Parallel Algorithms for 2-D Boundary Value Systems

by

Michael M. Daniel

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 1993

Author .....
Department of Electrical Engineering and Computer Science
December 18, 1992

Certified by .....
Alan S. Willsky
Professor
Thesis Supervisor

Accepted by .....
Campbell L. Searle
Chairman, Departmental Committee on Graduate Students

Parallel Algorithms for 2-D Boundary Value Systems

by

Michael M. Daniel

Submitted to the Department of Electrical Engineering and Computer Science
on December 18, 1992, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

The objective of this thesis is to develop and investigate efficient, direct, and parallelizable algorithms for 2-D boundary value systems, particularly in the context of 2-D digital filtering. FIR filters appear to be the preferred form of implementation in 2-D signal and image processing applications not only because they have desirable attributes like stability and efficient solutions, but also because the properties of 2-D IIR implementations are not well known. In this thesis we provide a framework for efficiently implementing 2-D IIR filters specified by a 2-D linear difference equation constrained by boundary conditions. A direct solution is developed whose performance is comparable to the nested dissection algorithm, and whose approximate solution is is more computationally efficient than nested dissection for filter systems identical to strictly diagonally dominant systems of equations.

Thesis Supervisor: Alan S. Willsky
Title: Professor

Acknowledgments

If you need a change, just stand here, by God, and you'll get it.
So turn once again, back up
the driveway, thinking, as lately you've come to,
that you can still make a wrong step,
on rough ground, in the dark, and not
quite cripple yourself. Porchlights, left on to show you the way
home, shine right in your eyes, and you
can't see a thing, but you'll get there.

—Henry Taylor, from *The Flying Change*

I first would like to thank Alan for providing me with inspiration, support, serious baseball dialogue, and loads of patience. The work in this thesis is, if anything, my manifestation of his ideas.

I would also like to thank Prof. Jacob White for *always* taking the time to answer my questions, and for being an incredibly nice guy.

This work would have been a dreary task without the support of my fellow LIDS members. In particular, thanks to Mark and Eric for office dialogue and direction. And for entertainment, I thank the nerdy duo at 25 Lime St.

Finally, I give thanks to my parents for their love, support, and endless supply of plane tickets, and thanks to my Porchlight, S.

This research was benevolently supported by:

Office of Naval Research, Grant No. N00014-91-J-1004

Air Force Office of Scientific Research, Grant No. F49620-92-J-0257DEF

National Science Foundation, Grant No. MIP-9015281

Contents

List of Figures

Chapter 1

Background and Motivation

This thesis aims to develop efficient, direct (non-iterative), and parallelizable algorithms for acausal two-dimensional IIR filters described by partial difference equations constrained by boundary conditions. There appear to be a wide range of applications, from computer vision problems like edge detection [18] to oceanographic signal processing [36] to the familiar discrete Poisson equation [5, 9, 17, 16], in which one is given a difference equation and a set of boundary conditions, and must then efficiently compute the output for any given input. While such acausal systems have numerous elegant solutions in 1-D, the problem can become extremely complex and computationally demanding for two or more dimensions.

In one dimension, both FIR and IIR filtering systems have efficient solutions, and their relative desirability depends upon the applications. In 2-D, however, FIR filters are almost exclusively preferred to their IIR counterparts. Although there exist important differences, 2-D FIR filters are nearly identical to their 1-D counterparts in that they are always stable, have straightforward implementations, can be implemented efficiently using the 2-D DFT, and have an output area of support determined exclusively by the area of the input and the order of the filter. Many "discrete-time" systems and discrete approximations to continuous-time systems, however, are most naturally represented as an IIR filter specified implicitly by a 2-D linear difference equation [4, 10, 22, 20, 24, 25, 39]. One common example is a partial differential equation (P.D.E.), such as Poisson's equation [5, 9, 17], approximated by finite difference methods. Unfortunately, the solution to such a 2-D IIR system does not readily follow from 1-D filtering theory, and the computational complexity in obtaining the solution increases dramatically with the size of the 2-D region of support. To further

complicate analysis, the stability of such filters is often difficult to characterize.

Is is possible to overcome these barriers? A major premise of this thesis is that the extension of 1-D IIR filtering theory developed within a uniprocessor environment to 2-D has stifled the development of more natural frameworks for analyzing and solving 2-D IIR filtering systems. Because a 2-D partial difference equation constrained by boundary conditions (B.C.'s) is just a sparse system of linear equations, a starting point for building a new framework would be to apply, adapt, and extend some of the powerful sparse system techniques from numerical linear algebra to 2-D IIR filtering. In particular, direct algorithms like nested dissection and LU factorizations resulting from minimum degree orderings are known to be the most efficient direct solutions to partial difference equations [14, 13].

1.1 Boundary Conditions and Acausal Systems

With any system of linear equations $Ax = b$, A must have full rank in order that the problem be *well-posed* (x has a unique solution). Any linear filter in the form of a difference equation can be cast as a system of linear equations, but the system is not well-posed without a set of auxiliary conditions. For instance, a 1-D difference equation of the form

$$x[n] = ax[n-1] + bu[n] \tag{1.1}$$

does not alone define a system, since there is not a unique output sequence $x[n]$ for any given input $u[n]$ unless auxiliary conditions are specified. Usually for a 1-D filter, the auxiliary conditions are initial conditions, and the independent variable (index n) is time. For the causal[1] first-order filter given in (1.1), an initial condition such as $x[0] = c$ allows one to uniquely determine the output $x[n]$ for $n > 0$ given $u[n]$ for $n > 0$ by recursively propagating the initial condition forward in time. For acausal 1-D filters, however, initial conditions are not an appropriate choice of auxiliary conditions, since the output response $x[n]$ is usually non-zero for infinite extent in both directions ($n \in [-\infty, \infty]$), which is often the case when n no longer represents time.

For example, suppose that one is given a stable linear shift-invariant (LSI) discrete-time

[1] In 1-D, a *causal* filter is one which has an impulse response which is non-zero only for $n \geq 0$. An *anti-causal* filter has an impulse response which is non-zero only for $n \leq 0$. An *acausal* filter has an impulse response which can be non-zero for infinite extent in either direction of n.

system specified by its system function, $H(z)$:

$$\begin{aligned} H(z) &= \frac{1}{1-\alpha z^{-1}} + \frac{\alpha z}{1-\alpha z} \\ &= H_1(z) + H_2(z) \end{aligned} \tag{1.2}$$

$$X(z) = H(z)U(z) \tag{1.3}$$

where $X(z)$ and $U(z)$ are the $\mathcal{Z}$-transforms of the system output and input, respectively. Any 1-D IIR filter's system function can be decomposed into a parallel combination of a causal and an anti-causal IIR filter. Assuming $|\alpha| < 1$ and filter stability, the causal filter takes the form

$$H_1(z) = \frac{1}{1-\alpha z^{-1}} \tag{1.4}$$

and the anti-causal filter is then

$$H_2(z) = \frac{\alpha z}{1-\alpha z} \tag{1.5}$$

Taking the inverse transform of (1.2) gives the system in the form of the difference equation

$$-\alpha x[n+1] + (1+\alpha^2)x[n] - \alpha x[n-1] = (1-\alpha^2)u[n] \tag{1.6}$$

