-1}^{l}.$$

It is easy to verify that a lowpass filter followed by a down-sampling operator is the same as the restriction operator in MG methods while an upsampling operator followed by a lowpass filter is equivalent to the interpolation operator.

Given a sequence of grids Ω_l, $1 \leq l \leq L$, down-sampling (I_{l+1}^l) and up-sampling (I_l^{l+1}) operators between grids Ω_l and Ω_{l+1}, and appropriate elementary filters H_l defined on Ω_l, the two-dimension MGMF algorithm can be summarized as follows,

Algorithm MGMF2D ($z = M^{-1}r$)

$w = r$

for $l = L, 2, -1$

$v = filter(w, l)$

$u_l = 4^{l-1} \times (w - v)$

$w = I_l^{l-1} v$;

$z = 4^{L-1} \times w$

for $l = 2, L$

$z = I_{l-1}^{l} z + u_l$;

end MGMF2D

This is the MGMF algorithm implemented in the numerical experiments.

The preconditioning $\tilde{M}_J^{-1} r$ can be viewed as a degenerate multigrid method, for which we have a sequence of restriction and interpolation operations but where the error smoothing at each grid level is replaced by an appropriate scaling. This observation leads us to generalize the MF preconditioner to the case of nonuniform grids commonly obtained from the finite-element discretization. That is, one can view projection as decomposition and interpolation as synthesis and any multigrid method can be used as an MGMF preconditioner if we replace the potentially more expensive error smoothing by a simple scaling. Since the eigenvalues λ_k in band $band_l$ behaves like $O(h_l^{-2})$, where h_l describes approximately the grid spacing for level l [22], therefore, a general rule for selecting the scaling constant c_l at grid level l is

$$c_l = O(h_l^{-2})$$

for second-order problems. In general, to select the scaling constant c_l for an β-order problem, we use

$$c_l = O(h_l^{-\beta}).$$

This generalized version is closely related to the preconditioner by Bramble,

Pasciak and Xu [22]. They derived their preconditioner in the finite-element context discretized with the nested triangular elements. From our filtering framework, the corresponding elementary filters $H_{l,J}$ takes the form

$$H_{l,BPX} : \frac{1}{8} \begin{vmatrix} 0 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{vmatrix}.$$

which is different from $H_{l,1}$ given earlier. We can derive other filters by applying it more than once. For example, by applying it twice, we get

$$H_{l,BPX2} : \frac{1}{64} \begin{vmatrix} 0 & 0 & 1 & 2 & 1 \\ 0 & 2 & 6 & 6 & 2 \\ 1 & 6 & 10 & 6 & 1 \\ 2 & 6 & 6 & 2 & 0 \\ 1 & 2 & 1 & 0 & 0 \end{vmatrix}.$$

In order to eliminate the directional preference, we can apply BPX in alternating directions which gives a symmetric filter:

$$H_{l,BPX3} : \frac{1}{64} \begin{vmatrix} 0 & 1 & 2 & 1 & 0 \\ 1 & 4 & 6 & 4 & 1 \\ 2 & 6 & 8 & 6 & 2 \\ 1 & 4 & 6 & 4 & 1 \\ 0 & 1 & 2 & 1 & 0 \end{vmatrix}.$$

The MF preconditioner is designed to capture the spectral property (or h-dependency) of a discretized elliptic operator but not the variation of its coefficients. This is also true for the hierarchical basis and BPX preconditioners. In order to take badly scaled variable coefficients into account, we use the MF preconditioner in association with diagonal scaling in our experiments [51]. The diagonal

scaling is often used for cases where the diagonal elements of the coefficient matrix A vary for a wide range. Suppose that the coefficient matrix can be written as

$$A = D^{\frac{1}{2}} \tilde{A} D^{\frac{1}{2}},$$

where we choose D to be a diagonal matrix with positive elements in such a way that the diagonal elements of $\tilde{A}$ are of the same order, say, $O(1)$. Then, in order to solve $Au = f$, we can solve an equivalent problem $\tilde{A}\tilde{u} = \tilde{f}$, where $\tilde{u} = D^{\frac{1}{2}} u$ and $\tilde{f} = D^{-\frac{1}{2}} f$, with the MF preconditioner. There exist other ways to incorporate the coefficient information into preconditioners of the multilevel type, say, to use the Gauss-Seidel smoothing suggested by Bank et al. [15].

4.2 Numerical Results

In this section, we present numerical results for two- and three-dimensional Poisson, variable coefficient and discontinuous coefficient problems to demonstrate the convergence behavior when MGMF preconditioning is used. Three variations of the MGMF preconditioning are implemented :

MGMF1 the MGMF preconditioner with 9-point (27-point) filter for 2D (3D) problems. (i.e. $J = 1$ filter)

MGMF2 a modified version of MGMF in which the 9-point (27-point) filter is applied twice. (i.e. $J = 1$ filter twice)

MGMF3 another modified version of MGMF in which the 9-point (27-point) filter is applied once at the finest grid level (to give smaller amount of work compared to (MGMF2) and twice at other grid levels (to achieve a convergence rate between MGMF1 and MGMF2 but close to MGMF2).

The preconditioning operation counts for 2D (3D) problems are $9N$, $27N$ and $15N$ ($9N$, $32N$ and $12N$) respectively for MGMF1, MGMF2 and MGMF3. These operation counts include also the diagonal scaling.

For all test problems, we use the standard 5- (or 7-) point stencil on a square (or cubic) uniform mesh with $h = \frac{1}{n+1}$ and $N = n^2$ (or $N = n^3$), zero boundary conditions and zero initial guesses. Experimental results are given for different values of h and the stopping criterion is $\| r^k \| / \| r^0 \| \leq 10^{-5}$. The six test problems are:

1. the 2D model problem with solution $u = x(x-1)y(y-1)e^{xy}$,

$$-\triangle u = f, \Omega = [0,1]^2,$$

2. a 2D variable coefficient problem with solution $u = xe^{xy}\sin\pi x \sin\pi y$,

$$\frac{\partial}{\partial x}\left(e^{-xy}\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(e^{xy}\frac{\partial u}{\partial y}\right) = f, \Omega = [0,1]^2,$$

3. a 2D discontinuous coefficient problem with $f = 2x(1-x) + 2y(1-y)$,

$$\frac{\partial}{\partial x}\left(\rho(x,y)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\rho(x,y)\frac{\partial u}{\partial y}\right) = f, \Omega = [0,1]^2,$$

where

$$\rho(x,y) = \begin{cases} 10^4 & x > 0.5, y \leq 0.5 \\ 10^{-4} & x \leq 0.5, y > 0.5 \\ 1 & otherwise \end{cases}$$

4. the 3D model problem with solution $u = x(x-1)y(y-1)z(z-1)e^{xyz}$,

$$-\triangle u = f, \Omega = [0,1]^3,$$

5. a 3D variable coefficient problem with solution $u = e^{xyz}\sin\pi x \sin\pi y \sin\pi z$,

$$\frac{\partial}{\partial x}\left(e^{-xyz}\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(e^{xyz}\frac{\partial u}{\partial y}\right) + \frac{\partial}{\partial z}\left(e^{-xyz}\frac{\partial u}{\partial z}\right) = f, \Omega = [0,1]^3,$$

6. a 3D discontinuous coefficient problem with $f = 2x(1-x) + 2y(1-y) + 2z(1-z)$,

$$\frac{\partial}{\partial x}\left(\rho(x,y,z)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\rho(x,y,z)\frac{\partial u}{\partial y}\right) + \frac{\partial}{\partial z}\left(\rho(x,y,z)\frac{\partial u}{\partial z}\right) = f, \Omega = [0,1]^3,$$

where

$$\rho(x,y,z) = \begin{cases} 10^{-4} & x > 0.5 \text{ with } y \leq 0.5, z \leq 0.5 \text{ or } y > 0.5, z > 0.5 \\ 10^{4} & x \leq 0.5 \text{ with } y > 0.5, z \leq 0.5 \text{ or } y \leq 0.5, z > 0.5 \\ 1 & \textit{elsewhere} \end{cases}$$

For comparison purposes, the hierarchical basis (HB) [15] [111] and multigrid with modified Jacobi as smoother (MG(k), where k is the number of pre- and post-smoothings used) preconditioners were also implemented. The operation counts per iteration for the HB and the MG(k) preconditioners are $7N$ and $26 + 32 \times k$ ($8N$ and $26 + 36 \times k$ for 3D) respectively. The number of iterations are shown in the following Tables 4.2 to 4.7 ('-' in Tables 4.5 and 4.6 means 'data not available'). For different test problems the k in MG(k) that gives the best overall operation count is shown.

The iteration counts shown in the tables do not reflect the overall operation counts for the preconditioners. In Tables 4.8 and 4.9 we also show the total operation count required per grid point for each preconditioner. (We show only the data for $n = 256$ and $n = 32$ for the 2D and 3D problems respectively).

We can observe from the tables that filtering twice (MGMF2) always improves the convergence rates over MGMF1 but not the overall operation count. This observation was the main driving force for the design of MGMF3 and we see that MGMF3 requires less work per grid point than MGMF1 and MGMF2. Also, from the tables it appears that both MGMF2 and MGMF3 give condition number of

Table 4.2: Iteration counts for Test Problem 1

n	*MGMF1*	*MGMF2*	*MGMF3*	*HB*	*MG(2)*
8	10	9	10	16	4
16	11	9	10	24	4
32	12	8	10	34	5
64	13	8	10	44	5
128	15	8	10	54	5
256	16	7	10	64	5

Table 4.3: Iteration counts for Test Problem 2

n	*MGMF1*	*MGMF2*	*MGMF3*	*HB*	*MG(1)*
8	13	12	13	18	7
16	17	14	16	27	8
32	22	17	19	36	10
64	26	18	22	46	12
128	30	20	24	56	13
256	33	21	26	67	15

Table 4.4: Iteration counts for Test Problem 3

n	*MGMF1*	*MGMF2*	*MGMF3*	*HB*	*MG(10)*
8	21	19	20	28	6
16	35	30	33	49	10
32	59	49	51	79	15
64	101	82	86	132	17
128	200	140	143	223	20
256	367	254	269	393	24

Table 4.5: Iteration counts for Test Problem 4

n	*MGMF1*	*MGMF2*	*MGMF3*	*HB*	*MG(2)*
8	11	8	11	20	5
16	13	8	10	30	5
32	13	8	10	45	6
64	14	7	10	70	-

Table 4.6: Iteration counts for Test Problem 5

n	*MGMF1*	*MGMF2*	*MGMF3*	*HB*	*MG(2)*
8	13	11	13	20	5
16	16	12	14	33	6
32	18	13	16	53	7
64	21	14	18	82	-

Table 4.7: Iteration counts for Test Problem 6

n	*MGMF1*	*MGMF2*	*MGMF3*	*HB*	*MG(10)*
8	24	21	24	43	8
16	46	38	41	96	15
32	95	71	74	229	20

Table 4.8: Operation counts per grid points for 2D problems (n=256)

test problem	*MGMF1*	*MGMF2*	*MGMF3*	*HB*	*MG*
1	448	328	340	1664	555
2	990	1008	936	1876	1185
3	11010	12192	9684	11604	8808

Table 4.9: Operation counts per grid points for 3D problems (n=32)

test problem	*MGMF1*	*MGMF2*	*MGMF3*	*HB*	*MG*
4	416	440	350	395	615
5	612	741	592	1749	861
6	3230	4047	2738	7557	8220

$O(1)$ for Poisson problems in 2D and 3D, of $O(\log n)$ for variable coefficient problems and of $O(n)$ for discontinuous coefficient problems. The HB preconditioner does not exhibit competitive performance both in terms of iteration count and operation count, especially for 3D problems, since the condition number behaves like $O(h^{-1})$ instead of $O(\log h^{-1})$ for 2D problems. The HB preconditioner should give better performance for problems with nonuniformly refined grids. The MG preconditioner gives the best convergence rates for all the test problems attempted. However, for smooth problems it performs worse than the MGMF preconditioners in operation counts mainly because of the expensive work spent in the relaxation steps. For discontinuous coefficient problems (e.g. test problems 3), the MG preconditioner sometimes gives better operation counts than the others when the number of relaxation steps is large enough (10 in our experiments). In our previous paper when we used 3 relaxation steps the operation count for problem 3 was found to be the worst of all.

Figure 4.1: Spectrum of Laplacian divided into Bands and Scalings(n=256)

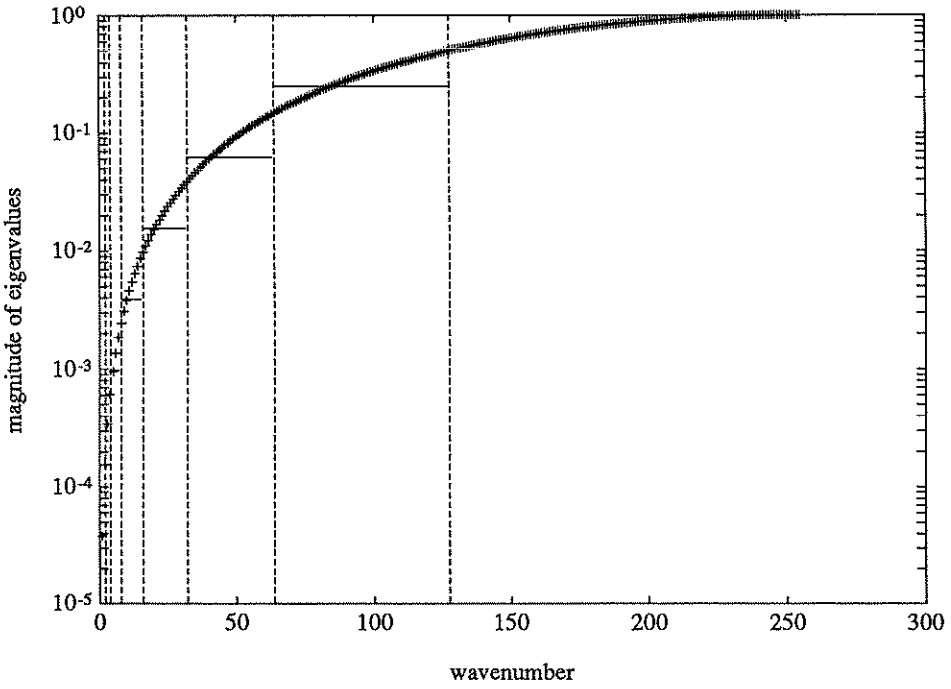


Figure 4.2: Preconditioned Spectrum for Laplacian Using Ideal Filters (n=256)

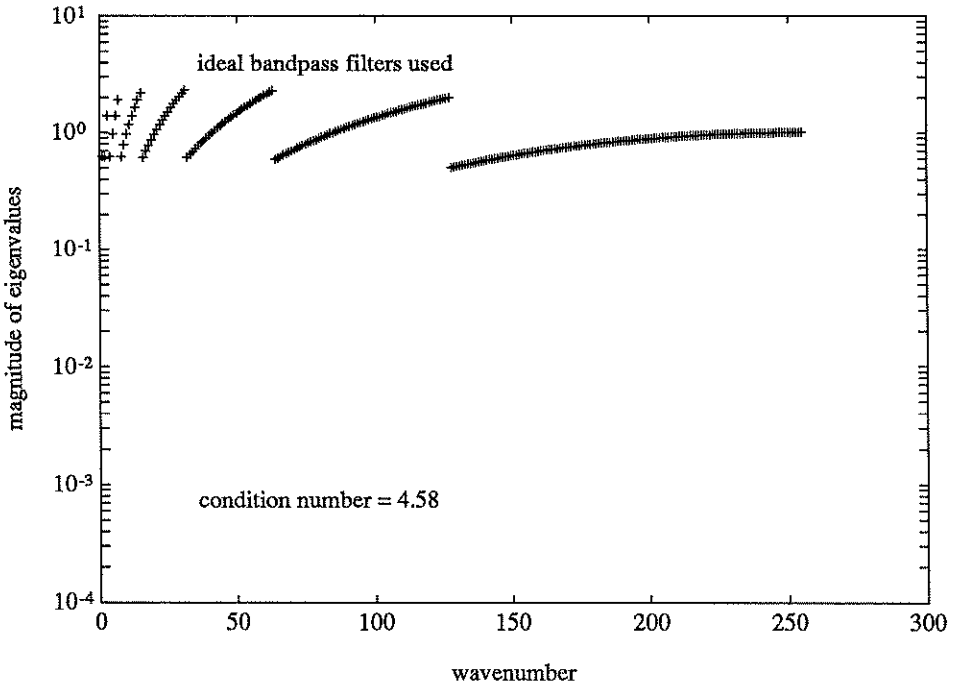


Figure 4.3: Preconditioned Spectrum for Laplacian with J=1 Filter (n=256)

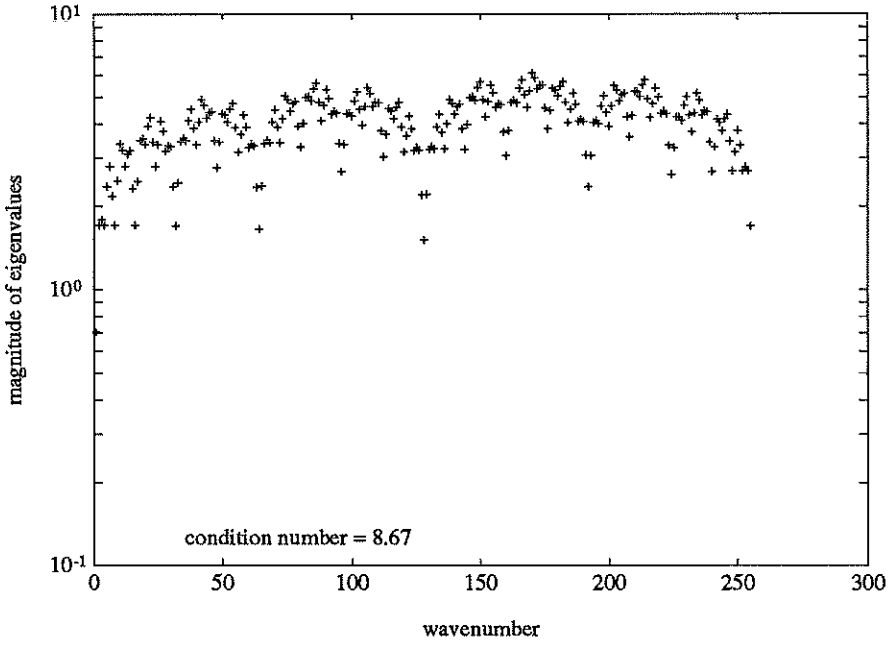


Figure 4.4: Preconditioned Spectrum for Laplacian with J=3 Filter (n=256)

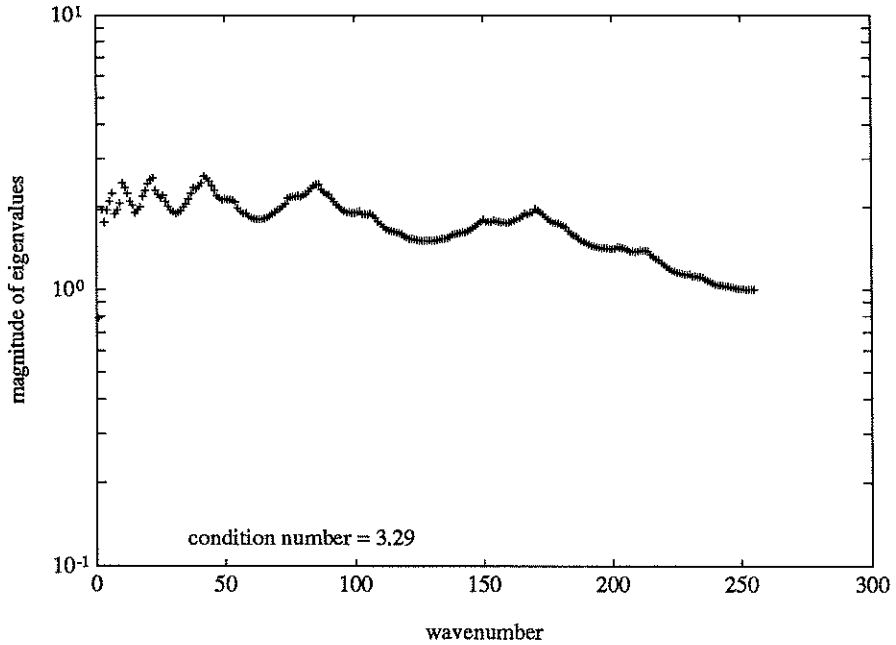


Figure 4.5: Condition Numbers for $\hat{M}_J^{-1}A$ with J=1 Filters

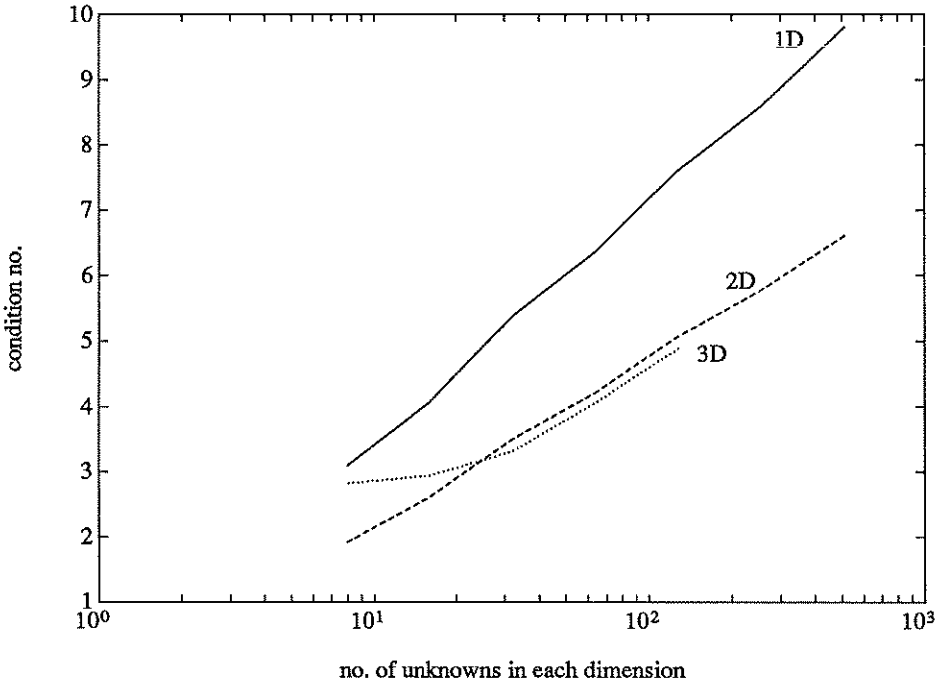
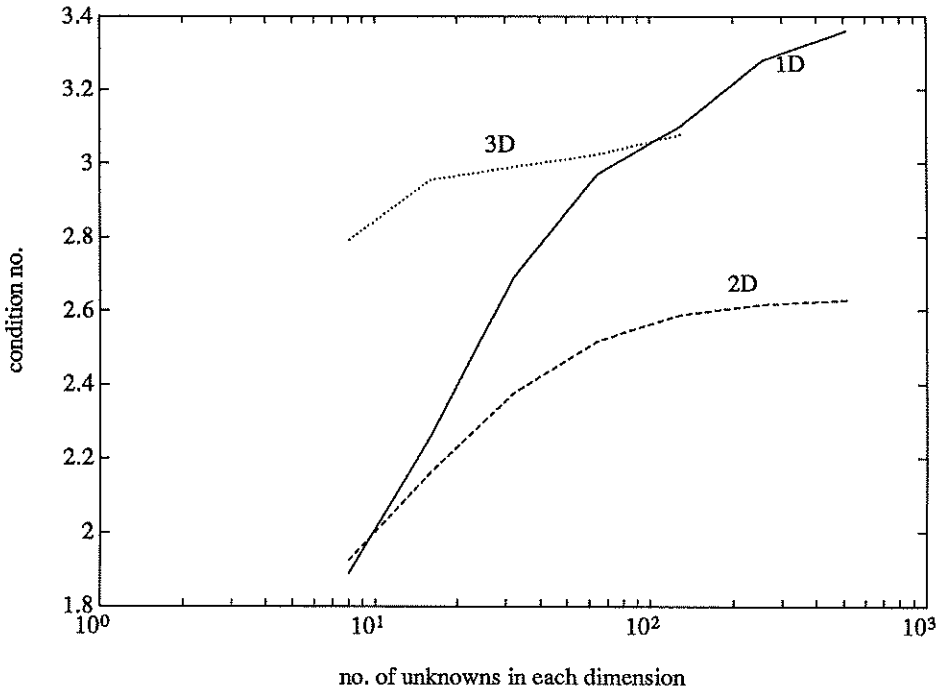


Figure 4.6: Condition Numbers for $\hat{M}_J^{-1}A$ with J=3 Filters



CHAPTER 5

Extension of MF to More General Elliptic Problems

In the last chapter we have presented the idea of multilevel filtering (MF) preconditioning applied to second-order self-adjoint elliptic problems. In this chapter we will show how to effectively apply the same idea to other elliptic problems such as the second-order anisotropic problem, Helmholtz equation, convection-diffusion equation, biharmonic equation, and problems on locally refined grids. Fourier analysis is used as a tool to understand the effect of the SGMF preconditioners on model problems on uniform grid with suitable boundary conditions. However, it should be emphasized that the MF preconditioners can be applied to problems with variable or discontinuous coefficients and/or irregular grids. Numerical results are given to show the effectiveness of the MF preconditioners on some model problems.