For many image processing, and non-real-time 1-D signal processing and estimation problems, one is only concerned with the outputs over a specific window. Consider the case in which one would like to solve for $x[n]$ over the interval $N_1 \leq n \leq N_2$ given $u[n]$ over this same interval $[N_1, N_2]$. The auxiliary conditions required to accomplish this then come in the form of boundary conditions, where $x[n]$ is constrained at the boundaries of the region of support $[N_1, N_2]$. The boundary conditions constrain two degrees of freedom in the second-order difference equation given in (1.6), and come in two general forms, *separable* and *non-separable*. Separable B.C.'s constrain the ends of the interval independently, such as

$$x[N_1] = c_1 \tag{1.7}$$

$$x[N_2] = c_2 \tag{1.8}$$

and non-separable constrain $x[n]$ at each of the boundaries simultaneously, such as

$$C \begin{bmatrix} x[N_1] \\ x[N_2] \end{bmatrix} = c \tag{1.9}$$

where C is a 2×2 non-singular matrix. The boundary conditions specify a system which has a completely different flavor from the causal system given by (1.1) and its initial condition. The solution to the system defined by (1.1) is found by recursively propagating the initial condition for $n > 0$. Likewise, if the system corresponding to (1.2) were instead considered to be a causal (but necessarily unstable) system, the auxiliary conditions would be initial conditions, such as $x[-1] = c_{-1}$ and $x[-2] = c_{-2}$, which could be recursively propagated by casting (1.6) as the computational procedure in (1.10).

$$x[n] = \frac{1+\alpha^2}{\alpha} x[n-1] - x[n-2] - \frac{1-\alpha^2}{\alpha} u[n] \tag{1.10}$$

Such systems are called *recursively computable* [10], because each step of the solution involves recursively computing a value of the output, $x[n]$, using previously computed outputs (in Equation (1.10), $x[n-1]$ and $x[n-2]$).

One should also note that for acausal systems the imposition of stability requires a particular form of boundary conditions. Stability allows one to separate a filter into its causal and anti-causal components, as was done in Equations (1.4) and (1.5). This decomposition is discussed for general acausal systems in [34]. Under stability, the boundary conditions must be specified such that the causal filter satisfies initial rest conditions and the anti-causal filter satisfies final rest conditions.

The point of this section is to emphasize that a system is a combination of a difference equation and a set of auxiliary conditions. A 1-D system's properties depend upon both its difference equation and its auxiliary conditions. In particular pole placement is determined by the difference equation. Whether a system is causal or acausal depends upon the form of the auxiliary conditions. Initial conditions were shown to yield causal systems, while boundary conditions yield an acausal system. The same is true for 2-D systems, except for the fact that poles are more difficult to identify, and there is more flexibility in specifying the auxiliary conditions.

1.2 Algorithms for Acausal IIR filters

Aside from determining system properties, the form of a system's difference equation and boundary conditions determine the form of the solution. 1-D and 2-D IIR filters are implemented with either *direct* or *iterative* solution algorithms. Direct implementations produce a system's exact solution, excluding errors due to finite-precision- arithmetic, in a finite number of steps [14]. Iterative implementations converge towards the solution with each step, yet ideally take an infinite number of steps to obtain the exact solution. For one and two-dimensional IIR filters constrained by initial conditions, the solution is generally obtained with a direct form algorithm in which a new output value is obtained at each step of the algorithm. The only "memory" required for such algorithms is a finite set of previously obtained output values. As was stated in the preceding section, these systems are referred to as being recursively computable. IIR filtering systems which are constrained by boundary conditions are not recursively computable. In 2-D these acausal systems have traditionally been solved with iterative algorithms, yet efficient direct algorithms do exist. Direct algorithms which are suitable implementations for acausal IIR filters have for the most part been confined to the field of numerical linear algebra, such as [14, 16], and have been neglected in the 2-D signal processing literature [10, 26, 25].

For 1-D acausal IIR filters, there exist three general implementations (2 direct, 1 iterative) whose relative utility depends upon the nature of the application. The first approach relies upon the Fundamental Theorem of Algebra, which allows any univariate polynomial to be factored into a product of first-order polynomials. A convenient property of 1-D LSI IIR filters specified by difference equations is that their system functions are always rational functions of z,

$$H(z) = \frac{b(z)}{a(z)}$$

meaning that $a(z)$ and $b(z)$ can be expressed as products of first-order polynomials. The poles and zeros of a system can be readily determined from such a factorization, which assuming stability allows $H(z)$ to be expressed as a sum of one causal and one anti-causal filter. The system function given in (1.2) was expressed as a parallel combination of the causal filter (1.4) and the anti-causal filter (1.5). The acausal IIR filter can then be implemented by running the casual filter forward in n, using the following difference equation

derived from $H_1(z)$

$$x_1[n] = \alpha x_1[n-1] + u[n]$$

and running the anti-causal filter backwards in n, using the following difference equation derived from $H_2(z)$

$$x_2[n] = \alpha x_2[n+1] + \alpha u[n+1]$$

The causal filter must be at initial rest ($x_1[n] = 0$ for $n < N_1$) and the anti-causal filter must be at final rest ($x_2[n] = 0$ for $n > N_2$). These two auxiliary conditions are the boundary conditions of the acausal filter. The acausal filter output $x[n]$ is just the sum of $x_1[n]$ and $x_2[n]$ (see Figure 1-1 for an illustration). Assuming stability, one can thus always compute the outputs of an acausal 1-D IIR filter system by breaking it into its causal and anti-causal components, initializing each filter with a subset of the B.C.'s, and then summing the outputs of the sub-filters to produce $x[n]$. In the context of optimal estimation, such a filter is known as a Mayne-Fraser two-filter algorithm ([38, 37]).

The second approach to solving acausal 1-D IIR systems involves recursively propagating the boundary conditions at one end of the region of support through the filter dynamics to the other end, then recursively solving for the system outputs with a filter running in the opposite direction. For instance, consider an acausal filter similar to that given by the difference equation (1.6), rewritten as

$$a_1 x[n+1] + a_0 x[n] + a_{-1} x[n-1] = u[n] \tag{1.11}$$

with the separable boundary conditions

$$c_0 x[N_1] + c_1 x[N_1+1] = c_2 \tag{1.12}$$

$$d_{-1} x[N_2-1] + d_0 x[N_2] = d_2 \tag{1.13}$$

To obtain the filter outputs over $n \in [N_1, N_2]$ given $u[n]$ for $n \in [N_1+1, N_2-1]$, one can start by combining (1.12) and the difference equation (1.11) evaluated at $n = (N_1+1)$ to eliminate $x[N_1]$. This combination yields a new boundary condition of the form