5.1 MF Preconditioners for Anisotropic Problems

In this section, we extend the concept of multilevel filtering to the second-order anisotropic problems. To achieve high degree of efficiency, the preconditioning step requires some modifications in the design of filters. We first provide justification for such modifications and then we will show the condition number computed by Fourier analysis. Numerical experiments are also included.

Consider the following 2D second-order anisotropic problem :

$$-\alpha u_{xx} - u_{yy} = f(x,y) \text{ in } \Omega = [0,1]^2$$

where $\alpha > 1$ and with zero Dirichlet boundary conditions. The discretization of the equation using uniform square mesh with $h = \frac{1}{n+1}$ gives a block-tridiagonal matrix A such that $Au = f$ where u is the solution. In the Fourier domain we can express this as :

$$\hat{A}(j,k)\hat{u}_{j,k} = \hat{f}_{j,k},\ j,k = 1,2,\cdots,n-1$$

where

$$\hat{u}_{j,k} = \frac{\sqrt{2}}{n}\sum_{l=1}^{n}\sum_{m=1}^{n} u_{l,m}\sin(j\pi lh)\sin(k\pi mh)$$

and

$$\hat{f}_{j,k} = \frac{\sqrt{2}}{n}\sum_{l=1}^{n}\sum_{m=1}^{n} f_{l,m}\sin(j\pi lh)\sin(k\pi mh)$$

such that

$$\hat{A}(j,k) = (2+2\alpha) - 2(\alpha\cos j\pi h + \cos k\pi h)$$

We can observe from the eigenvalue spectrum of $\hat{A}$ that for $\alpha \gg 1$ the variation in magnitudes of the eigenvalues in the k-direction is relatively small compared to that in the j-direction. To maintain uniform variation of eigenvalues within each band, we divide more wavenumber bands in the j-direction than in the k-direction. We call this technique *directionally adaptive filtering.* This can be done in practice by first performing 1D filtering in the j-direction for a number of levels (say number of levels $= \gamma$) and after that resuming 2D filtering. This is in contrast to performing 2D filtering for all the levels for the nearly isotropic problems described in the last section. Here γ depends on α as well as the problem to be solved. For second order elliptic problems with quasiuniform grid and $h_l \approx 2h_{l+1}$ it is sufficient to use $\gamma = round(\log_4 \alpha)$. Suppose $\alpha = 4$, then $\gamma = 1$ and the modified $H_{l,1}$ for the finest grid level takes the following stencil form :

$$\frac{1}{4}\left|\begin{array}{ccc} 1 & 2 & 1 \end{array}\right|$$

while the filter for the other coarse grid levels have a 2D stencil (tensor product of 1D filter, i.e. $H_{l,1} \times H_{l,1}$).

Note that if the finest level is defined on a $(n+2)\times(n+2)$ grid, then for $\gamma \geq 1$ the next coarse level is defined on $(\frac{n+1}{2}+1)\times(n+2)$ grid instead of $(\frac{n+1}{2}+1)\times(\frac{n+1}{2}+1)$ grid for $\gamma = 0$. It should also be noted that this modified filtering scheme is analogous to the idea of semi-coarsening in the multigrid literatures.

We performed Fourier analysis of the single grid version of this scheme (called SGMF1a) on the 2D anisotropic problem with different α and h. The condition numbers of the preconditioned system are given in Table 5.1. For comparison purpose, the condition numbers of the preconditioned system using the unmodified SGMF1 preconditioner are also included. Table 5.1 shows that this modified scheme is quite effective. For example, for $\alpha = 1000$ the condition number grows slowly with n while this is not true for the unmodified SGMF1 preconditioner.

Table 5.1: condition number for different α and n

		$\alpha = 10$		$\alpha = 100$		$\alpha = 1000$	
n	A	SGMF1a	SGMF1	SGMF1a	SGMF1	SGMF1a	SGMF1
7	25	3.8	13	3.8	38	3.8	47
15	103	4.3	21	4.7	117	4.7	216
31	414	5.4	28	5.8	233	5.9	849
63	1659	6.6	34	6.8	328	6.9	2142
127	6639	8.2	40	7.9	395	8.0	3480
255	26560	9.7	46	9.0	454	9.0	4396

The MGMF1 preconditioning algorithm for the above anisotropic problems can

be summarized as follows :

Algorithm MGMF1a : input = r, output = $z = M^{-1}r$

$v_L := r$

Decomposition :

$count = \gamma$

for $l = L-1, \cdots, 1$

if $(count = 0)$ then

$t_l :=$ x-filter1(v_{l+1})

$v_l :=$ y-filter1(t_l)

else

$count = count - 1$

$v_l :=$ x-filter1(v_{l+1})

end if

end for

Scaling :

for $l = 1, \cdots, L$

$v_l := v_l \div c_l$

end for

Synthesis :

$t_1 := v_1$

for $l = 2, \cdots, L$

$t_l := v_l + H_{l,1} I_{l-1}^{l} t_{l-1}$

end for

$z := t_L$

end MGMF1a

Next we show the numerical results using the multigrid MF (MGMF1a) preconditioner in conjunction with the conjugate gradient method. Again, we use the standard 5-point discretization on a uniform square mesh with $h = \frac{1}{n+1}$ and the forcing function $f(x,y)$ is such that the solution is $u = x(x-1)y(y-1)e^{xy}$. The stopping criterion used is $\| r^k \|_2 / \| r^0 \|_2 \leq 10^{-5}$ and the initial guess is 0. The iteration counts for different h and α are shown in Table 5.2.

Table 5.2: Iteration counts for different α and n

	$\alpha = 100$			$\alpha = 1000$		
n	A	MGMF1a	MGMF1	A	MGMF1a	MGMF1
7	17	7	19	13	6	20
15	41	10	41	27	9	44
31	90	12	64	63	12	84
63	187	13	83	126	13	140
127	388	15	95	258	15	193
255	812	17	106	608	17	224

The numerical results show that this scheme works very well for a wide range of α. It should be noted that a similar scheme can be applied to the case when $\alpha < 1$ and for the 3D anisotropic problems.

5.2 MF Preconditioners for Positive Definite Helmholtz Equation

Consider the following 2D Helmholtz equation :

$$-\triangle u + \beta u = f \text{ in } \Omega = [0,1]^2$$

with zero Dirichlet boundary condition and when β is a positive or a small negative constant so that the discretization matrix A is symmetric and positive definite. (Most Helmholtz problems with negative β, however, give rise to symmetric but indefinite stiffness matrices. We plan to pursue this type of problems in the future). An effective MF preconditioner for this equation requires modifications in the scaling constants c_l's, as explained below. Again we can express $Au = f$ in Fourier domain with $\hat{A}$ as :

$$\hat{A}(j,k) = 4\sin^2(i\pi h) + 4\sin^2(j\pi h) + \beta h^2.$$

The spectrum of $\hat{A}$ differs from that of the Poisson equation by βh^2 and we need to incorporate this offset in the scaling constants $c_l's$. Instead of using $c_{l+1} = 4c_l$ for Poisson equation with $h_l \approx 2h_{l+1}$, The recurrence relation is now given by $c_l = \frac{c_{l+1}+3kh^2}{4}$ with $c_L = 8 + kh^2$ (let us call this scheme SGMF1b). To find the range of β such that all the eigenvalues are real and positive, we can first observe from the equation above that this is indeed the case when $\beta > 0$. And if $\beta < 0$, it is straightforward to find the lower bound of β as

$$\beta > \frac{-8\sin^2(\pi h/2)}{h^2}.$$

For the 2D Helmholtz equation with $n = 256$, this lower bound is about -19.7. In Table 5.3, we show the condition numbers of the preconditioned systems (SGMF1b) computed by Fourier analysis for a range of β and compare them with those of the unpreconditioned system (A). We also include the condition numbers when the unmodified SGMF1 preconditioner is used. The Fourier results show that for large β, the modified preconditioner SGMF1b improves the condition number significantly over SGMF1.

Table 5.3: condition number for different n and β

	$\beta = 10$			$\beta = 1000$		
n	A	SGMF1b	SGMF1	A	SGMF1b	SGMF1
7	17	1.97	2.18	1.5	2.60	12.5
15	69	2.61	2.85	3	3.70	31.0
31	275	3.36	3.60	9	4.63	49.3
63	1102	4.09	4.33	33	5.04	57.9
127	4407	4.84	5.09	129	5.17	60.5
255	17629	5.59	9.94	515	5.20	61.2

We now show the numerical results using the multigrid formulation of the SGMF1b preconditioner (MGMF1b). Again, we use the standard 5-point discretization on a uniform square mesh with $h = \frac{1}{n+1}$ and the forcing function $f(x, y)$ is such that the solution is $u = x(x-1)y(y-1)e^{xy}$. The stopping criterion used is $\| r^k \|_2 / \| r^0 \|_2 \leq 10^{-5}$ and the initial guess is 0. The iteration counts for the unpreconditioned (A) and preconditioned CG methods (MGMF1b and MGMF1) with different n and β are shown in Table 5.4.

Again, we can see that the numerical results agree with the results from Fourier analysis. For small β, the MGMF1b and MGMF1 requires almost the same number of iterations to achieve convergence. However, for large β, the advantage of using the MGMF1b becomes obvious.

Table 5.4: Iteration counts for Helmholtz equation with different n and β

	$\beta = 10$			$\beta = 1000$		
n	A	MGMF1b	MGMF1	A	MGMF1b	MGMF1
7	15	10	10	4	6	8
15	32	11	11	6	8	11
31	66	12	12	10	10	16
63	133	13	13	22	11	18
127	273	15	15	48	12	19
255	555	16	16	101	13	20

5.3 MF Preconditioners for Convection-diffusion Equation

Consider the 2D convection-diffusion equation

$$k(x,y) \cdot \bigtriangledown u = \epsilon \bigtriangleup u + f(x,y) \text{ in } \Omega = [0,a] \times [0,b]$$

with Dirichlet boundary conditions on $\partial\Omega$. This equation, for example, describes the concentration of a chemical in a solution flowing with a time-independent velocity field $k(x,y)$. We examine the simple case when $k(x,y) = [k,0]^T$, $\epsilon = 1$ and $a = b = 1$ so that the equation becomes

$$-\bigtriangleup u + ku_x = f(x,y) \text{ in } \Omega = [0,1]^2.$$

The application of MF preconditioning to this convection-diffusion equation requires special handling because the discretization matrix A is nonsymmetric. In the following sub-section we consider the cases when $k \times h$ is small enough so that the discretized system is symmetrizable, i.e. there exists a diagonal matrix

D such that DAD^{-1} is symmetric, and positive definite). Then we consider MF-preconditioned conjugate gradient method applied to normal equation.

5.3.1 MF Preconditioners for Symmetrized Systems

In the following we describe a method to handle the problem when kh is relatively small. The method symmetrizes A before applying the PCG algorithm. The symmetrized system resembles the positive-definite Helmholtz equation and thus efficient MF preconditioners are known from the last section. We examine two discretization schemes for the method and show numerically that the MF preconditioner is effective for both schemes.

Scheme 1A Central Difference Scheme

This scheme on a square mesh of side h gives rise to :

$$-\triangle_h u_{i,j} + \frac{k}{2h}(u_{i+1,j} - u_{i-1,j}) = f_{i,j},$$

where $\triangle_h u_{i,j}$ is the standard 5-point finite difference discretization for the Poisson equation. This scheme has an accuracy of $O(h^2)$. However, to obtain stability, the mesh size h has to be $\leq \frac{2}{k}$ [18]. Therefore, if k is large (which is typical for many applications), very small mesh size has to be used in order to maintain stability. However, too small a mesh size also means unnecessarily high operation count to arrive at the solution.

Scheme 1B HODIE Scheme [79]

The HODIE scheme for the above convection-diffusion equation has the following difference formula :

$$\begin{aligned} & (2\tau+2)u_{i,j} - (\tau+\tfrac{kh}{2})u_{i-1,j} - (\tau-\tfrac{kh}{2})u_{i+1,j} - u_{i,j-1} - u_{i,j+1} \\ = \quad & kh^2 f(ih + \tfrac{(1-\tau)h}{k}, jh) \end{aligned}$$

where $\tau = \sqrt{1 + \frac{k^2h^2}{3}}$.

An advantage of this method is that it has an accuracy of $O(h^2)$ and is also unconditionally stable.

The discretization matrices from the schemes 1A and 1B above are nonsymmetric but symmetrizable when h is in the range of stability. We can symmetrize these matrices before applying preconditioned conjugate gradient methods. The symmetrized matrices in general are equivalent to the discretization of certain Helmholtz equations. Consequently, the MF-preconditioning techniques for the Helmholtz equation can be used here. Recall that for the Helmholtz equation, only the scaling constants need to be modified. Using the same technique, we can derive the scaling recurrence for these schemes as :

$$c_l = \frac{c_{l+1} + 3s}{4},$$

where s is different for different methods :

- Central difference :

$$s = \frac{2 - 2\sqrt{(1 + \frac{kh}{2})(1 - \frac{kh}{2})}}{4 + 4\sqrt{(1 + \frac{kh}{2})(1 - \frac{kh}{2})}}, c_L = 6 + 2\sqrt{(1 + \frac{kh}{2})(1 - \frac{kh}{2})}$$

- HODIE Method :

$$s = \frac{2\sqrt{1 + \frac{k^2h^2}{3}} - 2\sqrt{1 + \frac{k^2h^2}{12}}}{4 + 4\sqrt{1 + \frac{k^2h^2}{12}}}, c_L = 4 + 2\tau + 2\sqrt{(\tau + \frac{h^3}{2})(\tau - \frac{h^3}{2})}$$

We use the following test problem :

$$-\triangle u + ku_x = f(x, y) \text{ in } \Omega = [0, 1]^2$$

where $f(x,y) = -\pi_2(1 - \frac{e^{x/k}}{e^{1/k}})\sin \pi y$ so that the solution is given by $u(x,y) = (1 - \frac{e^{x/k}}{e^{1/k}})\sin \pi y$. The stopping criterion is $\| r^k \|_2 / \| r^0 \|_2 \leq 10^{-10}$ and zero initial guess is used. The iteration counts are given in Table 5.5 for $k = 30$ and different $h = \frac{1}{n+1}$.

Table 5.5: Iteration Counts for Convection Diffusion equation

n	$1A$	$1B$
15	26	21
31	27	25
63	27	26
127	27	27
255	27	27

We can observe from Table 5.5 that both schemes used here require about the same number of iterations and the convergence rates seem to depend only slightly on n. Both the central difference and HODIE schemes seem to give reasonable accuracy. The central difference scheme is both easy to use and reasonably accurate when the convective term is not too large. When the convective term is relatively large and high accuracy is needed, then the HODIE scheme may be promising.

5.3.2 MF Preconditioners for Normal Equation

For convection-diffusion equation with large kh term, we use MF preconditioner combined with conjugate gradient method applied to normal equation. By using the normal equation approach for second-order problems, the eigenvalues of the system are due to the contributions from both the fourth-order and second-order

terms. If the fourth-order term is dominant (i.e. kh is small), we can ignore the second-order term and treat the problem as a biharmonic equation for which we know about efficient MF preconditioner from previous sections. If the second-order is dominant (i.e. kh is large), we can ignore the fourth-order term and treat the problem as a second-order anisotropic problem for which again we know about efficient MF preconditioner. When kh is medium large, it is not clear what the best way is to apply MF preconditioning.

We again use the following test problem :

$$-\triangle u + ku_x = f(x,y) \text{ in } \Omega = [0,1]^2$$

with right hand side such that the solution is $u = e^{xy}\sin(\pi x)\sin(\pi y)$. The stopping criterion is $\| r^k \|_2 \; / \; \| r^0 \|_2 \leq 10^{-6}$ and initial guess is 1.

Table 5.6 shows the iteration counts for no preconditioning, MGMF1a and MGMF2 preconditionings. We see that for k relatively large or relatively small, MGMF2 gives very good convergence rates and is consistently better than MGMF1a and much better than without preconditioning. For medium values of k, however, MGMF1a appears to be better even though both preconditioners seem to be not very effective.

5.4 MF Preconditioners for Biharmonic Equation

Consider the following biharmonic equation in 2D :

$$-\triangle^2 u = f \text{ in } \Omega = [0,1]^2$$

with second boundary conditions :

$$u(x,y)|_\Gamma = 0 \quad and \quad \frac{\partial^2 u}{\partial n^2}|_\Gamma = 0.$$

Table 5.6: Numerical Results - iteration counts

grid	a	CGNR	CGNR-MGMF1	CGNR-MGMF2
32×32	0	474	41	28
32×32	10	595	40	30
32×32	100	210	50	53
32×32	1000	196	49	36
32×32	10000	64	22	16
32×32	100000	34	14	12
32×32	1000000	31	14	12
64×64	0	1867	59	30
64×64	10	2152	55	31
64×64	100	534	50	55
64×64	1000	402	85	64
64×64	10000	257	36	27
64×64	100000	127	21	16
64×64	1000000	64	16	13

We discretize this equation using 13-point second-order centered finite difference approximation with $h = \frac{1}{n+1}$, and obtain a sparse matrix A. The eigenvalue spectrum of $\hat{A}$ can be approximated by :

$$\hat{A}(j,k) = (4 - 2(\cos(i\pi h) + \cos(j\pi h))^2$$

which is the square of that of the Poisson equation.

Since the eigenvalues in B_l for this equation behave like $O(h_l^{-4})$, a natural extension of the MF preconditioner involves changing the scaling recurrence $c_{l+1} = 4c_l$ to $c_{l+1} = 16c_l$ (again, $h_l \approx 2h_{l+1}$ is assumed). In Table 5.7, we show the result of the Fourier analysis on the MF-preconditioned biharmonic equation. In the table, SGMF1c, SGMF2c and SGMF3c represent the original SGMF1, SGMF2 and SGMF3 preconditioners with the new scaling.

Table 5.7: Condition number for SGMF preconditioning for biharmonic equation

n	*No preconditioning*	*SGMF1c*	*SGMF2c*	*SGMF3c*
7	690	25	5.3	17
15	1.1×10^4	108	5.6	66
31	1.7×10^5	438	7.2	256
63	2.8×10^6	1814	8.7	1017
127	4.4×10^7	7367	10.2	4061
255	7.0×10^8	29705	11.7	16238

We see that the condition number of A grows about 16 times with each halving of h. The use of SGMF1c has effectively helped to reduce the condition number. Nevertheless, SGMF2c helps to reduce the condition number even more dramatically.

In our numerical experiments, we implement the SGMF1c, SGMF2c and SGMF3c preconditioners for the Biharmonic equation with first boundary condition (i.e. $u|_\Gamma$ and $\frac{\partial u}{\partial n}|_\Gamma$ are given). The discretization using 13-point second-order centered difference approximation gives the following difference equations for the grid points not close to the boundary:

$$\begin{aligned} 20u_{i,j} &- 8(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}) \\ &+ 2(u_{i+1,j+1} + u_{i-1,j+1} + u_{i+1,j-1} + u_{i-1,j-1}) \\ &+ u_{i+2,j} + u_{i-2,j} + u_{i,j+2} + u_{i,j-2} = h^4 f_i, j \end{aligned}$$

for $i, j = 2, n-1$. If we let the boundary condition be $u = g(x, y)$ and its first derivative be $b(x, y)$. Then the difference equation for $i = 1$, and $j = 3, \cdots, n-2$ (grid point next to the boundary) is :

$$\begin{aligned} 21u_{1,j} &- 8(u_{2,j} + u_{1,j+1} + u_{1,j-1}) + 2(u_{2,j+1} + u_{2,j-1}) + u_{3,j} + u_{1,j+2} + u_{1,j-2} \\ &= h^4(f_i, j + 8g_{0,j} - 2(g_{0,j+1} + g_{0,j-1}) - 2hb_{0,j}) \end{aligned}$$

since

$$\frac{\partial u}{\partial n} = -\frac{\partial u}{\partial x} \text{ on } x = 0,$$

and using central differencing, we get

$$-\frac{(u_{1,j} - u_{-1,j})}{2h} = b_{0,j}.$$

Also, at $i = j = 1$ (grid points at the four corners), we have

$$\begin{aligned} 22u_{1,1} &- 8(u_{2,1} + u_{1,2}) + 2(u_{2,2}) + u_{3,j} + u_{1,3} \\ &= h^4(f_i, j + 8(g_{0,j} + g_{1,0}) - 2(g_{0,j+1} + g_{0,j-1} + g_{2,0}) - 2h(b_{0,1} + b_{1,0})). \end{aligned}$$

The difference equations for other near boundary grid points can be derived similarly.

The right hand side $f(x,y)$ and boundary conditions $g(x,y)$ and $b(x,y)$ are such that the solution is $u = x(x-1)y(y-1)\sin(\pi x)\sin(\pi y)$. The stopping criterion is $\| r^k \|_2 / \| r^0 \|_2 \leq 10^{-6}$ and the initial guess is zero. The iteration counts are shown in Table 5.8.

Table 5.8: Iteration Counts for SGMF-preconditioned PCG for biharmonic equation

n	*No preconditioning*	*SGMF1c*	*SGMF2c*	*SGMF3c*
7	10	9	10	9
15	42	17	12	16
31	160	36	14	30
63	586	82	17	57
127	2218	177	23	113
255	8587	366	33	220

Next we show (in Table 5.9) the iteration counts when the multigrid formulation of SGMF1c, SGMF2c and SGMF3c (i.e. MGMF1c, MGMF2c and MGMF3c) are applied to the same problem.