$$l_{N_1 1} x[N_1+1] + l_{N_1 2} x[N_1+2] = b_{N_1} \tag{1.14}$$

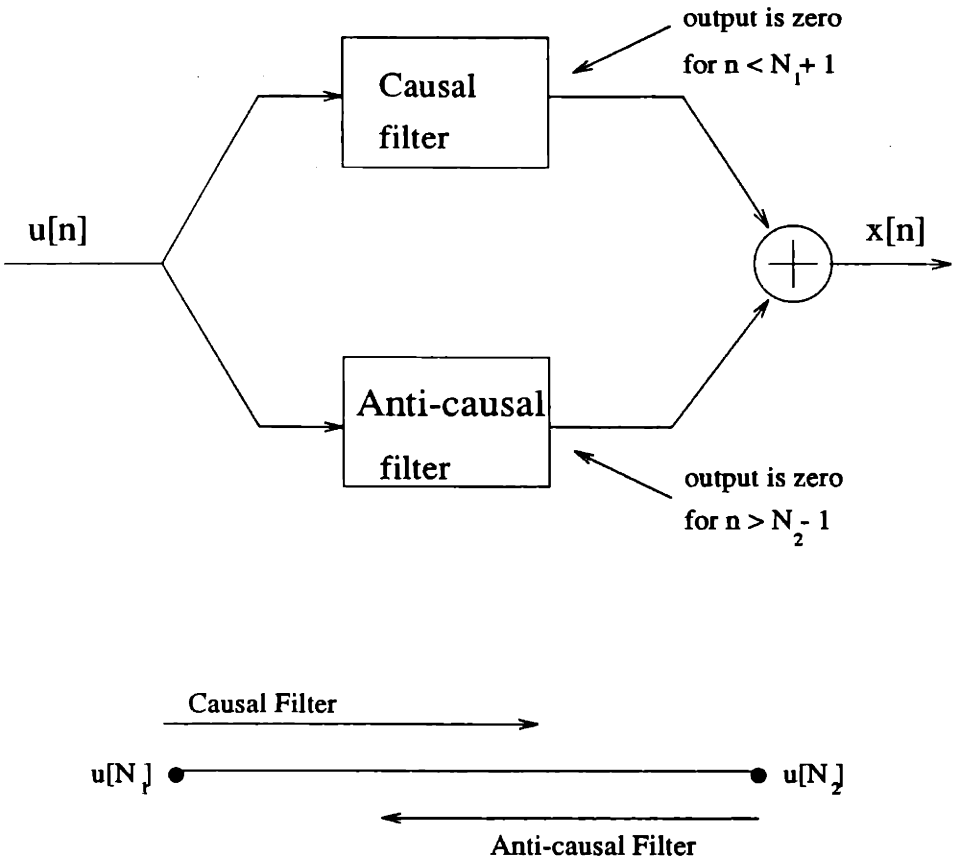

Figure 1-1: Parallel implementation for a general 1-D IIR Filter

The constraint (1.14) is then substituted into the system dynamics (1.11) at $n = (N_1+2)$, to yield yet another boundary condition which is one step closer to N_2. The process is repeated recursively until the other side of the region of support is reached, yielding a constraint of the form

$$l_{N_2 1} x[N_2 - 1] + l_{N_2 2} x[N_2] = b_{N_2} \tag{1.15}$$

Solved simultaneously with Equation (1.13), Equation (1.15) can be used to obtain the filter output at $n = N_2$ and $n = (N_2 - 1)$. The propagated constraints, like equations (1.14)-(1.15), are then used to solve recursively for $x[N_2 - 2], x[N_2 - 3]$, ..., $x[N_1 + 1]$, whereupon $x[N_1 + 1]$ is substituted into (1.12) to yield the final output, $x[N_1]$.

One can visualize this algorithm as a filter running forwards in n, propagating the B.C. (1.12) at $n = N_1$ to $n = N_2$, followed by a filter running backwards in n, which back-substitutes the solution found at $n = N_2$ into the constraints obtained by the forward filter. Such a filter is commonly referred to in the context of optimal estimation as a Rauch-Tung-Striebel (RTS) filter ([37, 38]). Note the difference from a Mayne-Fraser (MF) filter, which has both a forward and backward filter running in parallel, whereas the RTS filter consists of a forward sweep of the dynamics *followed* by a backward filter.

Although both MF and RTS algorithms have linear-algebraic interpretations, only that of the RTS will be discussed herein. Casting (1.11)-(1.13) as a system of linear equation yields

$$Ax = b \tag{1.16}$$

$$\begin{bmatrix} c_0 & c_1 & 0 & 0 & \cdots & 0 \\ a_{-1} & a_0 & a_1 & 0 & \cdots & 0 \\ 0 & a_{-1} & a_0 & a_1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{-1} & a_0 & a_1 \\ 0 & \cdots & 0 & 0 & d_{-1} & d_0 \end{bmatrix} \begin{bmatrix} x[N_1] \\ x[N_1+1] \\ x[N_1+2] \\ \vdots \\ x[N_2-1] \\ x[N_2] \end{bmatrix} = \begin{bmatrix} c_2 \\ u[N_1+1] \\ u[N_1+2] \\ \vdots \\ u[N_2-1] \\ d_2 \end{bmatrix}$$

Obtaining the solution is now seen as inverting A to obtain x. For $(N_2 - N_1)$ large, one can save considerable computational effort and storage by not accessing the many zero elements in A. Many sparse matrices with structure, like the tri-diagonal A, can be inverted without ever storing, manipulating, or writing over the zero-entries. Conveniently, the LU

factorization of A,

$$A = LU$$

is such that L is bidiagonal and lower-triangular, and U is bidiagonal and upper-triangular. No fill-in has resulted from the factorization. Multiplying both sides of (1.16) on the left by L^{-1} to obtain $Ux = y$ (where $Ly = b$) is in fact identical to the forward filter of the RTS algorithm. Solving first for $x[N_2]$, the backward filter of the RTS algorithm is the same as just back-substituting from the "bottom-up" in $Ux = y$ to obtain x. The RTS algorithm is an alias for Gaussian elimination followed by back-substitution.

The MF and RTS algorithms are direct algorithms. The third solution algorithm is iteration, and it is the preferred solution method for a large number of problems in which it produces an acceptable solution extremely fast. Gauss-Seidel, SOR (Successive Over-relaxation), Multi-grid, and Preconditioned Conjugate Gradient methods are common iterative algorithms for solving linear systems of the form $Ax = b$ [16], a form in which any linear filter can be cast. Iterative algorithms are often preferred for their simplicity. Unlike many direct schemes, they are easy to implement. They also require minimal amounts of storage (approximately twice the amount necessary to store the input), as well as producing a solution of acceptable accuracy faster than or equal to direct algorithms for a wide range of applications. These algorithms, however, often suffer convergence problems, and fail to produce solutions for a large class of filters [14, 16].

Unfortunately, the story is completely different when switching from one dimension to two. Acausal 2-D IIR filtering systems are often considered intractable, especially for large boundary sizes. A solution for 2-D filters analogous to that which relies upon splitting a 1-D acausal filter into its causal and anti-causal parts does not exist. The method of breaking a filter into its causal an acausal components relies solely upon the ability to factor the denominator of H(z) into a product of first-order univariate polynomials. A 2-D system function has the form

$$H(z_1, z_2) = \frac{b(z_1, z_2)}{a(z_1, z_2)}$$

where $a(z_1, z_2)$ and $b(z_1, z_2)$ are both polynomials in two variables. Unlike the 1-D case, the polyvariate polynomials $a(z_1, z_2)$ and $b(z_1, z_2)$ do not in general factor into a canonical form of products of lower order terms. This implies that the polynomials do not in general

separate into a product of univariate polynomials, such as

$$a(z_1, z_2) = c(z_1)d(z_2)$$

A filter which can be separated in this form is called separable, but is rarely encountered for systems naturally described by 2-D difference equations [7, 10]. Separable filters usually result as an artifice of filter design techniques. The lack of a theorem for polyvariate polynomials analogous to the Fundamental Theorem of Algebra rules out the possibility of developing a direct algorithm for 2-D acausal IIR filters similar to the MF, and it also accounts for difficulties in testing 2-D systems for stability. Most useful stability tests for 2-D DT filters only apply to recursively computable (causal) systems [25], and even then

> "The complexity of testing a 2-D system's stability explains, in part, why 2-D FIR digital filters, which are always stable, are much preferred over 2-D IIR digital filters in practice. The preference for FIR filters over IIR filters is much more marked in 2-D than in 1-D signal processing applications [25, p. 124]."