We observe a close correlation between the numerical and Fourier results for the SGMF preconditioners. Indeed, SGMF2c improves significantly over SGMF1c with only a little increase in cost per iteration. SGMF3c improves somewhat over SGMF1c but is still not good enough compared to SGMF2c. Therefore, SGMF2c requires the least operation counts out of the three. Looking into the numerical results for the MGMF preconditioners, we first observe that both MGMF1c and MGMF3c give better convergence rates than their SGMF counterparts. We cannot

Table 5.9: Iteration Counts for MGMF-preconditioned PCG for biharmonic equation

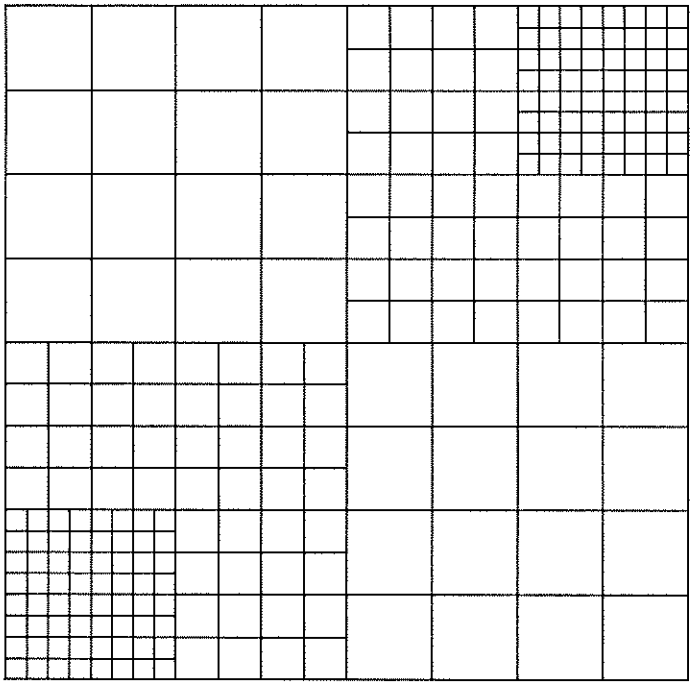
n	*No preconditioning*	*MGMF1c*	*MGMF2c*	*MGMF3c*
7	10	10	10	10
15	42	27	22	24
31	160	40	29	32
63	586	56	30	37
127	2218	80	35	40
255	8587	120	43	48

explain why this is the case. Finally, with a little arithmetic, it is not difficult to show that MGMF3c gives the least overall operation counts.

5.5 MF Preconditioners for Problems with Locally Refined Grids

In this section, we shall consider the application of the MF preconditioners to second-order elliptic problems with local mesh refinement. Such mesh refinements are necessary for accurate modeling of problems with various type of singular behavior. We consider the discretization scheme for locally mesh refined grids by McCormick and Thomas [84]. This discretization scheme was motivated by the desire to preserve the highly regular grid structure (to maintain efficiency on parallel computer architectures) as well as to satisfy the need for local resolution in many physical models. For example, the mesh in Fig. 5.1 would be effective if the forcing function $f(x, y)$ behaves like a δ function distribution at the points $(1, 1)$ and (n, n) (both lower left and upper right corners).

Figure 5.1: Locally Refined Grids - An Example



The Fourier analysis cannot be applied here because of the presence of nonuniform grids. However, as was shown in our previous paper [76], the parallel multilevel preconditioner proposed by Bramble, Pasciak and Xu [22] can be considered as a special case of MF preconditioners with appropriately chosen filters. We can borrow the finite element analysis result from them and we would expect the MGMF preconditioners to be effective also for meshes with local refinement. Below we show the MGMF algorithm for this problem. Here $\hat{I}_l^j$ and $\hat{H}_l$ are restriction (or interpolation) and elementary filtering operators restricted to the locally refined grids only. Moreover, we can use the same recurrence relation $c_l = 4c_{l+1}$ and we have the following algorithm:

Algorithm MGMF1d : input $= r$, output $= z = M^{-1}r$

Decomposition :

$v_L := r$

(* filtering at refined levels *)

for $l = L-1, \cdots, J-k$

$v_l := \hat{I}^l_{l+1} \hat{H}_{l+1,1} v_{l+1}$

end for

(* filtering on uniform grid levels *)

for $l = L-k-1, \cdots, 1$

$v_l := I^l_{l+1} H_{l+1,1} v_{l+1}$

end for

Scaling :

for $l = 1, \cdots, L$

$v_l := v_l \div c_l$

end for

Synthesis :

$z_1 := v_1$

for $l = 2, \cdots, L-k$

$z_l := v_l + H_{l,1} I^l_{l-1} z_{l-1}$

end for

for $l = L-k+1, \cdots, L$

$z_l := v_l + \hat{H}_{l,1} \hat{I}^l_{l-1} z_{l-1}$

end for

$z = z_L$

end MGMF1d

We solve a Poisson equation on the grid

- shown in Fig. 5.1 but with refinement only at the upper right corner and the forcing function is $f(x,y) = 2^{-l}\delta(1-h, 1-h)$, and
- shown in Fig. 5.1 and the forcing function is $f(x,y) = 2^{-l}(\delta(h,h) + \delta(1-h, 1-h))$ where l is the number of level of refinements used and h is the grid size for the nonrefined grid.

We use the discretization scheme for the domain and the interfaces proposed by McCormick [84] for aligned grid. The stopping criterion and initial guess are the same as before. The iteration counts for different number of levels and different h are given in Table 5.10 and 5.11. The iteration counts for unpreconditioned CG method and the parallel multilevel preconditioner (BPX) [22] are also included for comparison purpose.

The tables show the effectiveness of the MF preconditioner compared to the unpreconditioned CG method and the PCG method with parallel multilevel preconditioner. The convergence rates seem to be quite insensitive to the number of refinement levels used.

5.6 Conclusion

In the last chapter we show the competitiveness of the MF preconditioners compared with other preconditioners such as the hierarchical basis preconditioner, MG preconditioner, etc. In this chapter we have further demonstrated the ease with which we can extend the MF preconditioners to effectively solve other more general elliptic problems. The flexibility of filter and scaling block design offers different ways of achieving high degree of efficiency for these problems.

Table 5.10: Iteration Counts for Poisson equation with refinements at upper right corner only

n	no. of levels	*CG*	*MGMF1*	*BPX*
15	0	26	9	12
15	1	37	10	14
15	2	45	11	16
15	3	53	12	17
31	0	48	9	13
31	1	70	10	15
31	2	88	11	17
31	3	109	12	18
63	0	84	10	14
63	1	126	11	15
63	2	166	11	17
63	3	210	12	19
127	0	133	10	14
127	1	219	11	15
127	2	309	12	17
127	3	395	13	19

Table 5.11: Iteration Counts for Poisson equation with refinements at both corners

n	no. of levels	*CG*	*MGMF1*	*BPX*
15	0	26	9	12
15	1	54	11	15
15	2	63	12	17
15	3	75	16	18
31	0	48	9	13
31	1	86	11	16
31	2	117	13	17
31	3	140	13	19
63	0	84	10	14
63	1	126	12	16
63	2	190	12	18
63	3	235	14	19
127	0	133	10	14
127	1	204	12	16
127	2	297	13	18
127	3	391	14	20

CHAPTER 6

Multilevel Preconditioners and Domain Decomposition Methods

6.1 Introduction

In the previous two chapters we consider a multilevel filtering preconditioner for second-order and fourth-order, self-adjoint elliptic partial differential equations. Closely related to the MF preconditioner are the hierarchical basis preconditioner by Yserentant [111] and the parallel multilevel preconditioner (or multilevel nodal basis preconditioner as used in this chapter) by Bramble, Pasciak and Xu [22]. In this chapter we combine the idea of multilevel method with domain decomposition methods to arrive at a new preconditioner.

Domain decomposition refers to a class of methods for solving partial differential equation. The main idea is to decompose the original domain into smaller subdomains, solve the original problem on the subdomains, and somehow "patch" the subdomain solutions to form the solution to the original problem. Besides the ease of parallelization which makes the domain decomposition methods attractive on parallel computers, it also allows one to treat complex geometries or to isolate singular parts of the domain through adaptive refinement. Thus, domain decomposition methods have attracted much attention in recent years [27, 25].

In constructing the new preconditioner, we first consider second-order, self-adjoint, uniformly elliptic partial differential equations on a two dimensional polygonal domain Ω. The problems are solved numerically by using piecewise linear finite elements. The domain is first divided into nonoverlapping subregions Ω_i's which

are further divided into triangular finite elements. We denote H the diameter of a typical subregion and h the diameter of its elements.

We begin with the linear system arising from a discretization of the problem and we first eliminate the variables interior to the subregions Ω_i. The resulting reduced system, the Schur complement, involves only the variables associated with Γ, the set of edges and vertices of the subregions. This system is then solved by a preconditioned conjugate gradient method, where the preconditioner is constructed from certain problems associated with the interfaces $\Gamma_{ij} = \partial\Omega_i \cap \partial\Omega_j$ and vertices and a global coarse problem associated with the vertices.

Many preconditioners have been proposed for the subproblems associated with the edges Γ_{ij}. For example, the method by Bramble, Pasciak, and Schatz [20] uses an operator similar to the square root of the Laplacian operator as the subproblem. Recently Smith and Widlund [95] propose a computationally more efficient hybrid preconditioning method which involves only a simple change of basis (between nodal and hierarchical basis) with the unknowns on the edges Γ_{ij}. They show that the new method has a condition number which grows no faster than $C(1+\log(\frac{H}{h}))^2$, which is comparable to that of the BPS method.

The domain-decomposed preconditioner we consider in this chapter is inspired by the work of Smith and Widlund [95]. In the same way that [95] uses the hierarchical basis on the edges to obtain a domain decomposition method, we use the multilevel nodal basis of Bramble, Pasciak, and Xu [22] applied to the reduced system on the interfaces (i.e. the edges *and the vertices*) to obtain our domain decomposition method. We call this preconditioner the multilevel nodal basis domain decomposition (MNBDD) preconditioner. We derive a proof, similar to the proof by Smith and Widlund [95], such that the condition number is bounded

by $O(\log^2 \frac{H}{h})$ for smooth coefficient problems. Numerical experiments, however, show that the condition number appears to be $O(1)$ for the model problem. The computational cost of our method is about the same as that of Smith and Widlund's method.

6.2 The Multilevel Nodal Basis Algorithm and Domain Decomposition Methods

In this section we provide the necessary background to define the MNBDD algorithm. We consider a second-order, self-adjoint, uniformly elliptic, bilinear form $a(u,v)$ on Ω with Dirichlet condition on $\partial\Omega$:

$$a(u,v) = (f,v) \quad \forall v \in H_0^1(\Omega), \quad u \in H_0^1(\Omega).$$

Let $V^H(\Omega)$ and $V^h(\Omega)$ be the spaces of continuous, piecewise linear functions, on the two triangulations, which vanish on $\partial\Omega$. We use elements which obey certain regularity assumptions, and obtain the discrete variational problem:

$$a(u_h,v_h) = (f,v_h) \quad \forall v_h \in V^h(\Omega), \quad u_h \in V^h(\Omega).$$

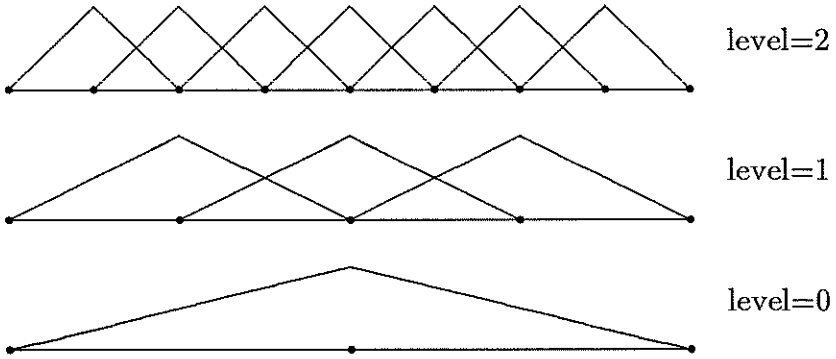
By introducing the standard nodal basis $\{\phi_i\}$ for the space V^h, the above finite dimensional variational problem is reduced to a linear system:

$$Kx = b.$$

Here x is the vector of unknowns x_i, b is the vector of components (f,ϕ_i), and K is the stiffness matrix where $K_{ij} = a(\phi_i,\phi_j)$.

6.2.1 The Multilevel Nodal Basis Algorithm

Figure 6.1: Multilevel Nodal Basis Functions



The multilevel nodal basis method [22] is given in terms of a set of nested sequence of finite element spaces,

$$V_0^h \subset V_1^h \subset \cdots \subset V_J^h \equiv V^h, \quad J = \log_2 \frac{H}{h} \geq 1,$$

which are successive refinements by a factor of two of $V_0^h = V^H$. Here V_i^h is the set of piecewise linear finite element functions after i levels of refinement from the original coarse triangulation. In other words, V_i^h is the set of piecewise linear functions $\{\phi_i^j\}_{j=1}^{n_i}$ $(n_i = dim V_i^h)$ in $V^{2^{J-i}h}$ that satisfies

$$\phi_i^j(y_i^l) = \delta_{jl}, \quad \forall j, l = 1, \cdots, n_i,$$

where $\{y_i^l : l = 1, \cdots, n_i\}$ is the set of all interior nodal points of the triangulation on which V_i^h is defined. In short, $\{\phi_i^j\}_{j=1}^{n_i}$ is the standard nodal basis for the space V_i^h. Figure 6.2.1 shows the multilevel nodal basis functions in one dimension. The multilevel nodal basis preconditioner M of Bramble, Pasciak, and Xu (BPX) [22, 76] applied to $v \in V^h$ takes the following form:

$$M^{-1}v = A_0^{-1}Q_0 v + \sum_{l=1}^{J}\sum_{i=1}^{n_l}(v, \phi_l^i)\phi_l^i \tag{6.1}$$

where the operator A_0 is a discretization of the elliptic operator $-\triangle$ on V_0^h and Q_0 is the standard orthogonal L^2 projection from V_J^h to V_0^h.

The operator M^{-1} involves transformations between standard nodal basis and multilevel nodal basis (the second term in the above equation) as well as the solution of the problem corresponding to the original coarse triangulation (the first term in the equation). In matrix form, this preconditioner $\hat{M}$ can be written as:

$$\hat{M}^{-1} = GD^{-1}G^T$$

where G and G^T are the transformation matrices from multilevel nodal basis to nodal basis and vice versa, respectively; and $D^{-1} = blockdiag[I_{n_J}, \cdots, I_{n_2}, I_{n_1}, A_0^{-1}]$ solves the elliptic problem on the coarse level while leaving the other levels unchanged. It was proved in [22] that for smooth coefficient problems, the condition number of the preconditioned system $\kappa(\hat{M}^{-1}K) \leq CJ^2$ where C is a constant independent of h and H. In addition, this algorithm requires only $O(n_J)$ operations where n_J is the dimension of the finite element space V^h.

The matrix G transforms the input vector from multilevel nodal basis to nodal basis. The dimension of the multilevel nodal basis is $m \equiv n_J + n_{J-1} + \cdots + n_1 + n_0$ while that of the standard nodal basis is $n = n_J$. Thus, G is a rectangular matrix of size $n \times m$ (which is unlike the square transformation matrices for hierarchical basis). Let $v = \{v_l^i, i = 1, \cdots, n_l\}_{l=0}^J \in R^M$ where v_l^i is the value at the nodal point y_l^i corresponding to level l, then

$$Gv = \sum_{l=0}^{J} \sum_{i=1}^{n_l} v_l^i \phi_l^i$$

where $\phi_l^i, i = 1, 2, \cdots n_l$ is the set of basis functions in V_l^h.

The algorithm for G as applied to a vector v (of dimension m) is as follow:

Algorithm $G: v_i \rightarrow u, i = 0, \cdots J$

for $l = 0, \cdots, J-1$

$$v_{l+1} := v_{l+1} + I_l^{l+1} v_l$$

end for

$u = v_J$

end G

Here the I_l^{l+1} matrix is obtained from the choice of ϕ_i's. On a two-dimensional uniform domain using triangular elements, it corresponds to a seven-point interpolation operator.

6.2.2 Domain Decomposition Methods

Domain decomposition methods generally split the space V^h into $N+1$ subspaces

$$V^h = V_{har}^h \oplus V_0^h(\Omega_1) \oplus \cdots \oplus V_0^h(\Omega_N).$$

For each subregion Ω_i, we thus have a subspace $V_0^h(\Omega_i) = V^h \cap H_0^1(\Omega_i)$. The elements of V_{har}^h are piecewise, discrete harmonic functions, i.e. they are orthogonal, in the sense of the bilinear form $a(\cdot,\cdot)$, to all the other subspaces.

First we partition the stiffness matrix K and vector x into those corresponding to the interior of the subregions and the edges and vertices. We then have

$$Kx = \begin{bmatrix} K_I & K_{IB} \\ K_{IB}^T & K_B \end{bmatrix} \begin{bmatrix} x_I \\ x_B \end{bmatrix} = \begin{bmatrix} b_I \\ b_B \end{bmatrix}.$$

If we apply block Gaussian elimination to eliminate the interior points, we obtain the following reduced system or Schur complement for the edges and interfaces

$$S_B x_B = (K_B - K_{IB}^T K_I^{-1} K_{IB}) x_B = b_B - K_{IB}^T K_I^{-1} b_I = \hat{b}_B.$$

The Schur complement matrix S_B is in general dense. However, it is not necessary to generate this matrix since, in the conjugate gradient iteration, this matrix is needed only in terms of matrix vector products which can be computed by solving each subregion once and collecting the solution on the interfaces and vertices.

6.2.3 Multilevel Nodal Basis Domain Decomposition Preconditioner

In this section, we combine the ideas from previous sections to derive a new domain decomposition algorithm. The symmetric form of the preconditioned system using multilevel nodal basis preconditioner can be written as:

$$D^{-\frac{1}{2}}G^T K G D^{-\frac{1}{2}}\tilde{x} = \tilde{b}$$

where $GD^{-\frac{1}{2}}\tilde{x} = x$ and $\tilde{b} = D^{-\frac{1}{2}}G^T b$. Let us partition the unknowns corresponding to the multilevel nodal basis $\tilde{x}$ into those on the subregion interior $\tilde{x}_I$ and those on the interface $\tilde{x}_B$ and we eliminate the subregion interior variables $\tilde{x}_I$, we obtain the reduced system:

$$\tilde{S}_B \tilde{x}_B = \tilde{b}_B,$$

where $\tilde{S}_B$ is the Schur complement of $D^{-\frac{1}{2}}G^T K G D^{-\frac{1}{2}}$ after eliminating x_I. Here we can also decompose G according to the interior and interface unknowns so that

$$G = \begin{bmatrix} G_I & G_{IB} \\ G_{BI} & G_B \end{bmatrix}$$

where G_I and G_B are the transformation (rectangular) matrices involving subregion interior points and interface points respectively.

The above formulation requires many arithmetic operations when the Schur complement is applied to a vector during the conjugate gradient iterations. By using the following lemmas, the above system can be reduced to a simpler form.

Lemma 6.1 *G represents a change of basis which leaves the space of variables on Γ invariant (i.e. $G_{BI} = 0$).*

Proof : G_{BI} represents the contribution of the multilevel nodal bases in the subdomain interior to the nodal basis on the interfaces during the transformation. Recall from previous sections that

$$u \equiv Gv = \sum_{l=0}^{J} \sum_{i=1}^{n_J} v_l^i \phi_l^i \tag{6.2}$$

where $\{\phi_l^i, i = 1, 2, \cdots n_l\}$ is the set of basis functions in V_l^h, and $v_l^i, i = 1, \cdots, n_l$ is the set of values defined on the nodal points x_l^i. Let $u = (u_I, u_B)^T$ and $v = (v_I, v_B)$ be the partitionings according to subdomain interior and interfaces where $u \in R^n$ and $v \in R^m$. If we evaluate the above expression at node y_J^i on the interface Γ, we obtain

$$u_B^i = (G_B v_B)^i + (G_{BI} v_I)^i = \sum_{l=0}^{J} \left\{ \sum_{j \in \Gamma} (v_B)_l^j (\phi_l^j)_i + \sum_{j \notin \Gamma} (v_I)_l^j (\phi_l^j)_i \right\}$$

where $(\phi_l^j)_i$ is the value of the basis function at node j evaluated at node i on level l. It can be verified that

$$(\phi_l^j)_i = 0 \quad \forall j \in \Omega_k \text{ and } i \in \Gamma$$

since all multilevel nodal basis functions at the nodes interior to the subregions vanish on the interface. In other words, the second term in the summation of equation 6.2 is identically equal to 0. Thus $G_{BI} = 0$ and $u^i = (G_B v_B)^i$.

The following lemma is based on a similar one from Smith and Widlund's paper [95]. The proof can be obtained by a straightforward matrix manipulation.

Lemma 6.2 *Let $G_{BI} = 0$ and $D = blockdiag(D_I, D_B)$, then*

$$\tilde{S}_B = D_B^{-\frac{1}{2}} G_B^T S_B G_B D_B^{-\frac{1}{2}}.$$

The above two lemmas imply that if we first eliminate the interior variables and then transform to the multilevel nodal basis, we will be solving the same linear system as before (transformation to multilevel nodal basis and then do the elimination). As a result, we are solving the following simpler and smaller system:

$$D_B^{-\frac{1}{2}} G_B^T S_B G_B D_B^{-\frac{1}{2}} \tilde{x}_B = \tilde{b}_B,$$

which is equivalent to solving the Schur complement system $S_B x_B = b_B$ with the preconditioner $M^{-1} = G_B D^{-1} G_B^T$. We call the preconditioner using this new formulation the multilevel nodal basis domain decomposition (MNBDD) preconditioner. This method offers several possible advantages over the standard BPX method. The conjugate gradient iteration is carried out over a much smaller set of unknowns and we will show that the condition number is smaller. The solution of the subproblems on the interfaces is easily parallelizable since they are independent. One possible drawback, however, is that now it is necessary to solve each subregion exactly in each iteration which adds more computational overhead.

Here the algorithm for G_B algorithm is similar to the one for G shown before except now the basis functions used in the evaluation are restricted only to those on the interfaces. The operation of G_B on a vector v_B is defined by:

$$G_B v_B = \sum_{j=0}^{J} \sum_{i=1}^{m_j} (v_B)_j^i \phi_j^i$$

where $\phi_j^i, i = 1, 2, \cdots m_j$ is the set of basis functions on Γ.

The MNBDD algorithm can be summarized as follow:

Algorithm MNBDD : input $= r$, output $= z = M_B^{-1} r$

Perform partial change of basis to the MN basis $v = G_B^T r$

solve the coarse grid problem $y = D_B^{-1} v$

Perform change of basis back to the nodal basis $z = G_B y$

end MNBDD

We need one more lemma from [95].

Lemma 6.3 *Let K be symmetric and positive definite. Then the condition numbers of K and its Schur complement satisfy*

$$\kappa(Schur(K)) \leq \kappa(K).$$

Using Lemma 6.3 as well as the condition number bounds from [22], we arrive at the following main theorem:

Theorem 6.1 $\kappa(M_B^{-1} S_B) \leq O(\log^2(\frac{H}{h}))$ *for smooth coefficient problems.*

Proof : By using Lemma 6.1 and 6.2 we obtain

$$\tilde{S}_B = D_B^{-\frac{1}{2}} G_B^T S_B G_B D_B^{-\frac{1}{2}}.$$

By using Lemma 6.3, we obtain

$$\begin{aligned} \kappa(D_B^{-\frac{1}{2}} G_B^T S_B G_B D^{-\frac{1}{2}}) &= \kappa(\tilde{S}_B) \\ &\leq \kappa(D^{-\frac{1}{2}} G^T K G D^{-\frac{1}{2}}), \end{aligned}$$

which is bounded by $O(\log^2 \frac{H}{h})$, see [22].

It is also proved in [22] that the condition number bound in Theorem 6.1 depends also on the elliptic regularity of the problem. For example, for smooth coefficient problems on convex polygonal domains, the condition number is bounded by $O(\log \frac{H}{h})$; and for certain discontinuous problems, the condition number is bounded by $O(\log^3 \frac{H}{h})$.

The hierarchical basis domain-decomposed (HBDD) [95] algorithm is similar to the BPS algorithm by Bramble, Pasciak and Schatz [20]. The only difference is that the HBDD uses hierarchical basis preconditioner for the edges while the BPS uses variants of Dryja's preconditioner [37]. The MNBDD algorithm, however, has one important difference; namely, in addition to the use of multilevel nodal basis as preconditioners on the edges, the MNBDD algorithm also implements multilevel nodal basis on the vertices. This introduces some redundancy on the vertices. This redundancy may be the reason for its improved performance (see next section).

6.3 Numerical Results

6.3.1 Two-subdomain Example

We use the two-subdomain example of [95] with different right hand side and we compare our results with those reported in [95]. We also include results from using the Dryja's ($l_0^{1/2}$) [37] preconditioner. We use the domain $\bar{\Omega} = \bar{\Omega}_1 \cup \bar{\Omega}_2$ where $\bar{\Omega}_1$ and $\bar{\Omega}_2$ are unit squares aligned along an edge $\Gamma = \bar{\Omega}_1 \cap \bar{\Omega}_2$. We use the standard uniform mesh and the usual five point discretization for the Laplacian and the iteration counts are listed in Table 6.3.1 (where n is the number of unknowns on the interface). The right hand side is such that the solution is $x(x-1)y(y-1)$ and the stopping criterion is when the relative 2-norm of the residual falls below 10^{-6}. The initial guess used is $u^{(0)} = 1.0$. We observe that while the iteration count for the HBDD continues to grow with larger n, the other preconditioners seem to be bounded independent of n. While the iteration counts using the HBDD preconditioners grow with n, we see that the iteration counts using MNBDD seem to gradually levels off. Overall, we see the MNBDD preconditioner performs relatively well compared to the others.

Table 6.1: **iteration count versus n**

n	No precond	Dryja	MNBDD	HBDD
8	4	4	4	4
16	8	6	7	7
32	16	6	9	8
64	27	6	9	10
128	39	6	9	12

6.3.2 Many-subdomain Example

Next we consider the case of many subdomains. The unit square Ω is subdivided uniformly into $k \times k$ square subdomains and the same model problem is solved using uniform meshes. We compare our results with a set of experiments reported in [95]. We also compare the condition numbers as well as iteration counts between our method, Smith and Widlund's method, and Bramble, Pasciak and Schatz's method (BPS). For HBDD method, it is reported in [95] that the contribution of the coarse problem should be scaled by a factor $\alpha \approx 3.6$ to achieve fastest convergence. Inspired by Smith and Widlund's effort in search for an α that gives fast convergence, we also explore a variant of the preconditioner by Bramble, Pasciak and Xu [22] (equation 6.1) defined by:

$$M^{-1}v = \alpha A_0^{-1} Q_0 v + \sum_{l=1}^{J} \sum_{i=1}^{n_l} (v, \phi_l^i) \phi_l^i. \tag{6.3}$$

It can be shown that this modified preconditioner has the same order of condition number bound as the unmodified one (equation 6.1) provided that α is independent of h and H. The α that gives fastest convergence in our experiment is 0.5. The numerical results using this α are also included under the method MNBDD(0.5).

The right hand side f is constructed such that the solution is $u(x,y) = x(x-1)y(y-1)$ and the stopping criterion is 10^{-5}. Again, the initial guess is $u^{(0)} = 1.0$. The results are shown in Table 6.3.2 where the results for HBDD(3.6) is obtained using $\alpha = 3.6$.

Our first observation is that our condition number results for the HBDD method agree very well with those reported in [95]. We also observe that the condition numbers using the MNBDD method are much lower than the BPS and HBDD methods. The condition numbers grow very slowly with n while this is not the case with the BPS and HBDD methods. With the use of the scaling factor $\alpha = 0.5$, we observe an even better performance and in fact, the condition numbers appear to be $O(1)$.

Through the numerical experiment on the model problem, we have showed that the MNBDD preconditioner offers good convergence rates (better than BPS and HBDD for the model problem used in our experiment) as well as low computational cost ($O(n)$ for MNBDD and HBDD and $O(n \log n)$ for BPS). The BPS, HBDD, and MNBDD algorithms have a similar property that the edge preconditioners do not take into account the variation of coefficients in the original equation. It is not clear whether the good condition number behavior will hold for problems with variable coefficients and it is our intention to pursue this issue in the near future.

Table 6.2: Condition Numbers and iteration counts for the Many-subdomain Case

grid	number of subdomains	HBDD(3.6)		BPS		MNBDD(1)		MNBDD(0.5)	
		κ	iter	κ	iter	κ	iter	κ	iter
32x32	2x2	9.62	11	11.85	11	2.27	8	2.24	7
32x32	4x4	7.96	11	8.75	14	2.88	9	2.19	8
32x32	8x8	5.30	10	6.08	12	3.09	9	2.10	7
64x64	2x2	12.68	13	16.47	12	2.34	8	2.32	8
64x64	4x4	11.84	13	13.03	15	2.96	9	2.28	8
64x64	8x8	8.52	12	9.79	15	3.21	10	2.21	8
64x64	16x16	5.41	10	6.32	13	3.21	10	2.11	7
128x128	4x4	16.49	15	17.92	18	3.01	9	2.35	8
128x128	8x8	12.54	15	14.18	16	3.30	10	2.35	8
128x128	16x16	8.69	13	10.21	15	3.31	10	2.24	8
128x128	32x32	5.42	10	6.36	13	3.22	10	2.11	7
256x256	4x4	21.90	17	23.45	19	3.03	9	2.39	8
256x256	8x8	17.30	17	19.33	18	3.34	10	2.43	8
256x256	16x16	12.72	15	14.79	17	3.37	10	2.36	8
256x256	32x32	8.69	13	10.27	15	3.30	10	2.24	8
256x256	64x64	5.37	10	6.37	12	3.21	10	2.09	7

CHAPTER 7

Performance Analysis of MF preconditioners on the CM

7.1 Introduction

This chapter investigates the performance of the multilevel filtering preconditioners on the CM. We first compare the timing performance of the MF preconditioners with other preconditioners on two- and three-dimensional Poisson problems, 2D variable coefficient problem, and biharmonic equation. It is well known that the V-cycle multigrid-type algorithms encounter idle processor problem on massively parallel computers [78]. This problem is especially true on the CM because of insufficient software capability to detect processor activities, the result of which is large amount of overhead incurred. Techniques (such as on chip striding and virtual processor striding) have been proposed before [78] to alleviate the idle processor problem. Since such techniques have not been incorporated into the microcode, we derive timing models to study performance improvements using the striding mechanisms. Using the timing models, we show that the execution time decreases monotonically with the increase in the number of processors used, implying that the MF preconditioners can exploit the massively parallelism without degradation in the execution time.

7.2 Implementation of MF Preconditioners on the CM

7.2.1 The Conjugate Gradient Method on the CM

The implementation issues of the CG method for two- and three-dimensional second-order elliptic problems have been discussed in detail in Chapter 2. The major difference between implementing second-order problems and fourth-order problems such as the biharmonic equation lies in the matrix vector multiplication which for the latter is a 13-point stencil. A straightforward implementation of the biharmonic operator thus requires 12 communication steps. This expensive communication can be reduced by treating the biharmonic operator as a cascade of two Laplace operators which requires only 8 communication steps. Another point is that the stencils for the grid points near the boundary are slightly different from those of the interior grid points. This may pose an efficiency problem on the CM since it is a SIMD machine and inhomogeneous operations have to be performed at different time steps. Fortunately the first and second boundary conditions (boundary conditions with first and second derivatives respectively) can be handled nicely with one extra multiplication and one extra addition for the grid points near the boundary. Consequently the parallel operation count for the matrix vector multiplication for biharmonic equation is 20 (9 for each Laplace operator and 2 extra for points near the boundary).

For the problems we solve on the CM, we assume uniform grid with $2^i + 1$ grid points in each dimension for some integer i. It should be mentioned that mapping a general unstructured mesh on the CM efficiently is a nontrivial task and in general will result in degradation of performance. Active research is currently being done to investigate efficient embedding of unstructured meshes on the CM [57].

7.2.2 Implementation of Filters

At each level the MGMF1 filter involves each active processor getting data from its eight neighbors at a power-or-two distance away and computing the weighted sum. A straightforward implementation of this filter requires 8 communication steps. Communication can be reduced by decomposing the 2D filter into a cascade of x- and y-filter (generally called tensor product). Similarly, the 3D MGMF1 filter can be decomposed into cascades of x-, y-, and z-filter). Each x- or y-filter requires two communication steps and four arithmetic operations. Therefore, the tensor product MGMF1 filter requires a total of only 4 communication steps and 8 arithmetic operations. Moreover, the scaling operator can be incorporated into the filter which can effectively reduce the arithmetic operation count to 6 per level. The PARIS (assembly language for the CM) code for the MGMF1 filter is as follow:

```
/* get data from neighbors in x direction at distance of 2^k away */
/* forall active processors */
    call cm_get_from_power_two_1l(temp1,src,0,k,0,wl)
    call cm_get_from_power_two_1l(temp2,src,0,k,1,wl)
    call cm_f_add_2_1l(temp1,temp2,ml,el)
    call cm_f_multiply_constant_2_1l(temp1,0.5,ml,el)
    call cm_f_add_3_1l(dest,temp1,src,ml,el)

/* get data from neighbors in y direction at distance of 2^k away */
/* forall active processors */
    call cm_get_from_power_two_1l(temp1,src,1,k,0,wl)
    call cm_get_from_power_two_1l(temp2,src,1,k,1,wl)
    call cm_f_add_2_1l(temp1,temp2,ml,el)
```

```
call cm_f_multiply_constant_2_1l(temp1,0.5,ml,el)
call cm_f_add_2_1l(temp1,src,ml,el)
call cm_f_add_2_1l(dest,temp1,ml,el)
```

The program code above can be further optimized by combining some of the arithmetic operations to form a composite operation (e.g. put add and multiply together which is faster than two single operations due to less memory access).

7.2.3 Implementation of Interpolation

Interpolation can also be decomposed into x- and y-interpolation to reduce the communication. 4 communication steps and 4 arithmetic operations are needed to implement this tensor product interpolation. The interpolation algorithm is described in the following:

```
/* x-interpolation at distance of 2^k away */
/* forall active processors */
    call cm_get_from_power_two_1l(temp1,src,0,k,0,wl)
    call cm_get_from_power_two_1l(temp2,src,0,k,1,wl)
    call cm_f_add_2_1l(temp1,temp2,ml,el)
    call cm_f_multiply_constant_3_1l(dest,temp1,0.5,ml,el)

/* y-interpolation at distance of 2^k away */
/* forall active processors */
    call cm_get_from_power_two_1l(temp1,src,1,k,0,wl)
    call cm_get_from_power_two_1l(temp2,src,1,k,1,wl)
```

call cm_f_add_2_1l(temp1,temp2,ml,el)

call cm_f_multiply_constant_3_1l(dest,temp1,0.5,ml,el)

7.2.4 Selection of Active Processors

The number of active processors decreases as coarser and coarser grids are traversed, and determining which processors should be enabled or not at various levels can be a cumbersome task. This can be dealt with elegantly by declaring beforehand two variables in each processor to store the x- and y- addresses of the processor. By examining the bit pattern of the x- and y- addresses, active processors can be determined in a straightforward and efficient way.

7.3 Timing Results on the CM

Table 7.1 shows the timing and MFLOPS results for the MGMF1 preconditioner compared to other preconditioners for 2D Poisson problem. The value k in MGMF1-k in the tables indicates the number of levels used. We first observe that the excellent convergence rates (reflected in the iteration counts) and the relatively high degree of parallelism (reflected in the MFLOPS) of the MGMF1-6 give the best timing among all preconditioners implemented. For the 1024×1024 grid, for example, better than a factor of two improvement is seen compared to the unpreconditioned CG method.

The column 'DP time' shows the execution times when double precision arithmetic is used instead of the fast single precision arithmetic. Since the MMGMF preconditioners are communication intensive, we see less drastic deterioration in performance compared to other preconditioners (a factor of 10-20 compared to 60-80). The result is that the MGMF1-7 preconditioners become about 10 times

Table 7.1: PCG for 2D Poisson Problems - CM Statistics (16k CM-2)

precond.	grid	no. iter.	time(s)	MFLOPS	DP time(s)
CG(none)	512x512	783	6.6	589	491
RILU	512x512	54	447.0	1.0	—
Jacobi-2	512x512	384	6.2	491	385
Jacobi-4	512x512	270	7.5	451	423
LS-2	512x512	364	8.0	384	498
LS-3	512x512	258	7.1	398	436
LS-4	512x512	226	7.6	407	686
MGMF1-5	512x512	46	4.6	74	87
MGMF1-6	512x512	26	4.3	45	60
MGMF1-7	512x512	18	5.1	26	50
CG(none)	1024x1024	1525	40.0	760	—
RILU	1024x1024	—	—	—	—
Jacobi-2	1024x1024	748	38.7	641	—
Jacobi-4	1024x1024	527	43.8	606	—
LS-2	1024x1024	711	46.6	512	—
LS-3	1024x1024	503	41.5	534	—
LS-4	1024x1024	441	43.6	552	—
MGMF1-6	1024x1024	47	20.4	68	—
MGMF1-7	1024x1024	27	18.5	43	—
MGMF1-8	1024x1024	18	20.7	26	—

faster than the unpreconditioned CG. It will be interesting to compare the communication overhead between different preconditioners. Using the timing data given in this chapter, it is straightforward to estimate this communication overhead.

Also in the table (say for 1024×1024 grid), MGMF1-7 gives the best execution time even though the maximum number of levels is 10. This can be explained by realizing that at the tenth level, only one processor is active and the convergence rate improvement by going all the the way to the tenth level may not be great enough to compensate for the extra time consumed in traversing to the tenth level. In Figure 7.1, we show the iteration counts versus the number of levels used and we can see that even though the iteration count continues to decrease as more levels are used, the decreasing rate also slows down with more levels. In Figure 7.2 we show the execution times for different grids versus number of levels used and we observe that for the 2D Poisson problem, the optimal number of levels seems to be $\log_2 n - 3$ where $n - 1$ is the number of unknowns in each of the x- and y-dimension.

Table 7.2 shows the timing and MFLOPS results for 3D Poisson problem. We observe that the MGMF1 preconditioners still outperform the others. The improvement, however, is not as good as that for the 2D case since now filtering and interpolation require more communication steps and arithmetic operations.

At a glance, it seems that even though the MGMF1 performs better than the other preconditioners, the improvement is too small to be significant. In the following sections, we will show how this timing performance for the MGMF preconditioners can be further improved.

The conjugate gradient method without preconditioning is indeed very efficient on the CM. About 760 MFLOPS is observed on a 16k-processor CM-2. If we

Table 7.2: PCG for 3D Poisson Problems - CM Statistics (16k CM-2)

precond.	grid	no. iter.	time(s)	MFLOPS
CG(none)	32x32x32	66	0.35	142
RILU	32x32x32	17	8.61	2.5
Jacobi-2	32x32x32	33	0.34	121
Jacobi-4	32x32x32	24	0.47	107
LS-2	32x32x32	30	0.45	87
LS-3	32x32x32	22	0.43	91
LS-4	32x32x32	18	0.43	93
MGMF1-2	32x32x32	30	0.29	108
MGMF1-3	32x32x32	19	0.35	57
MGMF1-4	32x32x32	16	0.54	31
MGMF1-5	32x32x32	15	0.85	19
CG(none)	64x64x64	130	3.43	229
RILU	64x64x64	22	65.8	3.3
Jacobi-2	64x64x64	65	3.32	195
Jacobi-4	64x64x64	46	4.35	177
LS-2	64x64x64	60	4.36	144
LS-3	64x64x64	43	4.05	150
LS-4	64x64x64	37	4.29	154
MGMF1-2	64x64x64	58	2.51	194
MGMF1-3	64x64x64	29	2.47	98
MGMF1-4	64x64x64	19	2.91	55

assume a linear scale-up to the full 64k-processor machine, the performance will be about 3 GFLOPS. Currently there are still some viable ways to boost the performance further. One of them is the use of slicewise storage scheme (each 32-bit data is stored across 32 processors) instead of fieldwise storage scheme (each 32-bit data is stored within one processor) to exploit more efficiently the pipelining feature. Another way is the use of the stencil library which will further enhance the performance for matrix vector multiplication. With these features incorporated in our experiment, better performance is expected.

7.4 Performance Analysis for 2D Second-order Self-Adjoint Elliptic Problems

7.4.1 Techniques for Performance Improvements

There are two techniques that can dramatically improve the performance of both grid communication and arithmetic for the MGMF preconditioners. The first technique was called 'VP striding' in [78]. When the number of grid points is larger than the number of available physical processors, each physical processor has to simulate a number of grid points (VP) by time slicing. This idea is called 'virtual processing' and the number of grid points simulated on each physical processor is called 'VP ratio'. During the MGMF preconditioning when coarser levels are traversed, only some of the VPs need to do arithmetic and communication. However, even with the VP context flags (a logical flag within each virtual processor to enable or disable the processor) disabled for some processors, the CM microcontroller still issues the instructions (arithmetic and communication) to all VP's. The context flags are used only to determine whether to store the result to the destination. Thus, even some of the arithmetic and communication are not

needed, the CM microcontroller still allocates time to do such useless task.

A second technique was called 'on-chip striding' in [78]. The idea is similar to VP striding except that now when even coarser levels are traversed, there is a point when some of the physical processors are idle. Since there are 16 physical processors on one integrated circuit chip and they share the same hypercube channel to communicate to other chips, time slicing is used to send data from the 16 processors offchip. At a certain coarse level, even though only some of the physical processors on the chip need to communicate offchip, the microcontroller does not have the knowledge of this and it will allocate time for each processor to communicate. This introduces some wasteful communication time.

The implementation of the techniques above requires modification to the CM microcode and currently they have not been implemented. In order to study the performance improvement using these techniques, timing models are developed in this chapter for second-order and fourth-order problems.

7.4.2 Timing Model

Before deriving the timing model, let us define some parameters :

s = FLOPS / processor

n = no. of unknowns in each dimension (2D)

$F(i,m)$ - time for commmunication at dist=2^i in dimension m

p = number of physical processors used

I = number of iterations needed for convergence

k = number of levels used

T = total execution time.

We use the communication model formulated by Levit [78] as follow:

$$F(i,m) = \begin{cases} t_I \prod_j v_j + t_E d \prod_{j \neq i} v_j c_j & 2^i < c_m v_m \\ 16 t_E \prod_j v_j & 2^i = c_m v_m \\ 32 t_E \prod_j v_j & 2^i > c_m v_m \end{cases}$$

where t_I and t_E denote the on-chip and off-chip communication time per data respectively, and c_j's denotes the on-chip dimensions (i.e. the way the 16 processors are arranged on chip. For example, the 16 processors can be arranged in 4×4 2D array so that c_1 and c_2 are both 4). Similarly, v_j denotes the VP dimensions (i.e. thc way the virtual processors are arranged within each physical processor. For example, if VP ratio is 8 and the virtual processors are configured as 4×2 2D array, then v_1 and v_2 are 4 and 2 respectively).

This communication model basically classifies the grid communication into short-range, medium-range, and long-range communications. For short-range communication, some of the processors need only to communicate on chip and since on-chip communication is fast, the communication time is shorter. For medium-range communication, all of the processors need to communicate off-chip but the destination chip is only one hop away in the hypercube. Finally, for the long-range communication, all of the the processors need to communicate to another chip which is at a distance of 2 away in the hypercube and thus this communication takes twice the time as the medium-range communication.

We perform experiment on the CM to obtain the timing statistics for the 'cm_get_from_power_two_1l' instruction and the results seem to agree with the model. Using the data from Table 7.3, t_I and t_E can be approximated to be 32 and 28 microseconds respectively.

Table 7.3: cm_get_from_power_two_1l timing - CM Statistics (16 CM-2)

grid	log2(distance)	time (ms)
256x256	0	0.4
256x256	1	0.6
256x256	2	1.0
256x256	3	1.8
256x256	4	3.5
256x256	5	3.5
1024x1024	0	2.7
1024x1024	1	3.8
1024x1024	2	6.0
1024x1024	3	10.3
1024x1024	4	16.5
1024x1024	5	28.6
1024x1024	6	55.9
1024x1024	7	55.9
1024x1024	8	56.1

Using the definitions and communication model above, we arrive at the following timing model:

$$T = [\frac{21n^2}{ps} + 3\log_2 p(\frac{1}{s} + t_E) + 2F(0,1) + 2F(0,2) + 4\sum_{m=1}^{2}\sum_{i=0}^{k-2} F(i,m) + \frac{11(k-1)n^2}{ps}] \times I.$$

The first three terms in the model denote the the time to perform the outer CG algorithm while the other terms denote the time to do the preconditioning. Using the above model and the timing data, we plot the execution time predicted by the model and the actual execution time for $n = 256$ and 1024 as shown in Figure 7.3 and 7.4 and the predicted and actual times seem to correlate well.

7.5 Improved Timing Model

Using the VP striding and on-chip striding techniques, we modify the communication model suited for MGMF1 preconditioner as follow:

$$\tilde{F}(i,m) = \begin{cases} (t_I \prod_j v_j)\frac{3}{4^{i+1}} + (t_E \prod_{j \neq i} v_j c_j)\frac{3}{2^{i+1}} & 2^i < c_m v_m \\ (16 t_E \prod_j v_j)\frac{3}{4^{i+1}} & 2^i = c_k v_m \\ (32 t_E \prod_j v_j)\frac{3}{4^{i+1}} & 2^i > c_k v_m. \end{cases}$$

The improved timing model is then :

$$\tilde{T} = [\frac{21n^2}{ps} + 3\log_2 p(\frac{1}{s} + t_E) + 2\tilde{F}(0,1) + 2\tilde{F}(0,2) + 4\sum_{m=1}^{2}\sum_{i=0}^{k-2} \tilde{F}(i,m) + \frac{11(k-1)n^2}{ps}] \times I$$

We also plot the execution time predicted by this improved model in Figure 7.3 and 7.4 and we see a dramatic performance improvement compared to the timings without the striding mechanisms.