The difficulty in testing for stability can be explained in part by observation that recursive computation can be done in an infinite number of directions for 2-D causal systems. Recursively computable systems in 1-D had only two directions of recursion, causal or anti-causal. For acausal 2-D systems, stability is an unclear concept.

The third general solution mentioned for 1-D acausal IIR filters is iteration. Iteration is in fact the preferred form of implementation for many 2-D difference equations constrained by boundary conditions [40], and maintains the desirable properties characteristic of iterative 1-D algorithms (ease of implementation, minimal storage, and fast convergence for a number of applications). The question which now must be asked, however, is whether iterative or direct methods are better, in terms of storage, computational complexity, and accuracy. As is noted in the first chapter of [14], the answer depends both upon the coefficients of the 2-D IIR filter (namely, the characteristics of A when the filter is cast in the form (1.16)), and the suitability of the direct algorithm. Iterative algorithms almost always require less storage than direct algorithms, yet how much less depends heavily upon the ordering of the difference equation coefficients in A [14]. In terms of computational complexity, the comparison again depends upon the characteristics of A, for iterative methods have convergence problems for a large class of systems, whereas direct algorithms take

a predetermined, finite number of steps for any given system. The real payoff for direct methods comes

> "in some situations, such as in the design of mechanical devices... many systems of equations having the same coefficient matrix must be solved. In this case, the cost of the direct scheme may be essentially that of solving the triangular system *given the factorization*, since the factorization cost amortized over all solutions may be negligible... In addition, there are situations where it can be shown quite convincingly that direct methods are far more desirable than any conceivable iterative scheme. [14]"

This sentiment is echoed in [19]. Iterative methods can and do provide efficient solutions to many of the 2-D acausal IIR filtering systems considered in this thesis [40], but it is believed that direct methods will provide a novel analytical framework from which to solve such systems.

It has been noted a number of times that any discrete-time linear filtering system, of any order or dimension, can be represented as a (sparse) system of linear equations. Because the RTS algorithm is just a specific form of Gaussian elimination, one should expect to be able to extend the algorithm from one to two dimensional systems. Two-dimensional IIR filters, however, are much more complex than their 1-D brothers. When a 1-D filter is cast in the form of (1.16), where the variables in x are ordered sequentially in n, the bandwidth of the matrix A is strictly bounded by the order of the filter. This tight bound guarantees, without any reordering of the variables and equations in $Ax = b$, an LU factorization in which the sum of the number of non-zero elements in L and U is equal to the number of non-zero elements in A (assuming none of the filter coefficients are zero) [14]. One thus has a direct algorithm of minimal storage and complexity by simply casting the 1-D acausal filter as a banded system of equations. In two dimensions, however, the storage required to represent A and the amount of fill-in which occurs in its factorization depends heavily upon the ordering of the equations in A and the variables in x. Furthermore, for a region of support of $N \times N$ points, the coefficient matrix A contains N^4 elements (including zeros), meaning that intelligent orderings in the equation $Ax = b$ are *required* to allow for efficient storage and factorization schemes for regions of support of even modest size.

This thesis begins by first extending the ideas in [37] to develop an RTS algorithm for 2-D acausal IIR filtering systems, and then evaluates its complexity. A parallel implementation

of the algorithm is then given, which happens to be extremely similar to the nested dissection algorithm. Because in digital signal processing and estimation applications accuracy is often sacrificed when designing a system, such as a low-pass filter with a transition band, accuracy might also be sacrificed for efficiency in the solution algorithm. Algorithms are thus developed which are an approximation to the 2-D RTS algorithm, where accuracy in the solution is traded for both efficiency and ease of implementation. Finally, the thesis concludes with a comparison of some known direct and iterative algorithms, as well as giving some numerical results on a few example filters.

Chapter 2

Direct Algorithms for 2-D Acausal Filters

Whenever solving a large, sparse system of equations of the form $Ax = b$, the objective is usually to produce a solution for x with minimum error using as little computational and storage resources as possible. This goal is usually accomplished in direct implementations by reordering the variables in x such that the LU factorization has minimum fill-in [14, 15]; the savings can be extraordinary. This chapter takes a somewhat different approach, desiring not only efficient algorithms, but also an intuitive framework from which to analyze 2-D acausal IIR filters; therefore, it is a given that the initial developments will discuss algorithms which are sub-optimal in terms of both computational complexity and storage demands. From this framework, however, it is believed that more powerful algorithms can be developed for signal processing and estimation applications.

The RTS algorithm in Chapter 1 provided an intuitive model for acausal 1-D filters by propagating boundary conditions through the dynamics. In this chapter, we begin by showing how to cast any 2-D filter as a 1-D dynamic system of growing state dimension, which can then be solved by algorithms similar to the RTS. A parallel version of the algorithms is then given.

2.1 Casting 2-D Filters into 1-D

The difference equation for 2-D IIR filter is usually given as a 2-D LSI system in the form

$$\sum_{k=-M}^{M}\sum_{l=-N}^{N} a_{kl}x[i-k,j-l] \;=\; \sum_{k=-P}^{P}\sum_{l=-Q}^{Q} b_{kl}u[i-k,j-l] \tag{2.1}$$

where *u[i,j]* is the filter input and *x[i,j]* the filter output, which is also referred to as a mesh variable [14]. The order of (2.1) will be defined in this thesis as $\max\{2M, 2N\}$, for reasons which will become clear later. The difference equation alone, of course, does not define a system, as auxiliary conditions must also be specified.

A simple second-order 2-D IIR filter is the Nearest Neighbor Model (NNM) shown below in (2.2).

$$x[i,j] = Nx[i,j+1] + Sx[i,j-1] + Ex[i+1,j] + Wx[i-1,j] + Bu[i,j] \tag{2.2}$$

where the coefficients N, S, E, and W denote dependencies of $x[i,j]$ on neighbors to the north, south, east, and west, respectively. One should note that the input u and the output x can have arbitrary dimension in (2.2), yet most of the derivations in this thesis are for the scalar case. The NNM has been shown to model a large number of physical phenomena, and is a useful paradigm for analyzing signal processing and estimation applications, such as Markov Random Fields [4, 6, 8, 24, 41]. The NNM is also a natural by-product of approximating PDE's by finite-difference methods [24]. For these applications, the auxiliary conditions are generally in the form of boundary conditions.