In an effort to deal with the idle processor problem, Frederickson and McBryan [45] develop a massively parallel multigrid algorithm which they called the parallel superconvergent multigrid (PSMG) algorithm. We implement this PSMG

algorithm and compare it to the MGMF1 preconditioned PCG method and the results are shown in Table 7.4. In the table, the MGMF1-k* denotes the predicted MGMF1 preconditioner using the improved model. We see that MGMF1-k* improves dramatically over even the PSMG algorithm. For the 1024×1024 grid, the MGMF1-k* is about 20 times faster than the unpreconditioned CG method.

Table 7.4: PCG for 2D Poisson Problems - predicted CM Statistics (16k CM-2)

precond.	grid	no. iter.	time	MFLOPS
CG(none)	256x256	401	1.23	589
Jacobi-2	256x256	197	1.15	337
LS-3	256x256	132	1.34	271
MGMF1-5	256x256	26	1.02	47
MGMF1-8*	256x256	14	0.22	115
PSMG-7	256x256	4	0.85	–
CG(none)	1024x1024	1525	40.0	760
Jacobi-2	1024x1024	748	38.7	641
LS-3	1024x1024	503	41.5	534
MGMF1-7	1024x1024	27	18.5	43
MGMF1-8*	1024x1024	18	2.2	250
PSMG	1024x1024	5	13.9	—

7.5.1 Optimal Number of Processor

Using the improved timing model, we calculate the number of processors P_{opt} that gives the lowest execution time (Figure 7.5 and 7.6). We found that the P_{opt} is

equal to n^2 (VP ratio = 1), showing that the MGMF-preconditioned CG method can indeed exploit the massively parallelism offered by parallel computers such as the CM. However, it should be pointed out that from Figure 5.5 and 5.6, at some point doubling the number of processors results in only a slight improvement in execution time. Therefore, if we take the hardware cost into consideration in addition to execution time, it may not be worthwhile to use n^2 processors afterall.

7.6 Performance Analysis for Biharmonic Equation

7.6.1 Improved Timing Model

The improved timing models for MGMF1- and MGMF2-preconditioned CG method for biharmonic equation are as follow:

$$\tilde{T}_1 = [\frac{32n^2}{ps} + 3\log_2 p(\frac{1}{s} + t_E) + 4\tilde{F}(0,1) + 4\tilde{F}(0,2) + 4\sum_{m=1}^{2}\sum_{i=0}^{k-2}\tilde{F}(i,m) + \frac{11(k-1)n^2}{ps}] \times I,$$

and

$$\tilde{T}_2 = [\frac{32n^2}{ps} + 3\log_2 p(\frac{1}{s} + t_E) + 4\tilde{F}(0,1) + 4\tilde{F}(0,2) + 8\sum_{m=1}^{2}\sum_{i=0}^{k-2}\tilde{F}(i,m) + \frac{25(k-1)n^2}{ps}] \times I.$$

Again the first three terms denote the the time to perform the outer CG algorithm while the other terms denote the time to do the preconditioning. We plot the actual execution time and the predicted time for the original and improved timing model for $n = 256$ as shown in Figure 7.7 and 7.8. Again the predicted and actual times seem to correlate well.

Table 7.5 shows the execution times and MFLOPS for 256×256 and 512×512 grids. Again, the MGMF2* for 256×256 grid denotes the predicted performance using the improved timing model. We observe a hundred-fold improvement over the unpreconditioned CG method.

Table 7.5: PCG for Biharmonic Equation - CM Statistics (16k CM-2)

precond.	grid	no. iter.	time(s)	MFLOPS
CG(none)	256x256	8846	57.3	314
MGMF1-4	256x256	331	8.6	90
MGMF1-5	256x256	170	7.4	57
MGMF1-6	256x256	135	9.9	34
MGMF2-4	256x256	188	9.4	73
MGMF2-5	256x256	73	6.0	45
MGMF2-6	256x256	43	6.1	26
MGMF2*	256x256	43	0.54	293
MGMF3-4	256x256	217	9.8	71
MGMF3-5	256x256	85	6.5	38
MGMF3-6	256x256	49	6.8	21
CG(none)	512x512	34817	659	314
MGMF1-5	512x512	440	51.2	86
MGMF1-6	512x512	251	45.1	55
MGMF1-7	512x512	201	59.6	34
MGMF2-5	512x512	188	41	67
MGMF2-6	512x512	74	25.7	42
MGMF2-7	512x512	54	31.5	25
MGMF3-4	512x512	687	86.3	92
MGMF3-5	512x512	208	42.5	56

7.6.2 Optimal Number of Processor

Using the improved timing model, we calculate the number of processors P_{opt} that gives the lowest execution time (Figure 7.9 and 7.10). Again, we found that the P_{opt} is equal to n^2 (VP ratio = 1).

7.7 Summary

The MGMF-preconditioned CG methods are attractive on massively parallel computers such as the CM. The importance of the hypercube network on the CM for making the MGMF preconditioners efficient cannot be undermined. Since the MGMF preconditioners are communication intensive, it is important to have an efficient interconnection network and relatively low communication to computation ratio.

A final point is that by using the communication improvement techniques, we believe that the V-cycle multigrid method should perform better than the massively parallel multigrid algorithms such as the PSMG. Indeed, the architectural characteristics of the CM (many processors on one chip, many processors share one communication channel, virtual processing, etc.) are likely to be inherited by the future massively parallel hypercube computers (if hypercube interconnection continues to be used in future massively parallel systems) due to the advent of VLSI technology and the realization that communication hardware is expensive. If this is the case, then the striding techniques will continue to find useful application in V-cycle multilevel algorithms.

Figure 7.1: Iteration Count versus no. of levels used (n=256)

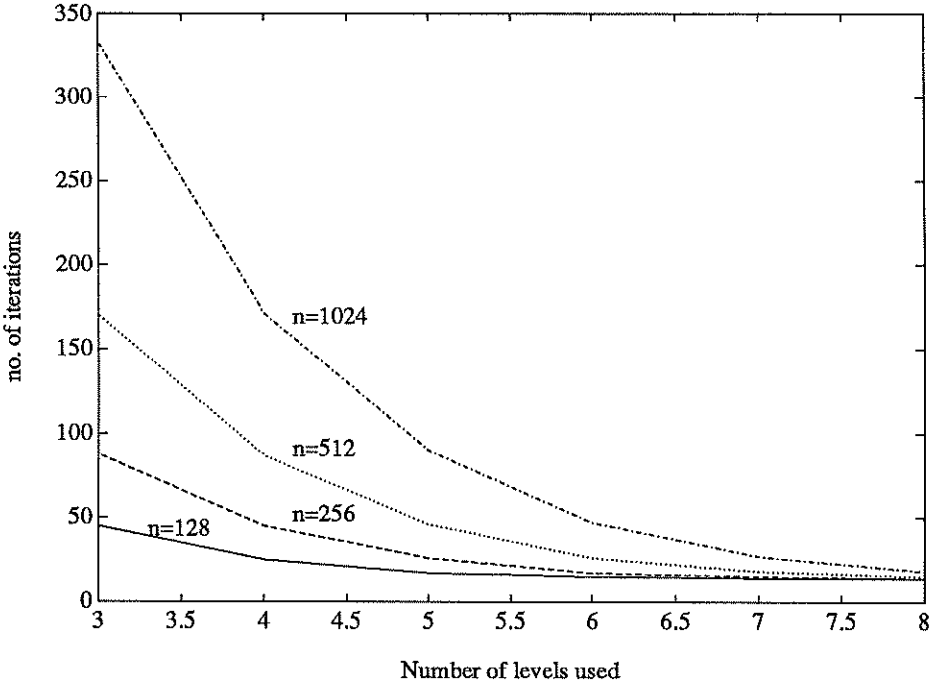


Figure 7.2: CM Time versus no. of levels used (n=256)

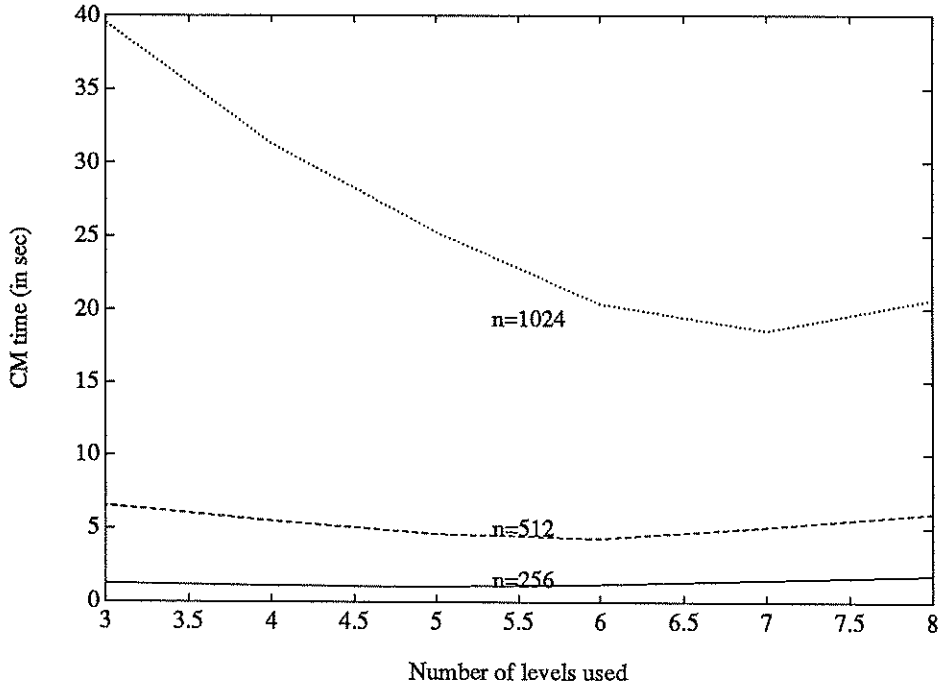


Figure 7.3: Predicted and Observed Times for Poisson Problem(n=256)

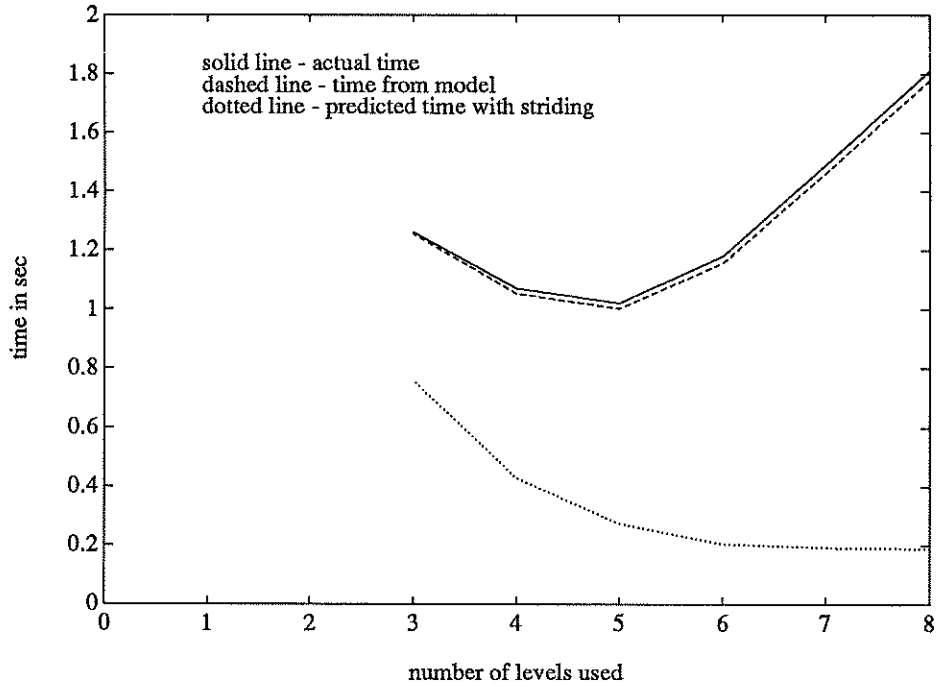


Figure 7.4: Predicted and Observed Times for Poisson Problem(n=1024)

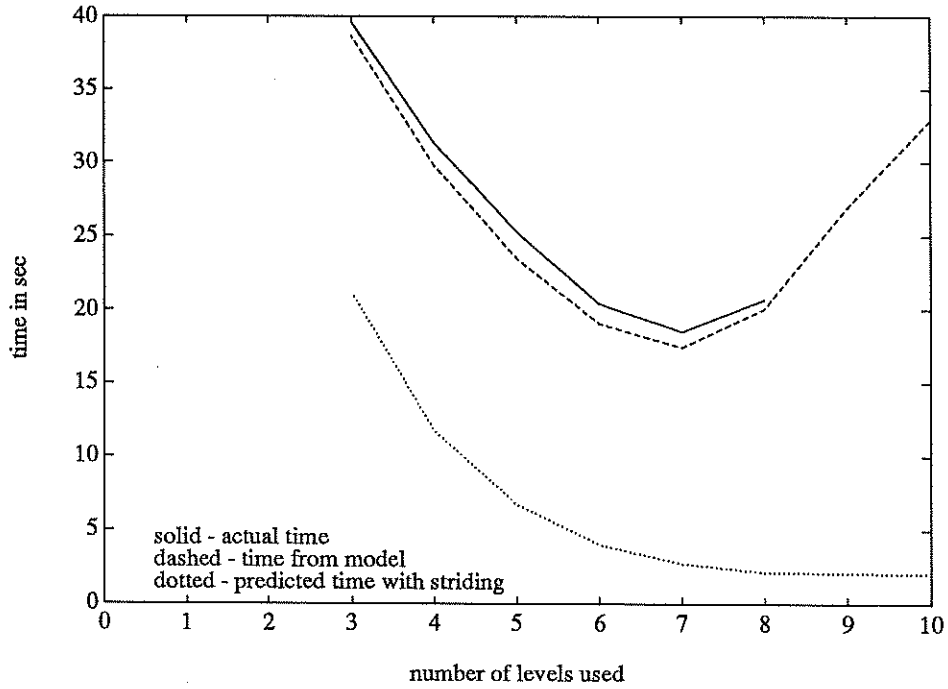


Figure 7.5: Predicted CM Times for Poisson Equation(n=256)

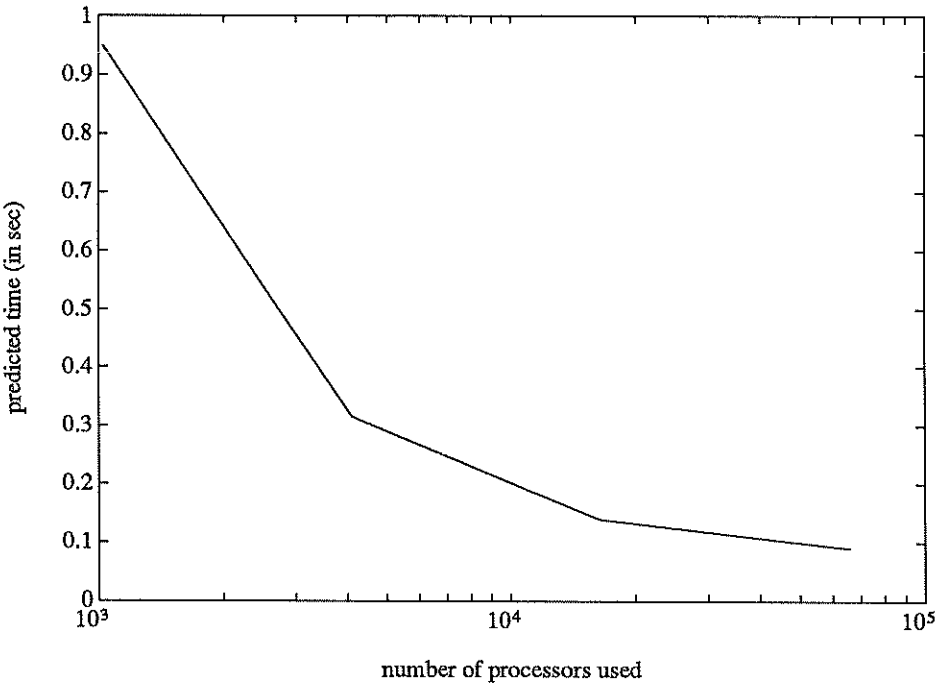


Figure 7.6: Predicted CM Times for Poisson Equation(n=1024)

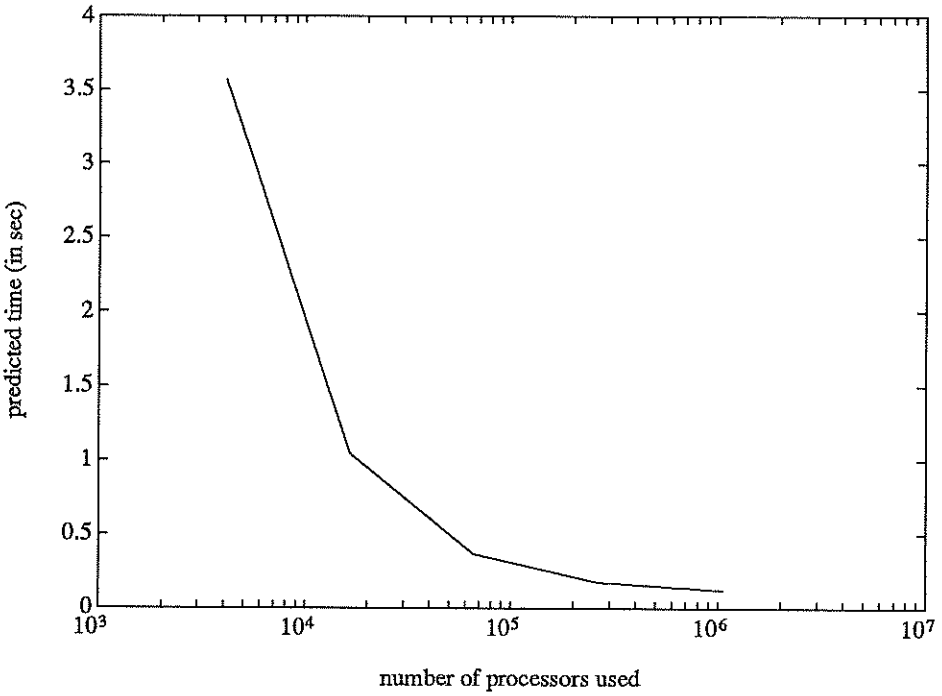


Figure 7.7: CM Times for Biharmonic Equation with MGMF1(n=256)

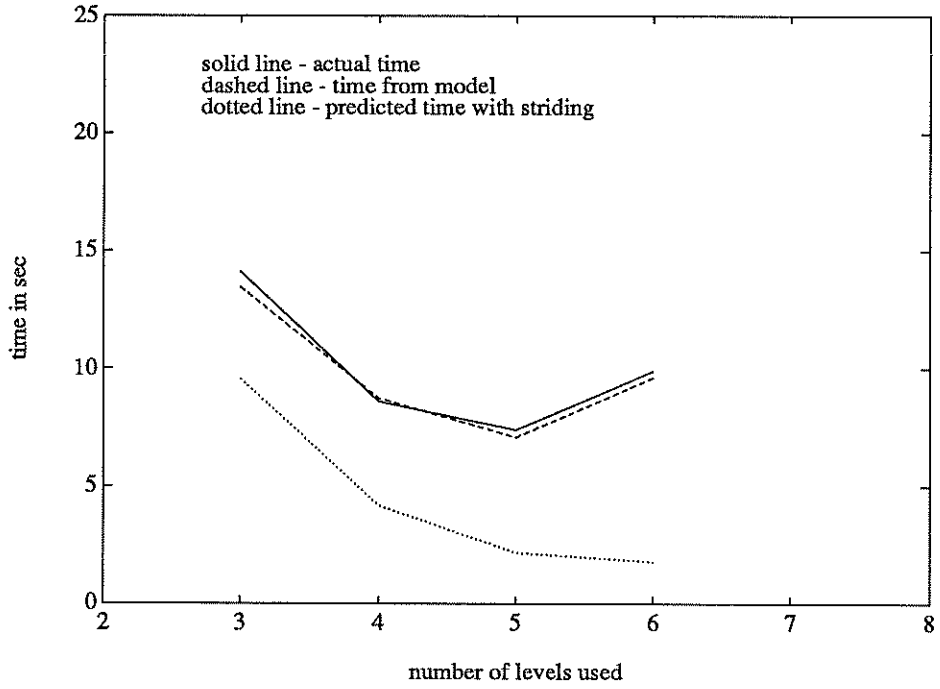


Figure 7.8: CM Times for Biharmonic Equation with MGMF2(n=256)

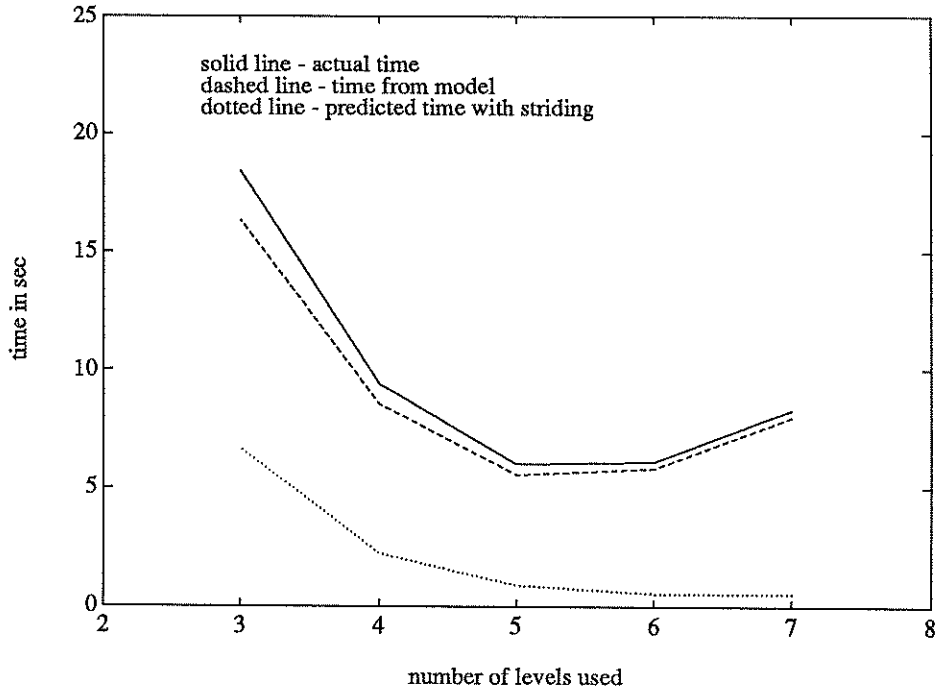


Figure 7.9: Predicted Times for Biharmonic Eqn. with MGMF1 (n=256)