One has a considerable amount of flexibility in specifying B.C.'s for 2-D acausal systems. In this thesis, the assumption is made that the B.C.'s are given. Two common forms are discrete versions of *Dirichlet* and *Neumann* conditions. If one is given the difference equation (2.2) and the inputs $u[i,j]$ over $(i,j) \in [2, I-1] \times [2, J-1]$, and would like to solve for the output x over $(i,j) \in [1,I] \times [1,J]$, then Dirichlet conditions consist of initializing the filter outputs which encircle this *region of support* (ROS). Namely, $x[i,j]$ would be initialized for all

$$(i,j) \in \{(i \in \{1,I\} \cap\ j = [2, J-1]) \cup (j \in \{1,J\} \cap\ i = [2, I-1])\}$$

For a rectangular sampling of 2-D space, Dirichlet conditions are illustrated in Figure 2-1

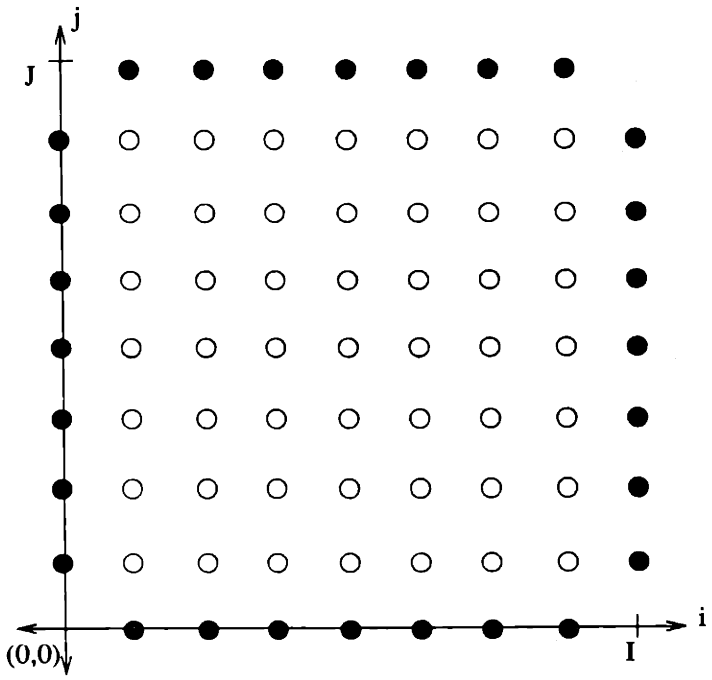

Figure 2-1: Rectangular mesh on which Dirichlet conditions are imposed.

as those points in the mesh which are shaded, while the output must be solved for those points which are not shaded. Note that, as with any choice of B.C.'s, the extent of the Dirichlet conditions grows proportionally to the size of the ROS. If the difference equation were of higher order, the Dirichlet condition would grow in *thickness* by initializing more output variables at the edge of the ROS. Furthermore, note that the shape of the Dirichlet conditions depends not only upon the order of the filter, but also the shape of the boundary of the ROS, yet they always consist of auxiliary constraints which initialize the filter outputs at the edge of the ROS. A hexagonally sampled mesh might also yield a difference equation in the form of (2.2), and Dirichlet conditions will then have the shape of a diamond. The initialized points of the diamond are illustrated with dark circles in Figure 2-2.

A second common form of B.C.'s, Neumann conditions, are just a linear constraint on the output values which lie on both the exterior and interior edges of the ROS, reflecting a discretization of the gradient normal to the boundary of the ROS. For an NNM defined on a rectangular mesh with the input again given for those (i, j) where $(i, j) \in [2, I-1] \times [2, J-1]$, Neumann conditions are just a linear constraint on the output values shown by shaded circles

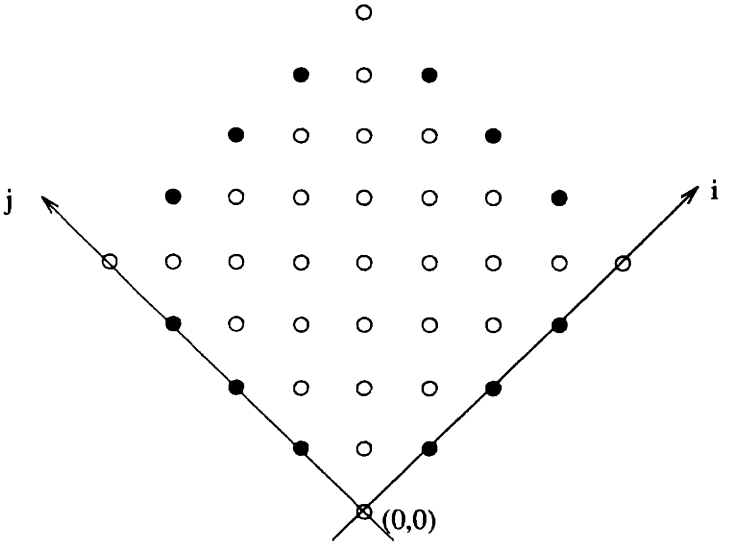

Figure 2-2: Hexagonally sampled space on which Dirichlet conditions are imposed.

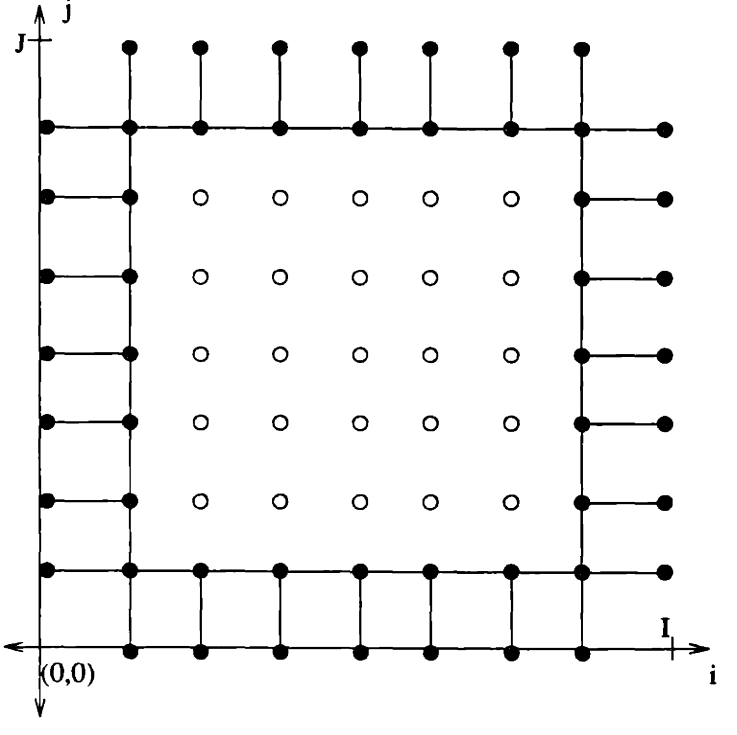

Figure 2-3: Rectangular mesh on which Neumann conditions are imposed.

in Figure 2-3. Stated as a formula, Neumann conditions come in the form

$$V\boldsymbol{x}_{I,J} + W\boldsymbol{x}_{I-1,J-1} = d \tag{2.3}$$

where the vectors $\boldsymbol{x}_{I,J}$ (the outermost edge of the rectangle) and $\boldsymbol{x}_{I-1,J-1}$ are defined as follows

$$\begin{aligned}
x[i,j] \in \boldsymbol{x}_{I,J} \quad \text{iff} \quad & (i,j) \in \{(i = 1 \cup i = I) \cap j \in [2, J-1]\} \cup \\
& \{i \in [2, I-1] \cap (j = 1 \cup j = J)\} \\
x[i,j] \in \boldsymbol{x}_{I-1,J-1} \quad \text{iff} \quad & (i,j) \in \{(i = 2 \cup i = I-1) \cap j \in [2, J-1]\} \cup \\
& \{i \in [2, I-1] \cap (j = 2 \cup j = J-1)\} \\
d \quad \in \quad & \Re^{\{\dim(\boldsymbol{x}_{I,J})-4\} \times 1}
\end{aligned}$$

The important requirement is that both Dirichlet and Neumann B.C.'s be specified such that the overall system, $Ax = b$, including the B.C.'s is well-posed.

This purpose of this discussion of boundary conditions is not to highlight the finer points of B.C. design, but to allow the formulation of a canonic 2-D acausal IIR filter. Once one is given a 2-D linear difference equation constrained by B.C.'s, the next task is to develop a framework and algorithm for which to solve such systems. One would like to be able to propagate the B.C., as was done for 1-D acausal systems, but the boundary is no longer two points at opposite ends of a line; instead it is a continuous region which grows in size with the ROS. The 2-D boundary, however, can be treated as an end-point of an interval on the 1-D line by casting the 2-D IIR filtering system into a one dimension.

One method, used in [5, 9, 16, 20, 24], of casting 2-D acausal IIR filters as 1-D acausal filters is to perform a column-stacking transformation of the mesh. Consider again the acausal NNM filter (2.2) constrained by Neumann B.C.'s, where the output is to be solved over the finite rectangle $\Omega = [1, I] \times [1, J]$ (minus the four corner points, as was illustrated in Figure 2-1). The input is given over the rectangle $[2, I-1] \times [2, J-1]$. The input and

output are stacked into columns x_i and u_i such that

$$x_i = \begin{bmatrix} x[i,0] \\ x[i,1] \\ \vdots \\ x[i,J] \end{bmatrix} \qquad u_i = \begin{bmatrix} u[i,1] \\ u[i,2] \\ \vdots \\ u[i,J-1] \end{bmatrix} \tag{2.4}$$

$$x_i \in \Re^{(J+1)\times 1} \qquad u_i \in \Re^{(J-1)\times 1} \tag{2.5}$$

The Neumann conditions are written in [24] as

$$V_L x_{0,j} + W_L x_{1,j} + V_R x_{I,j} + W_R x_{I-1,j} = d_{H,j} \tag{2.6}$$

for $0 \leq j \leq J$, and also as

$$V_B x_{i,0} + W_B x_{i,1} + V_T x_{i,J} + W_T x_{i,J-1} = d_{V,i} \tag{2.7}$$

for $1 \leq i \leq I-1$. Equations (2.6) and (2.7) are identical in form to Equation (2.3), but in different guise. The subscripts L, R, T, and B denote the left, right, top, and bottom edges of the rectangle Ω, respectively. With the mesh variables augmented into columns, the 2-D system can then be rewritten as the following 1-D system

$$\phi_+ x_{i+1} + \phi_0 x_i + \phi_- x_{i-1} = n_i \quad 1 \leq i \leq I-1 \tag{2.8}$$

where n_i is a vector combination of both u_i and $d_{V,i}$ from the right-hand-side of the "vertical" boundary condition (2.7) (the matrices ϕ_0, ϕ_+, *and* ϕ_- are augmentations of the coefficient matrices given in (2.2) and (2.7); the elements of each matrix in Equation (2.8) are described in detail in [24]). Note how Equation (2.8) is a coupling of adjacent columns in the mesh, which reflects the dynamics of a second-order 2-D difference equation. Since Equation (2.7) is incorporated into the dynamics (2.8), the boundary conditions for this 1-D difference equation (2.8) are just the non-separable B.C.'s given in equation (2.6). Having a 1-D difference equation with B.C.'s, one can then effectively propagate the B.C. in Equation (2.6) through the dynamics (2.8) to produce a constraint at one end of the 1-D interval (which end depends upon the direction of propagation), and then back-substitute to obtain the solution.

This algorithm falls under a more general class of solutions called *Marching Methods* [1, 2], which have been used to solve P.D.E.'s. However, this RTS algorithm, where the state x_i is multi-dimensional, does not in general exist, since ϕ_+, ϕ_-, or any matrix in the B.C. propagation can be singular. For most NNM models, efficient, general solutions do exist (see [24]) when a 2-D acausal filter is cast into a 1-D filter with columns as the state, but there are significant drawbacks.

One drawback to the direct solutions given in [24] is that for regions of large size the state x_i will become proportionately large, and so will the matrices ϕ_0, ϕ_+, and ϕ_-. Operating on such large matrices is both computationally expensive and can create extremely large storage demands. The algorithms in [24] are applicable to a large number of systems, but undoubtedly sacrifice computational efficiency in their generality. Furthermore, the column-stacking transformation creates a 1-D system in which a subset of the 2-D boundary conditions are combined with the filtering dynamics (through the vector n_i and the three ϕ matrices). A subset of the boundary conditions must then be accessed each step of the solution algorithm, a requirement which will later be shown to hamper the creation a parallel, data partitioning algorithm.

An ordering of the output variables $x[i,j]$ which avoids combining the B.C.'s with filter dynamics is given in [37]. A 2-D filter, such as the NNM, is a linear constraint among variables in a local neighborhood of the mesh. A natural state augmentation is to sequentially order variables in local neighborhoods of the 2-D mesh. One example is column-stacking, which is a sequential listing of neighboring columns. In [37], the mesh variables $x[i,j]$ are ordered into concentric states x_ρ, where x_ρ contains all the mesh variables which lie at radial distance of ρ from the center of the mesh. Such an ordering, as will be seen, allows one to formulate the 2-D dynamics as 1-D dynamics with varying state size, which are specified independently from the boundary conditions.

For a system with NNM dynamics, a square ROS where $(i,j) \in [1,I] \times [1,I]$ (assume for simplicity that I is even), inputs given over $[2,I-1] \times [2,I-1]$, and Neumann B.C.'s, a starting point for the radial ordering is to shift the indices such that $(i',j') = (i - I/2, j - I/2)$. The center of the mesh is now $(i',j') = (0,0)$. The first state, x_ρ at $\rho = 0$, consists of $x[0,0]$, since it is obviously at a radial distance of zero from the center of the mesh. If one then decided to use the ∞-norm to define radial distance from the center of the mesh, x_ρ

would be defined as follows:

$$x[i',j'] \in x_\rho \quad \text{iff} \quad \|[i'\ j']^T\|_\infty = \rho \tag{2.9}$$

where $\|[i'\ j']^T\|_\infty$ is the ∞-norm of the index vector. The input is augmented similarly as

$$u[i',j'] \in u_\rho \quad \text{iff} \quad \|[i'\ j']^T\|_\infty = \rho \tag{2.10}$$

The first few states x_ρ from the center of the mesh are illustrated in Figure 2-4. From Figure 2-4 it can be seen that the state dimensions are defined by

$$x_\rho \in \Re^{n_\rho} \qquad u_\rho \in \Re^{n_\rho} \tag{2.11}$$

$$n_\rho = 8\rho + \delta(\rho) \tag{2.12}$$

($\delta(\rho)$ is the Kroneker delta function).