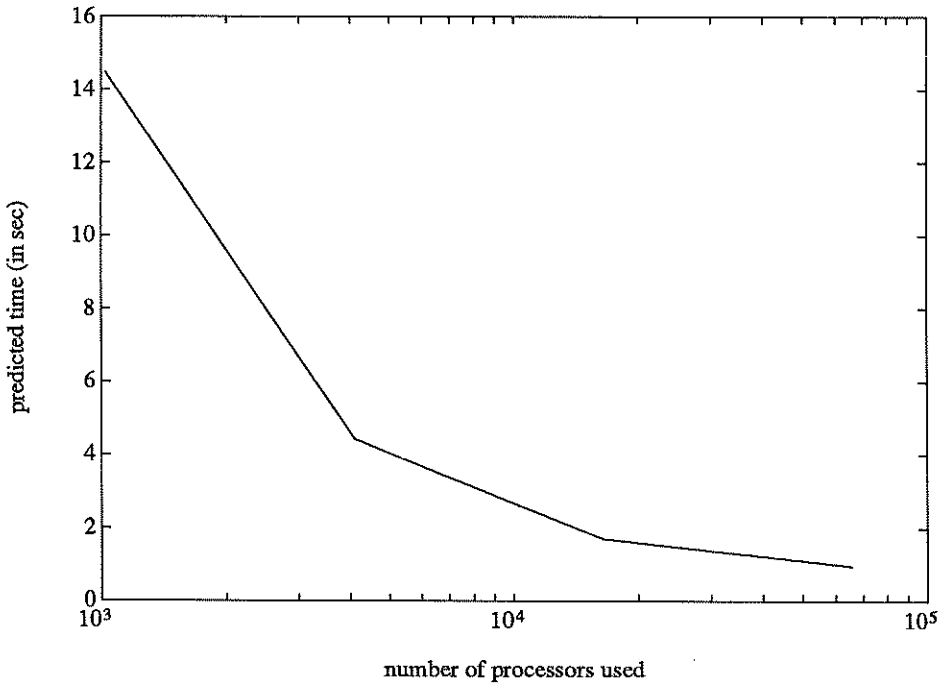
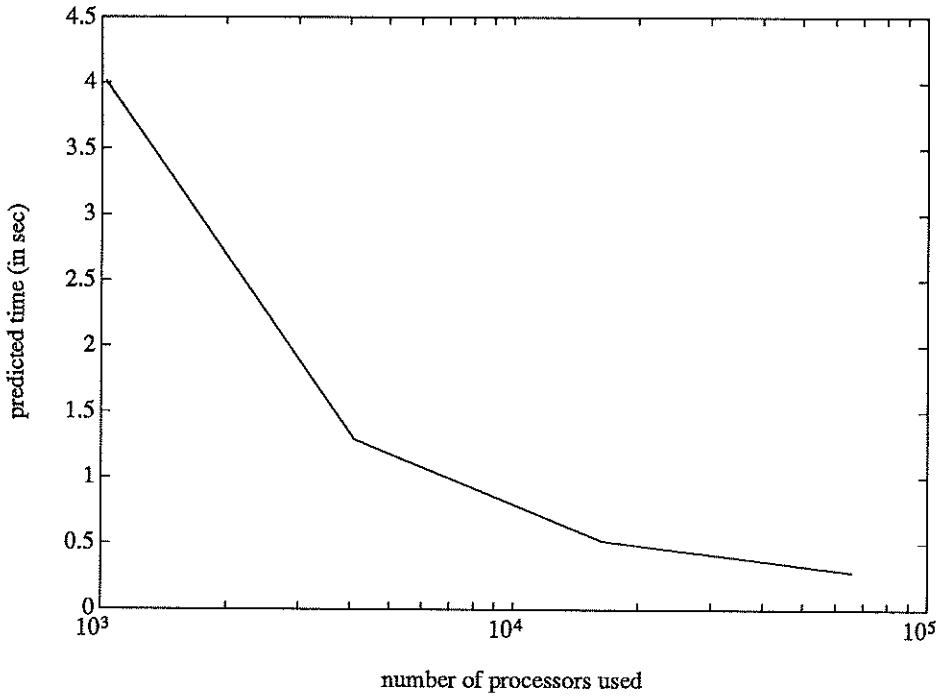


Figure 7.10: Predicted Times for Biharmonic Eqn. with MGMF2(n=256)



CHAPTER 8

FFT as a Massively Parallel Multilevel Algorithm

8.1 Overview

8.1.1 Overview on Massively Parallel Multilevel Algorithms

The global dependence property of elliptic problems, which translates into slow convergence rates for local-type iterative methods, has motivated us to look for efficient numerical methods in the class of multilevel algorithms, and in particular, multilevel preconditioners for the conjugate gradient method. We have since developed a class of MF preconditioners and have shown its efficiency on a massively parallel computer for some second- and fourth-order elliptic problems. In fact, in many situations the most efficient algorithms for the numerical solution of large elliptic problems are multilevel algorithms, e.g. the various multigrid algorithms [24, 53, 98].

When implemented on massively parallel computers where the number of processors is $O(N)$ and N is the number of unknowns, however, the multilevel algorithms encounter the idle processor problem at the coarse grids. For example, at the coarsest grid, only one grid point (or one processor) needs to be updated.

In view of this problem, McBryan and Frederickson [45] developed a massively parallel multigrid algorithm (the Parallel Superconvergent Multigrid or PSMG) which gives better convergence rates and keeps all processors busy all the time so that the overall computational time is reduced (they implemented the PSMG on

the CM and showed its efficiency over the regular Multigrid method.)

Along the same line, but in a different spirit, is the development of the Robust Multigrid method (RMG) by Hackbusch [54]. The motivation behind the RMG method is to improve the robustness of the MG algorithm for a wider class of problems. It happens that the RMG method possesses the massive parallelism just like the PSMG algorithm.

The MF preconditioners also have their massively parallel version (called SGMF), as described in Chapter 4. However, SGMF preconditioners require complex communication pattern near the boundary (due to odd symmetry property) which spreads to the interior at the coarse grids. This makes the SGMF preconditioners relatively inefficient on massively parallel computers such as the CM as they require the slow router to handle the complex communications rather than the much faster NEWS network.

Recently, several researchers [101, 44] have looked upon the FFT algorithm as a multigrid-type algorithm. FFT exhibits communication pattern similar to massively parallel V-cycle multilevel algorithms. FFT has been used widely in the numerical solutions of partial differential equations. Some of the examples are the Fast Poisson solver and the solution of Burgers' equation using spectral method. For example, consider the following 1D Burgers' equation

$$u_t + uu_x = \mu u_{xx}.$$

If upwind differencing is used to discretize the u_t term and spectral method is used to discretize the u_x and u_{xx} terms, then we have the following :

$$\frac{u^{(n+1)} - u^{(n)}}{\Delta t} - iu^{(n)}FKFu^{(n)} = -\mu FK^2Fu^{(n)}$$

or

$$u^{(n+1)} = u^{(n)} + i\Delta t F K F u^{(n)} - \Delta t F K^2 F u^{(n)}$$

where $u^{(n)}$ is a $n \times 1$ vector and $K = diag(0, 1 \cdots n-1)$.

Here FFT forms the core of the algorithm to solve the above recurrence. The rest of this chapter focuses on the development and implementation of an efficient ordered FFT algorithm on the CM.

8.1.2 Overview on Ordered FFT

The increased availability of various parallel architectures poses many challenges for algorithm development. One notable example is the Fast Fourier Transform (FFT) with many variants that are targeted for different types of computers. The main difference between these variants is the order of the intermediate sequences which have been selected to favor certain architectural characteristics. For example, orderings that result in long vectors with unit stride are selected for vector computers [99]. Orderings that minimize communication are selected for hypercube multiprocessors [100]. Interprocessor communication is the major source of performance degradation on hypercube multiprocessors.

In this chapter we examine efficient implementation of radix-2 ordered DIF (decimation in frequency) FFTs on massively parallel hypercube computers such as the Connection Machine (Any mention of FFT in this chapter should be assumed to mean radix-2 DIF FFT unless otherwise indicated). The concept of an ordered transform is expanded to include any transform in which the ordering of the input sequence matches that of its transform. This is a reasonable consideration on the CM where orderings can be selected with equal ease by the specification of geometries and priorities. Two "ordered" transforms are considered, namely,

standard-order [100] (or consecutive-order as used in [66]) and cyclic-order ([66] or scatter-order as used in [42]) transforms. These transforms differ in communication complexity and their suitability will likely depend on the application. If a standard-order transform is not required then an cyclic-order transform with less communication may be appropriate.

The standard-order transform was considered earlier [100] where it was demonstrated that a sequence with $N = 2^r$ elements could be transformed with $r/2+d+1$ parallel transmissions on a hypercube with $P = 2^d$ processors if $d > r/2$. Here we show that a cyclic-order transform can be computed with $2d - r/2$ parallel transmissions. Both orderings belong to the class of orderings called index-digit permutations [43]. In this chapter we are interested only in using binary digits. Besides reducing the amount of communication, we also show that this algorithm facilitates the parallel computation of the trigonometric coefficients without evaluating the trigonometric functions or interprocessor communication. Although we will consider only "ordered" transforms in the expanded sense, it is important to note that an unordered transform can be computed with only d parallel transmissions.

In section 2, we begin with a class of orderings called binary index-digit permutations. In particular, we review the concept of i-cycle which is central to the implementation of ordered hypercube FFT as well as the general index-digit permutation. In particular, we examine the standard-order and cyclic-order FFT. In section 3, we first discuss different ways of computing the trigonometric coefficients and then present a new parallel method for the direct computation of the trigonometric coefficients. Next we show that this method is particularly suited to a hypercube implementation using i-cycles. The performance results of these

FFTs are presented in section 4. Finally, we include an error analysis of the new method for computing the trigonometric factors.

8.2 Parallel Hypercube FFTs

8.2.1 Introduction

We consider the implementation of ordered FFTs on hypercube multiprocessors. It is assumed that the number of physical processors is $P = 2^d$ where d is the dimension of the hypercube. Each processor has its own local memory (also called distributed-memory system). It is also assumed that the number of elements to be transformed is $N = 2^r$ and that N/P is a small constant (massively parallel version of the original hypercube FFT [100]). Moreover, if $N/P > 2$ (number of elements is more than twice the number of physical processors), the elements are mapped to virtual processors which then contain exactly two elements, (after the CM model). It is known that interprocessor communication consumes a substantial amount of time and hence its minimization is of primary concern. Communication between virtual processors located in the same physical processor does not contribute to interprocessor communication. Throughout the text we will use the following notation.

If x_n has $N = 2^r$ elements then it can be mapped into the multidimensional array $x(i_{r-1}, \cdots, i_0)$ where $i_{r-1}i_{r-2}, \cdots, i_0$ is the binary form of n. The FFT can then be loosely described as a sequence of 2^{r-1} transforms of length two in each of r dimensions. An example for the case $N = 16$ is given in Table 8.1.

The original sequence is given as the first entry in Table 8.1. The transform in the dimension i_3 is designated by replacing i_3 by k_3 in the second entry. Subsequent multiple 1-D transforms correspond to subsequent entries in Table 8.1. The FFT

Table 8.1: Intermediate Orderings for Cooley-Tukey FFT, N=16, using Subscript Notation

$x(i_0, i_1, i_2, i_3)$
$X^{(1)}(i_0, i_1, i_2, k_3)$
$X^{(2)}(i_0, i_1, k_2, k_3)$
$X^{(3)}(i_0, k_1, k_2, k_3)$
$X^{(4)}(k_0, k_1, k_2, k_3)$
$X^{(4a)}(k_3, k_2, k_1, k_0)$

requires the multiple 1-D transforms to be computed in the order of decreasing indices, i.e., i_3, i_2, i_1, and i_0. The last entry corresponds to the bit-reversal that is necessary to order the FFT. Between each of the multiple 1-D transforms the sequence x_n is multiplied by certain roots of unity. For example, $X^{(1)}(i_0, i_1, i_2, k_3)$ is computed from

$$X^{(1)}(i_0, i_1, i_2, 0) = x(i_0, i_1, i_2, 0) + x(i_0, i_1, i_2, 1)$$

$$X^{(1)}(i_0, i_1, i_2, 1) = \omega^{i_0 i_1 i_2}[x(i_0, i_1, i_2, 0) - x(i_0, i_1, i_2, 1)]$$

where $\omega = e^{-i\pi/4}$.

We will adopt the binary notation in place of the subscript notation to avoid conversions between the two. Table 8.2 is the binary equivalent of the subscript notation that is used in Table 8.1. Element locations are then given directly in binary form.

The last two entries in Table 8.2 correspond to a reordering in which the element in position $k_3k_2k_1k_0$ binary is moved to position $k_0k_1k_2k_3$. This illustrates the advantage of the binary notation which provides the locations directly without

Table 8.2: Intermediate Orderings for Cooley-Tukey FFT, N=16 Binary Notation

$x(i_3i_2i_1i_0)$
$X^{(1)}(k_3i_2i_1i_0)$
$X^{(2)}(k_3k_2i_1i_0)$
$X^{(3)}(k_3k_2k_1i_0)$
$X^{(4)}(k_3k_2k_1k_0)$
$X^{(4a)}(k_0k_1k_2k_3)$

reversing the order of the subscripts.

The last entry in Tables 8.1 and 8.2 is an example of an index-digit permutation [43], called a bit-reversal. Other examples include the perfect shuffle and matrix transpositions. The time required for communication is known to contribute substantially to the overall computing time. It is also known to depend significantly on how the sequence x_n is mapped to the processors. We will begin with perhaps the most common mapping in which the first N/P elements are mapped to the first processor, the second N/P elements are mapped to the second processor and so forth.

Definition 8.1 *A* standard *(or consecutive as in [66]) sequence to processor map* $x(i_{r-1}\cdots i_{r-d} \mid i_{r-d-1}\cdots i_0)$ *is one in which the element* x_n *with* $n = i_{r-1}i_{r-2}\cdots i_0$ *(binary) has* address $i_{r-d-1}\cdots i_0$ *in* processor number $i_{r-1}i_{r-2}\cdots i_{r-d}$ *[100].*

Both a processor number and address are required to identify a particular element in the sequence. The partition | is introduced for expository purposes to separate the address on the right from the processor number on the left. For example if $r = 4$ and $d = 2$ then the element x_n with $n = i_3i_2i_1i_0$ has address

$i_1 i_0$ and is located in processor number $i_3 i_2$ and the mapping is designated by $x(i_3 i_2 \mid i_1 i_0)$.

Definition 8.2 *An (binary)* index-digit permuted *sequence to processor map is one in which the indices i_j are permuted. That is, the element x_n with $n = i_{r-1} i_{r-2} \cdots i_0$ (binary) has address $i_{m(r-d-1)} i_{m(r-d-2)} \cdots i_{m(0)}$ in processor number $i_{m(r-1)} i_{m(r-2)} \cdots i_{m(r-d)}$ where $m(j)$ is an arbitrary permutation of the integers $0, \cdots, r-1$ [100].*

From the last two entries in Table 8.2 it is evident that a method will be needed for converting between index-digit permuted maps on the hypercube. To that end we introduce a specific class of communication tasks.

Definition 8.3 *An (binary)* i-cycle *is an (binary) index-digit permutation of x_n in which the most significant digit of the address (called the* pivot*) is exchanged with any other digit, either in the address or the processor number [100].*

For example, if a standard sequence to processor map is used for x_n, an i-cycle is a reordering that exchanges the digit in position $r-d-1$ with any other digit. Two i-cycle examples are given in Table 8.3.

Table 8.3: Sample i-cycles for the case $d = 2$ and $r = 4$

$X(i_3 i_2 \mid i_1 i_0)$
$X(i_3 i_2 \mid i_0 i_1)$
$X(i_0 i_2 \mid i_3 i_1)$

The second entry in Table 8.3 is obtained from the first by an i-cycle that exchanges the first and second (pivot) digits. The third entry is obtained from the second by an i-cycle that exchanges the second and fourth digits.

For $N = 16$ and $P = 4$ the data exchanges for two sample i-cycles are given in Table 8.4 below.

The i-cycles consist of parallel exchanges of packets with $N/(2P)$ elements. The i-cycle on the left side of Table 8.4 consists of two exchanges. The last two elements in processor 0 are exchanged with the first two elements in processor 1 and the last two elements in processor 2 are exchanged with the first two elements in processor 3. The i-cycle on the right side of Table 8.4 also consists of two exchanges. The last two elements in processor 0 are exchanged with the first two elements in processor 2 and the last two elements in processor 1 are exchanged with the first two elements in processor 3.

The i-cycle has three properties that make it useful for the development of parallel communication algorithms on the hypercube.

I-cycle property A : An i-cycle may or may not require interprocessor communication, depending on whether or not the digit is in the processor number. For example, the first i-cycle in Table 8.3 does not require interprocessor communication because the processor number is unchanged. However the second i-cycle does require interprocessor communication because the processor number is changed. When interprocessor communication is required it is between processors that are directly connected because the processor numbers differ in only one bit. Through this discussion we are assuming that the sequence to processor map is an index-digit permuted map. A direct connection would not be established if the underlying map was (for example) a binary Gray code.

I-cycle property B : It can be shown that at each stage of the FFT the packets transmitted between processors each contains $2^{r-d-1} = N/(2P)$ elements and

Table 8.4: Sample i-cycle communication paths for $N = 16$ and $P = 4$

$X(i_3i_2 \mid i_1i_0)$ $X(i_3i_1 \mid i_2i_0)$			$X(i_3i_2 \mid i_1i_0)$ $X(i_1i_2 \mid i_3i_0)$	
$i_3i_2i_1i_0$	$i_3i_1i_2i_0$	p	$i_3i_2i_1i_0$	$i_1i_2i_3i_0$
0000	0000	0	0000	0000
0001	0001	0	0001	0001
0010	0100	0	0010	1000
0011	0101	0	0011	1001
0100	0010	1	0100	0100
0101	0011	1	0101	0101
0110	0110	1	0110	1100
0111	0111	1	0111	1101
1000	1000	2	1000	0010
1001	1001	2	1001	0011
1010	1100	2	1010	1010
1011	1101	2	1011	1011
1100	1010	3	1100	0110
1101	1011	3	1101	0111
1110	1110	3	1110	1110
1111	1111	3	1111	1111

that $P/2$ packets are exchanged in parallel.

I-cycle property C : Any index-digit permutation can be implemented as a sequence of i-cycles. To see this, first decompose the permutation into disjoint cycles. Next decompose each cycle into i-cycles by interchanging the first position with the pivot position and restore it following the completion of the cycle. For example, if the cycle is (2,8,7,5) and the pivot is in position 3, then this cycle is equivalent to the i-cycles (3,2)(3,8)(3,7)(3,5)(3,2) applied from left to right. Any index permutation can be implemented in no more than $1.5d$ i-cycles [100].

8.2.2 The Standard-order FFT

Consider now the implementation of a standard-order FFT. The i-cycles are given in Table 8.5 for the case $r = 8$ and $d = 5$. The subscripts of the digits are increasing for a transform in standard order like the last entry in Table 8.2. The letter 'a' in the superscript indicates an ordering rather than computational step and an '*' following an entry indicates that a parallel transmission was necessary for that step. The sequence of i-cycles is selected based on the theory presented in [100] where it is shown that for even $r > d/2$ a total of $r/2 + d + 1 = 10$ parallel transmissions are required.

8.2.3 The Cyclic-order FFT

The mapping of a sequence onto the processors is known to significantly influence the time that is required for communication and hence mappings that reduce communication are of considerable interest. The difficulty with selecting a map that minimizes communication for a particular algorithm is that it may not be op-

Table 8.5: Intermediate Orderings for a standard order FFT for $N = 256$ and $P = 32$

$x(i_7 i_6 i_5 i_4 i_3 \mid i_2 i_1 i_0)$
$X^{(1)}(i_2 i_6 i_5 i_4 i_3 \mid k_7 i_1 i_0)^*$
$X^{(2)}(i_2 k_7 i_5 i_4 i_3 \mid k_6 i_1 i_0)^*$
$X^{(3)}(i_2 k_7 k_6 i_4 i_3 \mid k_5 i_1 i_0)^*$
$X^{(4)}(i_2 k_7 k_6 k_5 i_3 \mid k_4 i_1 i_0)^*$
$X^{(5)}(i_2 k_7 k_6 k_5 k_4 \mid k_3 i_1 i_0)^*$
$X^{(5a)}(i_2 k_7 k_6 k_3 k_4 \mid k_5 i_1 i_0)^*$
$X^{(6)}(k_5 k_7 k_6 k_3 k_4 \mid k_2 i_1 i_0)^*$
$X^{(6a)}(k_5 k_7 k_2 k_3 k_4 \mid k_6 i_1 i_0)^*$
$X^{(7)}(k_5 k_7 k_2 k_3 k_4 \mid k_1 k_6 i_0)$
$X^{(7a)}(k_5 k_1 k_2 k_3 k_4 \mid k_7 k_6 i_0)^*$
$X^{(8)}(k_5 k_1 k_2 k_3 k_4 \mid k_0 k_6 k_7)$
$X^{(8a)}(k_0 k_1 k_2 k_3 k_4 \mid k_5 k_6 k_7)^*$

timum for a different part of the overall computation. However without knowledge of the other algorithms, and their optimal maps, it is not unreasonable to permit orderings other than the standard order. If order is not a consideration then it is known that the FFT can be performed with d parallel transmissions. However it is likely that the other parts of the overall computation will expect the order of the transform and the input sequence to be the same, particularly if utilities and subroutines are used. Therefore we define an *ordered transform* as any transform in which the order of the sequence and its transform are the same.

In this section we will consider a variant of the parallel FFT presented above in which the input sequence and transform are cyclic-ordered. Communication is reduced and, as mentioned in the introduction, it is just as simple to select this order as the standard order on the CM using geometry and priorities.

Definition 8.4 *A* cyclic *sequence to processor map* $x(i_{d-1}\cdots i_0 \mid i_{r-1}\cdots i_d)$ *is one in which the element* x_n *with* $n = i_{r-1}i_{r-2}\cdots i_0$ *(binary) has* address $i_{r-1}i_{r-2}\cdots i_d$ *in* processor number $i_{d-1}i_1\cdots i_0$ *[66].*

A cyclic-order FFT is an ordered FFT according to the definition that was given in previous sections and it requires fewer parallel transmissions than a standard-order FFT. An example is given in Table 8.6 for $N = 256$ and $P = 32$. As before, the locations that correspond to the digits on the right of the partition '|' reside in the same physical processor. The digits on the left of the partition correspond to the processor number. An entry that ends with a '*' indicates a parallel transmission and the lines with superscripts that end with a 'a' involve only communication.

The communication complexity for a cyclic-order FFT on parallel hypercube is given in the following lemma.

Table 8.6: Intermediate Orderings for a cyclic-order FFT with $N = 256$ and $P = 32$

$x(i_4 i_3 i_2 i_1 i_0 \mid i_7 i_6 i_5)$
$X^{(1)}(i_4 i_3 i_2 i_1 i_0 \mid k_7 i_6 i_5)$
$X^{(2)}(i_4 i_3 i_2 i_1 i_0 \mid k_6 k_7 i_5)$
$X^{(3)}(i_4 i_3 i_2 i_1 i_0 \mid k_5 k_7 k_6)$
$X^{(4)}(k_5 i_3 i_2 i_1 i_0 \mid k_4 k_7 k_6)^*$
$X^{(5)}(k_5 k_4 i_2 i_1 i_0 \mid k_3 k_7 k_6)^*$
$X^{(5a)}(k_3 k_4 i_2 i_1 i_0 \mid k_5 k_7 k_6)^*$
$X^{(6)}(k_3 k_4 k_5 i_1 i_0 \mid k_2 k_7 k_6)^*$
$X^{(6a)}(k_3 k_4 k_5 i_1 i_0 \mid k_6 k_7 k_2)$
$X^{(7)}(k_3 k_4 k_5 k_6 i_0 \mid k_1 k_7 k_2)^*$
$X^{(7a)}(k_3 k_4 k_5 k_6 i_0 \mid k_7 k_1 k_2)$
$X^{(8)}(k_3 k_4 k_5 k_6 k_7 \mid k_0 k_1 k_2)^*$

Lemma 8.1 *A cyclic-order FFT of length $N = 2^r$ can be implemented on a hypercube of dimension d (where $d > r/2$) with $2d - r/2$ parallel transmissions if r is even and $2d - (r-1)/2$ parallel transmissions if r is odd.*

Proof : The normal i-cycles require d parallel transmissions since every physical processor address digit has to be transferred into the pivot position. The extra i-cycles are performed on the most significant $r/2$ digits, $r/2 - (r-d)$ of which are located in the processor address. Thus, a total of $d + r/2 - (r-d)) = 2d - r/2$ parallel transmissions is needed. A similar proof can be developed for odd r.