Another possible radial ordering, which might be more appropriate for IIR filters derived from a hexagonally sampled space, is to use 1-norms to define radial distance. If the outputs are again indexed such that the center is $(i',j') = (0,0)$, the states x_ρ and u_ρ become

$$x[i',j'] \in x_\rho \quad \text{iff} \quad \|[i'\ j']^T\|_1 = \rho \tag{2.13}$$

$$u[i',j'] \in u_\rho \quad \text{iff} \quad \|[i'\ j']^T\|_1 = \rho \tag{2.14}$$

$$x_\rho \in \Re^{n_\rho} \qquad u_\rho \in \Re^{n_\rho} \tag{2.15}$$

where the dimension n_ρ is now derived from

$$n_\rho = 4\rho + \delta(\rho) \tag{2.16}$$

The first few states containing the output variables which are of equal 1-norm distance from the center of the mesh are illustrated in Figure 2-5. Note that for any radial ordering of concentric states in the mesh, two-dimensional space has been mapped onto a 1-D line where the index is ρ. Such a mapping is similar to that given by the column-stacking, yet x_ρ now grows in dimension with ρ (the columns have fixed dimension).

For the square region $[1,I] \times [1,I]$, ρ varies from 0 at the center of the mesh to $R =$

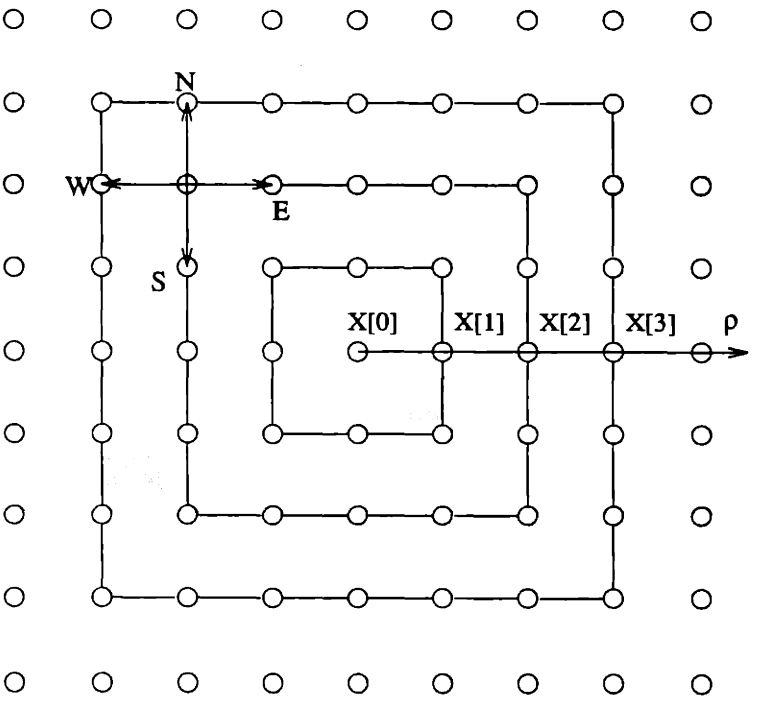

Figure 2-4: States x_ρ containing $x[i,j]$ at equal ∞-norm distance from the center.

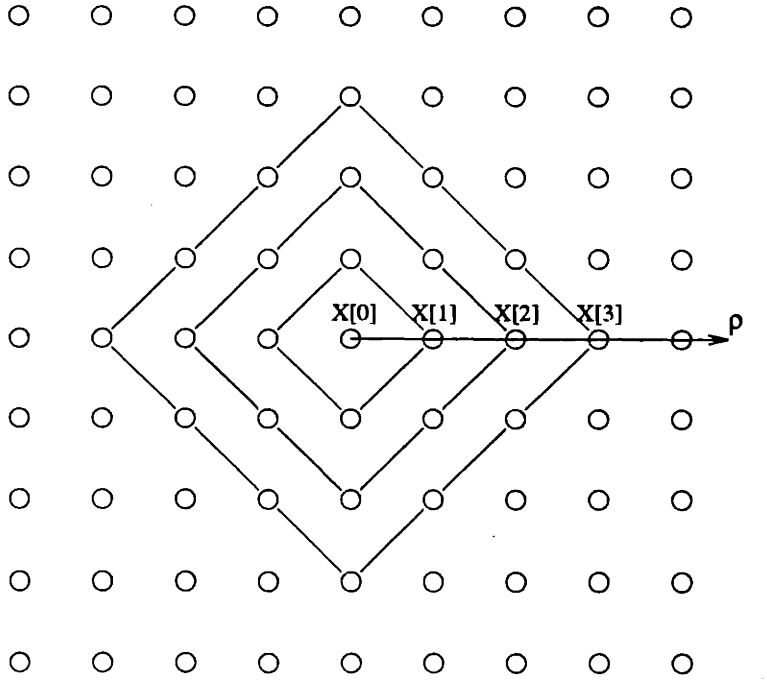

Figure 2-5: States x_ρ containing $x[i,j]$ at equal 1-norm distance from the center.

$(I-1)/2$ at the square's edge, where R can be considered to be the radius of the ROS. Converting the NNM difference equation to a dynamic relationship among the augmented states x_ρ, one obtains (for both 1-norm and ∞-norm orderings)

$$D_\rho x_\rho = F_{\rho+1}x_{\rho+1} + G_{\rho-1}x_{\rho-1} + H_\rho u_\rho \quad \text{for } \rho = 1, 2, ..., R-1 \tag{2.17}$$

$$D_0 x_0 = F_1 x_1 + H_0 u_0 \tag{2.18}$$

where the coefficient matrices D_ρ, F_ρ, G_ρ, and H_ρ are defined in Appendix A, and the dimension of x_ρ is given by Equations (2.12) and (2.16) for ∞-norm and 1-norm radial orderings, respectively. The boundary conditions of Neumann form can be stated exactly as in (2.3) (since x_R and x_{R-1} in (2.17) contain the same elements as $x_{I,J}$ and $x_{I-1,J-1}$, respectively, in (2.3)) and will be written as

$$Vx_R + Wx_{R-1} = d \tag{2.19}$$

If the acausal system's auxiliary conditions were Dirichlet conditions, which for a second-order 2-D system just initialize the outer edge of the square ROS, one would have the B.C.

$$x_R = d \tag{2.20}$$

For any radial ordering of concentric neighboring states, the dynamics will have the exact same form as those in Equations (2.17)-(2.18), as long as the shape of the boundary is such that the farthest radial state from the center contains the output variables $x[i,j]$ which comprise the boundary edge, and even in those cases where the boundary has a shape unlike that of the radial orderings, a little extra work will allow one to cast the filter in this form.

With the B.C. at one end ($\rho = R$) of the 1-D interval given by the 2-D system's boundary conditions, Equation (2.19) in this example, the boundary condition at the other end of the 1-D interval ($\rho = 0$) is given by the NNM dynamics in the center of the mesh, Equation (2.18). Given this 1-D second-order acausal system of growing state dimension defined by the difference equation (2.17) and the *separable* B.C.'s, Equations (2.18) and (2.19), the next section discusses one possible implementation of the solution.