The cyclic-order transform in Table 8.6 requires six parallel transmissions compared with ten for the standard-order FFT in Table 8.5. In general the cyclic-order FFT requires anywhere from d to $1.5d$ parallel transmissions and the standard-order FFT requires anywhere from $1.5d$ to $2d$ parallel transmissions. More specifically, for $d > r/2$, the cyclic-order FFT requires $2d - r/2$ transmissions compared to $d + r/2 + 1$ for the standard-order FFT. Therefore the cyclic-order FFT requires $r - d + 1$ fewer parallel transmissions than the standard-order FFT. For the finest grain computations with $d = r - 1$ they differ by only two parallel transmissions. Nevertheless this difference will likely be noticeable because the total communication time is proportional to $O(\log N)$ which is also a small integer.

The FFT is often a part of a larger computation that is posed on a grid so it is reasonable to ask about the compatibility of the Binary Reflected Gray code ordering and cyclic-ordering. In both the standard-order and the cyclic-order transform the processors can be mapped so that nearest neighbors are at a distance of one, but at the expense of the i-cycles being conducted at a distance of two.

8.2.4 The Algorithm

The parallel hypercube FFT algorithm, written in pseudocode (similar to CM FORTRAN) is included in the following. The variable declaration and initialization have not been included.

```
C Parallel Hypercube FFT using the cyclic-order Transform
C k : log2 (n) - 1
Subroutine FFT
do i = k, 0, -1
    if (i≠k) call icycle(i) /* I-cycle */
    call calculate_factor /* Calculate trigonometric factor */
    temp = data1 + data2 /* Compute new data points */
    data2 = (data1 - data2) * twiddle
    data1 = temp
    if (i ¡= n/2.and.i≠0) then /* Extra I-cycles */
        call icycle(n-i-1)
    end if
end do
end
```

8.3 Computing the Trigonometric Coefficients

There are a few alternative methods for computing the trigonometric coefficients depending on the available memory, I/O bandwidth, and processing capabilities [72].

Recursion All of the trigonometric coefficients at each stage are generated by recursion. This scheme requires only $O(1)$ storage and is popular on a uniprocessor or vector processors. However, the computation is highly sequential and not suitable for multiprocessors.

Table look-up The trigonometric coefficients are precomputed and stored in each processor. This scheme has an advantage for many FFTs since the trigonometric coefficients would be available for use without recalculation. However, this scheme also requires a large amount of storage ($O(\log N)$ per processor) for naive implementation. [69], for example, shows how to improve this memory utilization to $O(N + P \log P)$.

Direct calculation The trigonometric coefficients can be computed directly from the equation $W^{-k} = \cos(2k\pi/N) - i\sin(2k\pi k/N)$. However, the calculation of the trigonometric functions on each stage is very time consuming. Particularly since the FFT itself requires only a few operations.

Permutation Initially, the trigonometric coefficients are distributed among the processors according to the calculations required in the first stage. In the subsequent stages, half of the trigonometric coefficients are permuted each to two other processors. This scheme may be inefficient on parallel machine such as the CM where communication is expensive.

None of these methods are completely satisfactory on massively parallel computers if memory is limited and communication is expensive. However, by performing a few additional operations at each stage, the trigonometric coefficients can be computed in parallel without any communication.

Consider the following example of a 16-point FFT (unordered transform) and suppose that element i is mapped to processor i, then the trigonometric factors needed at each stage are as in Table 8.7. The entries in each column correspond to k in the trigonometric factor W^{-k}. Entries with the form (k) refer to the exponent of a coefficient that is not used at the current stage but is needed to compute the coefficients at a subsequent stage of the FFT.

It can be seen that the integers in each column are twice (mod N/2) the integers in the previous column and hence the trigonometric coefficients can be computed from the identities.

$$\cos 2\theta = \cos^2\theta - \sin^2\theta \quad , and$$

$$\sin 2\theta = 2\cos\theta \sin\theta.$$

Thus, we can calculate the trigonometric coefficients for the current stage from the previous stage by four multiplications and one addition (or three multiplications and two additions). This method can also be used to generate the table for the table look-up scheme. It can also be used to compute the coefficients for the ordered (both cyclic-order and standard-order) parallel hypercube FFT presented in section 2 with a slight modification for the initial trigonometric factor calculations. Table 8.8 contains the exponents for the cyclic-order transform with $N = 16$. An initial standard sequence to processor map is assumed.

Fewer computations are required because every trigonometric coefficient is used and therefore a factor of two is saved compared to the unordered FFT. In general, this method of computing trigonometric coefficients can be used if the order of the not-yet-transformed bits (i_j) is preserved. The characteristics of the methods for computing the trigonometric coefficients are summarized in Table 8.9.

Table 8.7: Trigonometric Coefficients for a 16-point unordered FFT

Processor	Value of k in W^{-k}			
Processor Number (binary)	Stage 1	Stage 2	Stage 3	Stage 4
0000	(0)	(0)	(0)	(0)
0001	(1)	(2)	(4)	0
0010	(2)	(4)	0	(0)
0011	(3)	(6)	4	0
0100	(4)	0	(0)	(0)
0101	(5)	2	(4)	0
0110	(6)	4	0	(0)
0111	(7)	6	4	0
1000	0	(0)	(0)	(0)
1001	1	(2)	(4)	0
1010	2	(4)	0	(0)
1011	3	(6)	4	0
1100	4	0	(0)	(0)
1101	5	2	(4)	0
1110	6	4	0	(0)
1111	7	6	4	0

Table 8.8: Trigonometric Coefficients for a 16-point parallel hypercube FFT using cyclic-order and i-cycles

Processor	Value of k in W^{-k}			
Processor Number (binary)	Stage 1	Stage 2	Stage 3	Stage 4
0000	–	–	–	–
0001	–	–	–	–
0010	–	–	–	–
0011	–	–	–	–
0100	–	–	–	–
0101	–	–	–	–
0110	–	–	–	–
0111	–	–	–	–
1000	0	0	0	0
1001	1	2	4	0
1010	2	4	0	0
1011	3	6	4	0
1100	4	0	0	0
1101	5	2	4	0
1110	6	4	0	0
1111	7	6	4	0

Table 8.9: Characteristics of Different Methods for Computing Trigonometric Factors

Method	storage	computation	comm.	comment
recursion	$O(1)$	$O(N \log N)$	0	very sequential
table look up	$O(N + P \log P)$	$O(\log N)$	0	reuseability
permutation	$O(N)$	$O(1)$	$O(\log N)$	–
direct calculation	$O(N)$	$O(\log N)$	0	use sin and cos
new method	$O(N)$	$O(\log N)$	0	no sin and cos

8.4 Performance of the Parallel Hypercube FFTs on the CM-2

8.4.1 Performance results for the CMSSL FFT

Consider first the performance of the CMSSL (CM scientific software library) FFT that is currently available (as of summer 1989) on the CM (located at NASA Ames Research Center) [69, 70]. The execution times of both the ordered and unordered FFT is presented in Table 8.10. FFT (A) and FFT (B) correspond to the unordered and ordered FFTs respectively and the results were obtained on a 32k processor CM-2. Gray code ordered data (or NEWS order) was used in this set of experiment. If, however, binary (or SEND) order is used, a speedup by a factor of 2 is expected. In the table, the entry '–' means that the result could not be computed because it required more memory than what was available. The MFLOPS are computed from the formula MFLOPS $= 5N \log N/$time which does not include the precomputed trigonometric coefficients.

The difference between the time for FFT (A) and FFT (B) is due to the additional communication that is required to bit-reverse the results of FFT(A). From

Table 8.10: Execution times for CMSSL FFT (32k)

size FFT	FFT (A) (sec)	MFLOPS(32k)	FFT (B) (sec)	MFLOPS(32k)
65536	0.02	262	0.03	175
131072	0.04	279	0.08	139
262144	0.09	262	0.22	107
524288	0.17	293	0.56	89
1048576	0.35	300	1.79	59
2097152	0.69	319	6.21	35
4194304	1.40	330	–	–
8388608	2.81	343	–	–

FFT (A) is the CMSSL FFT without bit-reversal

FFT (B) is the CMSSL FFT with bit-reversal

'–' memory was exceeded

the table it is clear that performing bit-reversal is expensive and that performance deteriorates for larger problems. The bit-reversal in the CMSSL FFT that was timed (summer 1989) makes use of the router. The reason for the missing entries ('–' memory exceeded) is that the router requires more temporary storage than the FFT. The newer version of FFT makes use of high-radix index-digit permutations, or all-to-all personalized communication (matrix transposition) as described in [62, 67, 68] which is considerably faster than the router.

8.4.2 Performance of a CM FORTRAN version of the standard-order FFT

In this subsection we will examine the performance of the standard-order FFT using i-cycles in the intermediate phases of the algorithm. The program was written in the beta release version of the CM FORTRAN with partial optimization using compiler options. At present, the system software will use a binary reflected Gray code mapping of the logical processors onto the physical processors. Therefore most i-cycles will communicate over a physical distance (Hamming distance) of two which requires twice the communication of a map in which the logical and physical processors have the same number. The latter case will be discussed in the next subsection.

The execution times and MFLOPS for the FORTRAN version are listed in Table 8.11.

The MFLOPS in Table 8.11 above are calculated from MFLOPS= $7.5N \log N/\mathrm{T}$ (which includes $2.5N \log N$ operations for computing the trigonometric coefficients). Comparing Table 8.10 and 8.11 it can be observed that for small N, the ordered CMSSL FFT is about twice as fast as the standard-order FFT, (e.g. 0.08

Table 8.11: Execution times for the CM FORTRAN standard-order FFT (32k)

size FFT	Execution time (sec)	MFLOPS(32k)
65536	0.08	98
131072	0.16	104
262144	0.32	111
524288	0.66	113
1048576	1.34	117
2097152	2.81	118
4194304	5.67	122
8388608	11.68	124

sec versus 0.16 sec for 131072-point FFT). However for large N, the standard-order FFT using i-cycles outperforms the ordered CMSSL FFT (e.g. 2.81 sec versus 6.21 sec for 2M-point FFT). Also, from Table 8.10, the execution times for FFT (B) triples when the size of the input doubles. On the other hand, from Table 8.11, the execution times for standard-order FFT using i-cycles approximately doubles when the size of the input doubles.

From these comparisons we conclude that the standard-order FFT using i-cycles provides enhanced performance compared to an FFT with separate bit-reversal and butterfly phases. It should be mentioned that the CMSSL FFT was written in lower level languages while the results in Table 8.11 were obtained with a high level language (CM FORTRAN) which is also in its beta release. Thus, further improvement is expected for an implementation in a optimized low level languages or with a mature FORTRAN compiler.

Even though the FFT has been implemented with an efficient communication algorithm using i-cycles, over 80 percent of the execution time is still spent in communication. In the next section, communication will be further reduced by avoiding the binary reflected Gray code mapping of the logical to physical processors.

8.4.3 A Comparison of three FFTs on the CM

In the previous subsection we examined the performance of a CM FORTRAN version of the FFT in which the binary reflected Gray code was used to map logical processors to physical processors. Although this map is ideal for nearest neighbor communication, it slows the i-cycle communication for the FFT by a factor of two. In this section we will consider the performance of three ordered FFTs on a hypercube whose logical and physical processor numbers are the same.

1. The standard-order FFT which combines the bit-reversal and the butterfly phases.

2. The cyclic-order FFT which also combines the bit-reversal and the butterfly phases.

3. An FFT written by Hertz [2] which separates the bit-reversal and the butterfly phases.

Using CM FORTRAN/PARIS it is possible to equate logical and physical processor numbers. That is, any reference to processor $i_{d-1} \cdots i_0$ is a reference to a processor with the same binary representation in the hypercube and not to a processor whose number is the binary reflected Gray code map of $i_{d-1} \cdots i_0$. A significant improvement is obtained because the key communication task (i-cycle)

is conducted at a physical distance of at most one using *news* communication for all i-cycles. The programs were written in CM FORTRAN/PARIS and run on a 32k CM-2. The times for different size FFT are listed in Table 8.12 and the corresponding MFLOPS counts are listed in Table 8.13.

The MFLOPS for (3) is calculated using the same formula as (1) and (2). In reality, method (3) requires more than 7.5 operation per point and thus the MFLOPS count should be higher due to the need for computing the sine and cosine functions at each stage of the FFT.

These results demonstrate the attributes of cyclic-ordering, i-cycles, and the new parallel method of computing the trigonometric coefficients. From Table 8.13, we estimate a performance of about .9 GFLOPS for a 16M-point FFT on a full 64k CM-2.

8.5 Error Analysis of the New Method for Computing Trigonometric factors

In this section we consider the numerical accuracy of our new method for generating the trigonometric factors. Both analytical and numerical results will be presented. We shall begin with a few definitions.

Definition 8.5 *[49] If a number x is represented by the nearest floating point number ($fl(x)$) with precision t (number of digits for the mantissa) and machine base b, then*

$$fl(x) = x(1+\epsilon), \qquad |\epsilon| \leq u$$

where u is the unit roundoff error defined by

$$u = \frac{1}{2}b^{1-t}.$$

Table 8.12: Computing time in seconds for three ordered FFTs

size FFT	machine size	FFT (1)	FFT (2)	FFT (3)
131072	8k	0.22	0.16	–
262144	8k	0.45	0.32	–
524288	8k	0.94	0.67	–
1048576	8k	1.92	1.39	–
2097152	8k	3.95	2.89	–
262144	16k	0.23	0.17	0.688
524288	16k	0.49	0.36	1.40
1048576	16k	1.01	0.72	2.95
2097152	16k	2.07	1.50	6.10
4194304	16k	4.23	3.07	12.68
524288	32k	0.25	0.19	–
1048576	32k	0.52	0.39	–
2097152	32k	1.09	0.80	–
4194304	32k	2.22	1.59	–
8388608	32k	4.55	3.29	–

FFT (1) standard order FFT.

FFT (2) cyclic-order FFT.

FFT (3) P. Hertz FFT. [60]

Table 8.13: MFLOPS for three ordered FFTs

size FFT	machine size	FFT (1)	FFT (2)	FFT (3)
131072	8k	76	104	–
262144	8k	79	111	–
524288	8k	79	112	–
1048576	8k	82	113	–
2097152	8k	84	114	–
262144	16k	154	208	51
524288	16k	152	208	53
1048576	16k	156	218	53
2097152	16k	160	220	54
4194304	16k	164	225	55
524288	32k	299	393	–
1048576	32k	302	403	–
2097152	32k	303	413	–
4194304	32k	318	435	–
8388608	32k	318	440	–

FFT (1) standard order FFT.

FFT (2) cyclic-order FFT.

FFT (3) P. Hertz FFT. [60]

'–' Data not available.

Definition 8.6 *Let c_j^k and s_j^k be the exact required cosine and sine factors for element j at stage k of the FFT, then the corresponding computed $\hat{c}_j^k$ and $\hat{s}_j^k$ satisfies*

$$| \hat{c}_j^k - c_j^k | \leq h(k)u, \quad and$$

$$| \hat{s}_j^k - s_j^k | \leq h(k)u$$

where $h(k)$ is the growth rate of the error for our method.

Assumption : If $\hat{c}_j^k$ and $\hat{s}_j^k$ are computed through direct cosine and sine function calls to the library, then

$$| \hat{c}_j^k - c_j^k | \leq u, \quad and$$

$$| \hat{s}_j^k - s_j^k | \leq u.$$

The algorithm for computing the trigonometric factors is as follow :

Algorithm compute_factor :

/* at stage $k = 1, 2, \cdots$ */

if (k .eq. 1) **then**

parfor $j = 0, 1, \cdots, n-1$

$\hat{c}_j^1 = \cos(\theta_j^1)$ /* direct calls */

$\hat{s}_j^1 = \sin(\theta_j^1)$

end for

else

parfor $j = 0, 1, \cdots, n-1$

$\hat{c}_j^k = \hat{c}_j^{k-1} * *2 - \hat{s}_j^{k-1} * *2$

$\hat{s}_j^k = 2 \times \hat{c}_j^{k-1} \times \hat{s}_j^{k-1}$

$\theta_j^k = 2 \times \theta_j^{k-1}$

if $(\theta_j^k$.ge.$\pi)$ **then**

$\theta_j^k = \theta_j^k - \pi$

$\hat{c}_j^k = -\hat{c}_j^k$

$\hat{s}_j^k = -\hat{s}_j^k$

end if

end for

end if

This method computes the trigonometric factors initially using direct calls to the cosine and sine library functions and by the above assumption,

$$| \hat{c}_j^1 - c_j^1 | \leq u, \quad and$$

$$| \hat{s}_j^1 - s_j^1 | \leq u, \quad \forall i.$$

Lemma 8.2 $| \hat{c}_j^k - (\hat{c}_j^{k-1})^2 + (\hat{s}_j^{k-1})^2 | \leq 2u + u^2.$

Proof :

$$\begin{aligned}
\hat{c}_j^k &= fl(fl(\hat{c}_j^{k-1}\hat{c}_j^{k-1}) - fl(\hat{s}_j^{k-1}\hat{s}_j^{k-1})) \\
&= \left[(\hat{c}_j^{k-1})^2(1+\delta_1) - (\hat{s}_j^{k-1})^2(1+\delta_2)\right](1+\delta_3) \\
&= (\hat{c}_j^{k-1})^2 - (\hat{s}_j^{k-1})^2 + \delta_1(\hat{c}_j^{k-1})^2 - \delta_2(\hat{s}_j^{k-1})^2 + \delta_3(\hat{c}_j^{k-1})^2 - \\
&\quad \delta_3(\hat{s}_j^{k-1})^2 + \delta_1\delta_3(\hat{c}_j^{k-1})^2 - \delta_2\delta_3(\hat{s}_j^{k-1})^2
\end{aligned}$$

where $| \delta_i | \leq u, i = 1, 2, 3$ and which leads to the lemma by noting that

$$| \hat{c}_j^k | \leq 1, \forall j, k.$$

Lemma 8.3 $\mid \hat{s}_j^k - 2\hat{c}_j^{k-1}\hat{s}_j^{k-1} \mid \leq 2u.$

Proof :

$$\begin{aligned} \hat{s}_j^k &= fl(2\hat{c}_j^{k-1}\hat{s}_j^{k-1}) \\ &= 2\hat{c}_j^{k-1}\hat{s}_j^{k-1}(1+\delta_4) \end{aligned}$$

where $\mid \delta_4 \mid \leq u$ and which again leads to the lemma.

Lemma 8.4

$$\mid \hat{c}_j^k - c_j^k \mid \leq 2\sqrt{2}h(k-1)u + 2u + u^2, and$$

$$\mid \hat{s}_j^k - s_j^k \mid \leq 2\sqrt{2}h(k-1)u + 2u + u^2.$$

Proof :

$$\begin{aligned} \mid \hat{c}_j^k - c_j^k \mid &= \mid \hat{c}_j^k - (\hat{c}_j^{k-1})^2 + (\hat{s}_j^{k-1})^2 \mid + \mid (\hat{c}_j^{k-1})^2 - (\hat{s}_j^{k-1})^2 - (c_j^{k-1})^2 + (s_j^{k-1})^2 \mid \\ &\leq 2u + u^2 + \mid (\hat{c}_j^{k-1})^2 - (c_j^{k-1})^2 \mid + \mid (\hat{s}_j^{k-1})^2 - (s_j^{k-1})^2 \mid \\ &\leq 2u + u^2 + \mid \hat{c}_j^{k-1} + c_j^{k-1} \mid\mid \hat{c}_j^{k-1} - c_j^{k-1} \mid + \mid \hat{s}_j^{k-1} + s_j^{k-1} \mid\mid \hat{s}_j^{k-1} - s_j^{k-1} \mid \\ &\leq 2u + u^2 + (\mid \hat{c}_j^{k-1} \mid + \mid c_j^{k-1} \mid + \mid \hat{s}_j^{k-1} \mid + \mid s_j^{k-1} \mid)h(k-1)u. \end{aligned}$$

Now since

$$(\hat{c}_j^{k-1})^2 + (\hat{s}_j^{k-1})^2 = 1, \quad and$$

$$(c_j^{k-1})^2 + (s_j^{k-1})^2 = 1,$$

with a little trigonometry, it is straightforward to show that

$$\mid \hat{c}_j^{k-1} \mid + \mid \hat{s}_j^{k-1} \mid \leq \sqrt{2}.$$

Hence we have,

$$| \hat{c}_j^k - c_j^k | \leq 2\sqrt{2}h(k-1)u + 2u + u^2.$$

The same analysis can be carried out for the computation of sine function.

Finally, we have the following theorem :

Theorem 8.1 *If $\hat{c}_j^k$ and $\hat{s}_j^k$ are computed via Algorithm compute_factor, then*

$$| \hat{c}_j^k - c_j^k | \leq h(k)u + O(u^2), \text{and}$$

$$| \hat{s}_j^k - s_j^k | \leq h(k)u + O(u^2),$$

where $h(k) = \frac{1}{2\sqrt{2}}(2\sqrt{2})^k$.

Proof :

From the previous lemma, we have the following recurrence

$$h(k) = 2\sqrt{2}h(k-1).$$

Solving this recurrence using initial condition $h(1) = 1$ gives

$$h(k) = \frac{1}{2\sqrt{2}}(2\sqrt{2})^k.$$

Corollary 8.1 *The maximum error given by Algorithm compute_factor is* $\frac{1}{2\sqrt{2}}n^{3/2}$.

This is obvious if we notice that k only goes up to $\log_2 n$.

An additional feature about this new method is that if the above error bound is not acceptable, we can restart (using direct calls), say, at the middle (stage $\frac{\log_2 n}{2}$) of FFT and we have the following corollary.

Corollary 8.2 *If the Algorithm compute_factor is restarted stage* $\frac{\log_2 n}{2}$, *then the maximum error incurred is* $\frac{1}{2\sqrt{2}}n^{3/4}$.

This corollary can be easily proved by noticing now k only goes up to $\frac{\log_2 n}{2}$. If this bound is still not acceptable, we can have more restarts to achieve the desired accuracy.

It is given in [31] that the error for the Cooley Tukey FFT is of $O(h^{method} \log_2 n)$ where h^{method} is the error incurred in the trigonometric factor calculations. Hence, we have the final theorem.

Theorem 8.2 *The error bound for the Cooley Tukey FFT using Algorithm compute_factor to compute the trigonometric factors with m restarts is* $O(n^{\frac{3}{2(m+1)}} \log_2 n)$.

In the following we present numerical results using single-precision arithmetic on the CM. A vector consisting of a discrete sine function in the real part and zero in the imaginary part is input to a FFT subroutine and the output is then input to an inverse FFT subroutine, both using the new method. An error vector is generated by taking the difference of the real parts of the original input and the final output. Table 8.14 shows the maximum norm and the root mean square of the error vector. The root mean square value is computed by taking the 2-norm of the error vector and then divided it by the size of the input. Table 8.15 shows the same norms but the new method is used with one restart. We observe a large increase in accuracy with the use of just one restart.

8.6 Summary and Conclusion

First, the experimental results in section 4 demonstrate that performance can be improved by using the ordered parallel FFTs that reduce communication by combining the communication and computational phases [100]. Although this result has been demonstrated on the CM it would also be true for any hypercube

Table 8.14: Error magnitudes using the new method

size FFT N	Maximum norm	2-norm/N
8192	6.43730e-4	3.31699e-6
16384	1.44638e-3	4.34077e-6
32768	3.05253e-3	6.41415e-6
65536	6.69646e-3	8.13486e-6
131072	1.65939e-2	1.24870e-5
262144	3.11155e-2	2.00082e-5
524288	6.67725e-2	2.45073e-5

Table 8.15: Error magnitudes using the new method with 1 restart

size FFT N	Maximum norm	2-norm/N
8192	2.44975e-5	9.07233e-8
16384	3.37362e-5	9.01880e-8
32768	4.61340e-5	8.24786e-8
65536	8.73804e-5	1.00031e-7
131072	1.00136e-4	8.58351e-8
262144	1.65522e-4	9.80134e-8
524288	2.08616e-4	8.93721e-8

because communication time is a significant part of the overall computing time. Second, the cyclic-order FFT has performance that is superior to the standard-order FFT and is therefore recommended where applicable. In addition, a parallel algorithm for computing the trigonometric coefficients was presented that represents an attractive compromise between the communication, computation, and memory constraints that exist on the CM. The use of the i-cycle, cyclic-ordering, and the new parallel algorithm for computing the trigonometric coefficients have resulted in the development of a high performance ordered FFT for the CM.

CHAPTER 9

Conclusion

Conjugate gradient methods, coupled with "effective" preconditioner, are efficient methods for the solution of many systems arising from the discretization of elliptic partial differential equations. In addition to the fast convergence rates, they also offer high degree of parallelism. In particular, on massively parallel computers, the conjugate gradient methods have been shown to be highly efficient.

Since the choice of preconditioner has a large effect on the convergence rate, much research efforts have been to develop effective preconditioners. On the other hands, many preconditioners which are effective in improving the convergence rates have poor inherent parallelism and thus are unable to efficiently utilize the resources offered by massively parallel computers. Unfortunately, these two desirable characteristics, namely fast convergence rates and high degree of parallelism, usually do not come together. We conclude firstly that tradeoffs have to be made between these characteristics in order to arrive at a preconditioner that gives lowest execution time on a given massively parallel machine.

In search of efficient preconditioners on massively parallel machines, we investigate the class of multilevel preconditioners. In particular, we have developed the class of multilevel filtering preconditioners which we have shown to be effective in improving convergence rates and which possess relatively high degree of parallelism. These are confirmed by the experimental results on the Connection Machine. The multilevel filtering preconditioners are similar to, but developed

independently of, the multilevel nodal basis preconditioners by Bramble, Pasciak and Xu [22]. Using Fourier analysis, we have done a condition number analysis of our method for models problems on uniform grids with certain boundary conditions. For more general second-order self-adjoint problems on non-uniform grids, the convergence results of our method follow the finite element theory from [22]. One advantage of the Fourier analysis in our case is that it helps to finetune our preconditioners to effectively cope with different classes of self-adjoint problems such as anisotropic problems and biharmonic equation. Implementation on the Connection Machine for biharmonic equation shows an improvement in execution time by an order of magnitude. With further optimization, an improvement by two order of magnitude can be achieved.

At this moment, it is not clear whether the multilevel filtering preconditioners are better than the standard multigrid method. While multigrid methods which achieve full multigrid efficiency are hard to beat, we see that for problems with complex operator and complex geometry, our operator-independent approach may be more preferable. A detailed comparison between multigrid methods, our method, and other multilevel preconditioners will be a valuable effort.

There are a few things that we have not done. For general fourth-order self-adjoint elliptic problems, there is a need for a finite element theory for our multilevel filtering preconditioners. In this dissertation, we have made comparison between a few multilevel preconditioners. This effort is by no means exhaustive. In particular, we have not investigated in detail the algebraic multilevel preconditioners developed by Axelsson and Vassilevski [9, 12, 13]. We plan to pursue this preconditioner in the near future.

Bibliography

[1] Adams, L., *Iterative Algorithms for Large Sparse Linear Systems on Parallel Computers,* PhD thesis, University of Virginia, 1982.

[2] Adams, L., *m-step Preconditioned Conjugate Gradient Methods,* SIAM J. Sci. Stat. Comput., Vol. 6, No. 2, April 1985.

[3] Adams, L. M., and Ong, E. G., *A Comparison of Preconditioners for GMRES on Parallel Computers,* in Parallel Computations and Their Impact on Mechanics, ed. A. K. Noor, pp. 171-186, The American Society of Mechanical Engineers, New York, N. Y., 1987.

[4] Agron, E., *Ordering Techniques for the Preconditioned Conjugate Gradient Method on Parallel Computers,* Master Thesis, University of Maryland, 1987.

[5] Ashby, S. F., *Polynomial Preconditioning for Conjugate Gradient Methods,* Department of Computer Science, U. of Illinois at Urbana-champaign, Report No. UIUCDCS-R-87-1355, 1987.

[6] Ashcraft, C. C., and Grimes, R.G., *On Vectorizing Incomplete Factorization and SSOR Preconditioners,* SIAM J. Sci. Stat. Comput., Vol. 9, No. 1, pp. 122-151, 1988.

[7] Axelsson, O., *A Generalized SSOR Method,* BIT, Vol. 13, pp. 443-467, 1972.

[8] Axelsson, O., *On the eigenvalue distribution of a class of preconditioning methods,* Numer. Math. 48, pp. 479-498, 1986.

[9] Axelsson, O., *An algebraic framework for multilevel methods* Report 8820, Department of Mathematics, Catholic University, The Netherlands, 1988.

[10] Axelsson, O. and Barker, V. A., *Finite Element Solution of Boundary Value Problems : Theory and Computation,* Academic Press, 1984.

[11] Axelsson, O. and Lindskog, G., *On the Rate of Convergence of the Preconditioned Conjugate Gradient Method,* Numer. Math. 48, pp. 499-523.

[12] O. Axelsson and P. Vassilevski, *Algebraic multilevel preconditioning methods, I* Report 8811, Department of Mathematics, Catholic University, The Netherlands, 1988.

[13] O. Axelsson and P. Vassilevski, *Algebraic multilevel preconditioning methods, II* Report 1988-15, Institute for Scientific Computation, University of Wyoming, Laramie, Wyoming, 1988.

[14] R. E. Bank and T. F. Dupont, *An Optimal Order Process for Solving Elliptic Finite Element Equations,* Math. Comp. 36, pp. 35-51, 1981.

[15] R. E. Bank, T. F. Dupont and H. Yserentant, *The hierarchical basis multigrid method,* J Numer. Math., 52, pp. 427-458, 1988,, 1988.

[16] C. L. Baucom, *Reduced Systems and the Preconditioned Conjugate Gradient Method on a Multiprocessor,* CSRD Report No. 807, University of Illinois at Urbana-Champaign, Nov. 1988.

[17] H. Berryman, J. Saltz and W. Gropp, *Krylov Methods Preconditioned with Incompletely Factored Matrices on the CM-2,* Department of Computer Science, YALEU/DCS/TR-685, Yale University, March 1989.

[18] G. Birkhoff and Lynch, *Numerical Solutions for Elliptic Problems*, SIAM, Philadelphia, 1984.

[19] D. Braess, *On the Combination of the Multigrid Method and Conjugate Gradients,* Lecture Notes in Mathematics, Multigrid Methods II, edited by W. Hachbusch and U. Trottenberg.

[20] J. H. Bramble, J. E. Pasciak and A. H. Schatz, *An iterative method for elliptic problems on regions partitioned into substructures,* Math. Comp., 46, (1986), pp. 361-369.

[21] J. H. Bramble, J. E. Pasciak and A. H. Schatz, *The construction of preconditioners for elliptic problems by substructuring, I,* Math. Comp., 47,(1986), pp.103-134.

[22] J. H. Bramble, J. E. Pasciak and J. Xu, *Parallel multilevel preconditioners,* To appear in Math. Comp.

[23] W. Briggs and Van Henson, Wavelets and Multigrid, A talk on given at the Copper Mountain Conference on Iterative Methods, 1990.

[24] A. Brandt, *Multi-level adaptive solutions to boundary-value problems,* J Math. Comp. vol. 31, No. 138, pp. 333-390, 1977.

[25] Chan, Tony F., *Domain Decomposition Algorithms and Computational Fluid Dynamics,* CAM Report 88-25, Department of Mathematics, UCLA, 1988.

[26] Chan, Tony F., and Elman, Howard C., *Fourier Analysis of Iterative Methods for Elliptic Problems,* SIAM Review, Vol. 31, NO. 1, pp. 20-49, March 1989.

[27] *Domain Decomposition Methods for Partial Differential Equations,* edited by T. F. Chan, R. Glowinski, J. Periaux, O. B. Widlund, SIAM, Philadelphia, 1989.

[28] T. F. Chan, Jay C.C. Kuo and C. H. Tong, *Parallel elliptic preconditioners: Fourier analysis and performance on the Connection Machine,* Computer Physics Communications 53, pp. 237-252, 1989.

[29] Chandra, R., *Conjugate Gradient Methods for Partial Differential Equations,* Ph. D. Thesis, Computer Science Department, Yale University, 1978.

[30] H. Chen and A. Sameh, *A Domain Decomposition method for 3D Elasticity Problems,* CSRD Report No. 890, Univeristy of Illinois, Sept. 1989.

[31] C. Chu, *Fast Fourier Transforms on Hypercube Computers,* Dissertation.

[32] *Connection Machine Model CM-2 Technical Summary,* by the Thinking Machine Corporation.

[33] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1983

[34] I. Daubechies, *Orthonormal Bases of Compactly Supported Wavelets,* Comm. Pure Appl. Math. 41, 1988, pp.909-996.

[35] Donato, J., *Fourier Analysis of Polynomial Preconditioners for the 5-point Laplacian,* Term Paper, Department of Mathematics, UCLA, 1988.

[36] C. Douglas and W. Miranker, *Constructive interference in parallel algorithms,* SIAM Journal on Numerical Analysis, 25 (1987), pp. 376-398.

[37] M. Dryja, *A capacitance matrix method for Dirichlet problem on polygon region,* Numer. Math., 39, 1982, pp. 51-54.

[38] Duff, I. S., and Meurant, G. A., *The Effect of Ordering on Preconditioned Conjugate Gradients,* September 1988.

[39] Dupont, T., Kendall, R. P., and Rachford, H. H. Jr., *An Approximate Factorization Procedure for Solving Self-adjoint Difference Equations,* SIAM J. Numer. Anal., vol. 5, No. 3, pp. 559-573, 1968.

[40] S. Eisenstat, *Efficient Implementation of a Class of Conjugate Gradient Methods,* SIAM J. Sci. Stat. Comput. 2, 1-4.

[41] H. Elman, *Personal Communication.*

[42] G. C. Fox and S. W. Otto, *Concurrent Computation and the theory of Complex systems,* Technical Report CCCP-255, California Inst. of Technology, Pasadena, CA, March 1986.

[43] D. Fraser, *Array permutation by index-digit permutation,* J. ACM, 22(1976), pp. 298-306.

[44] P. Frederickson, *Totally Parallel Multilevel Algorithms,* RIACS technical report 88-34.

[45] P. Frederickson and O. McBryan, *Parallel superconvergent multigrid,* in Multigrid Methods, S. F. McCormick (editor), Marcel Dekker, New York and Basel, 1988.

[46] D. Gannon and J. V. Rosendale, *On the structure of parallelism in a highly concurrent PDE solver,* Journal of Parallel and Distributed Computing, 3

(1986), pp. 106-135.

[47] A. George and J. Liu, *Computer Solution of Large Sparse Positive Definite Systems,* Prentice-Hall, Englewood Cliffs, NJ.

[48] R. Glowinski, W. Lawton and M. Ravachol, *Wavelet Solution of Linear and Nonlinear Elliptic, Parabolic and Hyperbolic Problems in One space Dimension*, Report of the AWARE, Inc.

[49] Golub, G. H., Van Loan, C. F., *Matrix Computations,* the Johns Hopkin University Press, 1983, chapter 6.

[50] A. Greenbaum, *A Multigrid Method for Multiprocessors,* Proceedings of the Second Copper Mountain Conference on Multigrid Methods, S. McCormick, ed., 1986, pp.75-88.

[51] A. Greenbaum, C. Li and H. Z. Chao, *Parallelizing preconditioned conjugate gradient algorithms,* Computer Physics Communications, vol. 53, 1989, pp. 295-309

[52] I. Gustafsson, *A Class of First Order Factorization Methods,* BIT, v. 18, pp. 144-156, 1978.

[53] W. Hackbusch, *Convergence of Multigrid Iterations aplied to difference equations,* Math. Comp. vol. 34, pp. 325-340, 1980.

[54] W. Hackbusch, *A New Approach to Robust Multi-Grid Methods,* Invited Lecture at ICIAM, Paris, June 29, 1987, and Bericht 8708, Christian-Albrechts-Universitaet Kiel, 1987. Proceedings of the ICIAM'87 Conference, SIAM books, Philadelphia, 1988.

[55] W. Hackbusch, *Multi-Grid Methods and Applications,* Springer-Verlag,Berlin, Germany, 1985.

[56] Hageman, L. A., Young, D. M., *Applied Iterative Methods,* Academic Press, N.Y., 1988.

[57] S. Hammond, *Solving Unstructured Grid Problems on Massively Parallel Computers,* A talk given at the Copper Mountain Conference on Iterative Methods, 1990.

[58] V. E. Henson, W. L. Briggs, *Wavelets: What are they and what do they have to do with Multigrid?* A presentation given at the Copper Mountain Conference on Iterative Methods, April 2-5, 1990.

[59] O. Herrmann, *On the approximation problem in nonrecursive digital filter design,* IEEE Trans. on Circuit Theory, vol. CT-18, 1971, pp. 411-413

[60] P. Hertz, *An Algorithm for the Fast Fourier Transform On the Connection Machine,* accepted by Computers in Physics, June 1989.

[61] Hillis, W. D., *The Connection Machine,* MIT Press, Cambridge, MA, 1985.

[62] Ho, C. T. and Johnsson, S. L., *Optimal Algorithms for Stable Dimension Permutation on Boolean cubes,* In the Third Conference on Hypercube Concurrent Computers and Applications, pp. 725-736.

[63] Hockney, R. W. and Jesshope, C. R., *Parallel Computers : Architectures, Programming and Algorithms,* Adam Hilger Ltd., Bristol, England, 1981.

[64] Hwang, K. and Briggs, F. A., *Computer Architectures and Parallel Processing,* McGraw-Hill Inc., New York, NY, 1984.

[65] Johnson, O., Micchelli, C. A., Paul, G., *Polynomial Preconditioners for Conjugate Gradient Calculations,* SIAM J. Numer. Anal., Vol. 20, No. 2, April 1983, pp. 362-376.

[66] S. L. Johnsson, *Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures,* J. Parallel Distributed Comput., 4(2):133-172, April 1987.

[67] S. L. Johnsson and C. T. Ho, *Matrix Transposition on Boolean n-cube Configured Ensemble Architectures,* SIAM J. Matrix Anal. Appl., Vol. 9, No. 3, pp. 419-454, July 1988.

[68] S. L. Johnsson and C. T. Ho, *Spanning Graphs for Optimum broadcasting and personalized Communication in Hypercubes,* IEEE Trans. Computers, Vol. 38, No. 9, pp. 1249-1268, September 1989.

[69] S. L. Johnsson, C. T. Ho, M. Jacquemin and A. Ruttenberg, *Computing Fast Fourier Transforms on Boolean Cubes and related networks,* In Advanced Algorithms and Architectures for Signal Processing II, Vol. 826, pp. 223-231, 1987.

[70] L. Johnsson, Rl L. Krawitz, D. MacDonald, and Roger Frye, *A radix-2 FFT on the Connection Machine,* In Supercomputing 89, pp. 809-819, ACM, Nov. 1989.

[71] Jordan, T. L., *Conjugate Gradient Preconditioners for Vector and Parallel Processors,* Elliptic Problem Solvers II, 1983, pp.127-140.

[72] R.A. Kamin III, and G.B. Adams III, *Fast Fourier Transform Algorithm Design and Tradeoffs on the CM,* Proceedings of the Conference on Scientific

Applications of the Connection Machine, Editor : H. Simon, World Scientific Publishing Co., 1989.

[73] R. Kettler, *Analysis and comparison of relaxation schemes in robust multigrid and preconditioned conjugate gradient methods,* Multigrid Methods, Hackbusch and U. Trottenberg, Springer-Verlag, New York, N.Y. 1982, pp. 502-534

[74] R. Kettler and J. A. Meijerink, *A multigrid method and a combined multigrid-conjugate gradient method for elliptic problems with strongly discontinuous coefficients in general domain,* Shell publication 604, KSEPL, Rijswijk, The Netherlands

[75] Kuo, C.-C. Jay, and Chan, Tony F., *Two-color Fourier Analysis of Iterative Algorithms for Elliptic Problems with Red/Black Ordering,* CAM Report 88-15, Department of Mathematics, UCLA, 1988.

[76] C.-C. J. Kuo, T. F. Chan and Charles Tong, *Multilevel Filtering Elliptic Preconditioners,* SIAM J. Matrix Analysis and Applications, Vol. 11, No. 3, July 1990.

[77] Y. A. Kuznetsov, *Multigrid domain decomposition methods for elliptic problems,* Proceedings VIII International Conference on Computational Methods for Applied Science and Eng. Vol. 2, 1987, pp. 605-616

[78] Creon Levit, *Grid Communication on the Connection Machine : Analysis, performance, and improvements,* Proceeding on the Scientific Application of the Connection Machine, Editor : Horst Simon, World Scientific Publishing Co., 1989.

[79] R. E. Lynch and J. R. Rice, *A high-order difference method for differential equations,* Math. Comp., 34 (1980), pp. 333-372.

[80] McBryan, O. A., *State-of-the-art in Highly Parallel Computer Systems,* in Parallel Computations and Their Impact Mechanics, ed. A. K. Noor, pp. 31-46, The American Society of Mechanical Engineers, New York, 1987.

[81] McBryan, O. A., *The Connection Machine : PDE solution on 65536 Processors,* Parallel Computing, to appear.

[82] O.A. McBryan, *Connection Machine Application Performance,* CU-CS-434-89, Department of Computer Science, University of Colorado, April 1989.

[83] O.A. McBryan and E. van de Velde, *Parallel Algorithms for Elliptic Equations,* Commun. Pure Appl. Math. 38, pp. 769-795.

[84] S. McCormick and J. Thomas, *The Fast Adaptive Composite Grid (FAC) Method for of Elliptic Equations,* Mathematics of Computation, Vol. 46, No. 174, April 1986, pp. 439-456

[85] S. McCormick, *Multilevel Adaptive Methods for Partial Differential Equations.*

[86] Meijerink, J. A., and Van der Vorst, H. A., *An iterative Solution Method for Linear Systems of which the Coefficient matrix is a symmetric M-Matrix,* Math. Comp., Vol. 31, no. 137, pp. 148-162, 1977.

[87] M. E. G. Ong, *The 3D linear hierarchical basis preconditioner and its shared memory parallel implementation,* Preprint, Department of Applied Mathematics, University of Washington, Seattle, WA 98195

[88] A.V. Oppenheim, and R.W. Schafer, *Digital Signal Processing,* Prentice Hall, 1975.

[89] Ortega, J. M., *Introduction to Parallel and Vector Solution of Linear Systems,* Frontier of Computer Science, Plenum Press, New York, 1988, Section 3.4.

[90] Ortega, J. M., and Voigt, R. G., *Solution of Partial Differential Equations on Vector and Parallel Computers,* SIAM Review, Vol. 27, No. 2, pp. 149-240, June 1985.

[91] Poole, E. and Ortega, J. M., *Multicolor ICCG Methods for Vector Computers,* SIAM J. Numer. Anal. 24, pp. 1394-1418.

[92] Y. Saad, *Practical Use of Polynomial Preconditionings for the Conjugate Gradient Method,* SIAM J. Sci. Stat. Comput., Vol. 6, No. 4, Oct. 1985.

[93] M. Schultz and Y. Saad, *Parallel Implementations of Preconditioned Conjugate Gradient Methods,* Department of Computer Science, YALEU/DCS/TR-425, Yale University, October, 1985.

[94] B. F. Smith, *An Optimal Domain Decomposition Preconditioner for the Finite Element Solution of Linear Elasticity Problems,* Preprint.

[95] B. Smith and O. Widlund, *A domain Decomposition Algorithm based on a change to a hierarchical basis,* submitted to SIAM J. Sci. Stat. Comput.

[96] G. Strang and G. J. Fix, *Wavelets and Dilation Equations : A brief Introduction,* SIAM Review, December 1989, pp. 614-627.

[97] G. Strang and G. J. Fix, *An Analysis of the Finite Element Method,* Prentice-Hall series in automatic computation, 1973,

[98] K. Stuben and U. Trottenberg, *On the construction of fast solvers for elliptic equations,* Computational Fluid Dynamics, Rhode-Saint-Genese, 1982.

[99] P.N. Swarztrauber, *FFT algorithms for vector computers,* Parallel Computing, 1 (1984), pp. 45-63.

[100] P.N. Swarztrauber, *Multiprocessor FFTs,* Parallel Computing, 5 (1987), pp. 197-210.

[101] P.N. Swarztrauber, *The FFT a s a Multigrid Algorithm,* Draft, January 5, 1989.

[102] C. H. Tong, *Preconditioned Conjugate Method on the Connection machine,* International Journal of High Speed Computing, 2nd issue 1989.

[103] C. H. Tong, C. and P. Swarztrauber, *Ordered Fast Fourier Transform on a Massively Parallel Hypercube Multiprocessor,* Accepted by Journal of Parallel and Distributed Computing.

[104] R. S. Tuminaro, *Multigrid Algorithms on Parallel Processing Systems,* PhD Thesis, Department of Computer Science, Stanford University, Dec. 1989.

[105] R. S. Varga, *Matrix Iterative Analysis,* Prentice-Hall, Englewood Cliffs, NJ.

[106] P. Vassilevski, *Iterative methods for solving finite element equations based on multilevel splitting of the matrix,* Preprint, Bulgarian Academy of Science, Sofia, Bulgaria, 1987

[107] J. Xu, *Theory of multilevel methods,* Ph.D. Thesis, Department of Mathematics, Cornell University, N.Y. 14853, 1989

[108] J. Xu, *Iterative Methods by Space Decomposition,* A talk given at the Copper Mountain Conference on Iterative Methods, 1990.

[109] J. Xu and J. Qin, *On Some multilevel preconditioners,* Submitted to SIAM J. on Sci.and Stat. Comput.

[110] D. M. Young, *Iterative Solution of Large Linear Systems,* Academic Press, New York, 1971.

[111] Yserentant, H., *On the Multilevel Splitting of Finite Element Spaces,* Numer. Math., Vol. 49. 1986, pp. 349-412.