2.2 Solving 1-D Acausal Systems with Growing State Size

In [37], 1-D acausal systems with growing state dimension are discussed in the context of optimal estimation, while side-stepping how to implement the solution of such systems in deterministic form. In Section 1.2, two direct algorithms for scalar 1-D acausal systems were proposed. One method, the Mayne-Fraser, splits the dynamics into its causal and anti-causal components. The RTS algorithm propagated a boundary condition through the filter dynamics to produce the solution at one end of the 1-D interval, and then back-substituted to obtain the solution. For the 1-D system of growing state dimension, created from a 2-D NNM constrained by Neumann conditions over a square region $(i,j) \in [0,I] \times [0,I]$ and rewritten in Equations (2.21)-(2.23), we restrict our development to RTS algorithms, although MF algorithms might also yield suitable solutions.

$$\text{Filter Dynamics:} \qquad D_\rho x_\rho = F_{\rho+1} x_{\rho+1} + G_{\rho-1} x_{\rho-1} + H_\rho u_\rho \quad (2.21)$$

$$\text{for } \rho = 1, 2, ..., R-1 \ (R = I/2)$$

$$\text{(Dynamic) B.C. at 2-D center:} \qquad D_0 x_0 = F_1 x_1 + H_0 u_0 \quad (2.22)$$

$$\text{(2-D) B.C. at 2-D boundary:} \qquad V x_R + W x_{R-1} = d \quad (2.23)$$

$$x_\rho \in \Re^{n_\rho} \qquad u_\rho \in \Re^{n_\rho} \quad (2.24)$$

The MF algorithm is discarded as a viable solution to a 1-D filter of growing state dimension, in part, because it is not clear how to decompose the dynamics in Equation (2.21) into causal and anti-causal recursive filters. Another reason that these algorithms are not considered is that the boundary conditions for the original 2-D NNM, given in Equation (2.23), are needed to initialize the anti-causal filter (anti-causal in ρ); yet, as will become more clear in Section 2.3, a parallel algorithm which physically partitions the filter dynamics cannot use the B.C.'s from the original 2-D system during the initial steps of the algorithm. The same drawback was mentioned for the column-stacking algorithm in Section 2.1.

An algorithm derived from the RTS class can always be developed for a linear filtering system, since RTS algorithms are just variants of Gaussian elimination. Furthermore, while there is no apparent advantage to using the RTS algorithm for scalar 1-D acausal filters, they appear to be ideal frameworks for 2-D acausal filtering. Because RTS algorithms involve operations on sets of linear equations, the 2-D NNM filtering system is first shown

using Equations (2.21)-(2.23) as a single matrix equation of very large dimension

$$Ax = Hu \tag{2.25}$$

$$A = \begin{bmatrix} D_0 & -F_1 & 0 & 0 & \cdots & 0 \\ -G_0 & D_1 & -F_2 & 0 & \cdots & 0 \\ 0 & -G_1 & D_2 & -F_3 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -G_{R-2} & D_{R-1} & -F_R \\ 0 & \cdots & 0 & 0 & W & V \end{bmatrix}$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{R-1} \\ x_R \end{bmatrix} \qquad u = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{R-1} \\ d \end{bmatrix}$$

$$H = diag\{H_0, H_1, \ldots, H_{R-1}, I\}$$

Equation (2.25) is a block tri-diagonal system of equations with varying block size (the blocks are defined in Appendix A). The contents of the individual blocks is determined by the radial ordering. For the overall system to be well-posed and well-conditioned, the B.C.'s (matrices V and W) and the NNM coefficients must be defined such that A is invertible and sufficiently well-conditioned. It is noted in [24] that, for large sparse systems of equations like (2.25), very efficient iterative methods like SOR and preconditioned conjugate gradient can produce solutions for some NNM's. Iterative methods, however, can be slow for a large class of NNM models, and direct methods will be superior when the costs of the factorization can be amortized over a large number of inputs.

2.2.1 Block LU Factorization

The most straightforward solution to a system like (2.25) is to perform an LU factorization of A (Cholesky factorization if A is symmetric positive definite). A Block LU factorization

is considered first in this thesis, not only for its simplicity, but also because there has been little study on the properties of 1-D acausal systems with growing state dimension, with the exception of [31, 37]. The properties of 1-D acausal systems with fixed state dimension, however, has been studied extensively under the name of *descriptor systems* in [23, 24, 28, 27, 34, 33, 32, 38], and it is likely that much of this work will extend to the varying-dimension case.

When A in the linear system $Ax = b$ is block tri-diagonal with regularly sized blocks and sufficiently well-conditioned, there exists a block LU factorization which efficiently manipulates the sparse structure of A [16, 17], although it doesn't make use of any possible sparsity in the blocks D_ρ, F_ρ, or G_ρ. Furthermore, an allusion to the existence of a block LU algorithm for tri-diagonal systems with irregular block size is made in [17]. Such a factorization is given in (2.26) and is similar to the algorithm for regularly sized blocks given in [16].

$$LU = \begin{bmatrix} I & 0 & 0 & \cdots & 0 & 0 \\ L_1 & I & 0 & \cdots & 0 & 0 \\ 0 & L_2 & I & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & & L_{R-1} & I & 0 \\ 0 & \cdots & & 0 & L_R & I \end{bmatrix} \begin{bmatrix} D_0 & -F_1 & 0 & 0 & \cdots & 0 \\ 0 & \tilde{D}_1 & -F_2 & 0 & \cdots & 0 \\ 0 & 0 & \tilde{D}_2 & -F_3 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \tilde{D}_{R-1} & -F_R \\ 0 & \cdots & 0 & 0 & 0 & \tilde{D}_R \end{bmatrix} \tag{2.26}$$

where $A = LU$ and

$$L_{\rho+1}\tilde{D}_\rho = -G_\rho \tag{2.27}$$

$$\tilde{D}_\rho = D_\rho + L_\rho F_\rho \tag{2.28}$$

For simplicity in notation, the matrices D_R and G_{R-1} in Equations (2.27) and (2.28) are equivalent to V and W, respectively, in the large matrix A.

The recursive procedure defined in Equations (2.27) and (2.28) is initialized for $\rho = 0$ by the fact that $\tilde{D}_0 = D_0 = 1$. For radii ρ larger than zero, however, the recursion depends upon the ability to invert $\tilde{D}_\rho$ in order to yield meaningful numerical results. There are NNM parameters (N, S, E, W) detailed in Appendix B for which $\tilde{D}_\rho$ is singular or poorly-conditioned, yet for most NNM parameters it can be assumed that $\tilde{D}_\rho$ is well-conditioned

for all ρ (see examples in Chapter 3). The recursive procedure given by Equations (2.27) and (2.28) can be used to form an RTS algorithm which propagates the B.C. at $\rho = 0$, Equation (2.22), through the dynamics to the other end of the 1-D interval (at $\rho = R$). Substituting Equation (2.22) into the dynamics at $\rho = 1$ yields

$$\begin{aligned} D_1 x_1 &= F_2 x_2 + G_0(F_1 x_1 + H_0 u_0) + H_1 u_1 \\ (D_1 - G_0 F_1) x_1 &= F_2 x_2 + G_0 H_0 u_0 + H_1 u_1 \\ \tilde{D}_1 x_1 &= F_2 x_2 + \Psi_1 \\ \text{where} \quad & \Psi_1 = G_0 H_0 u_0 + H_1 u_1 \end{aligned} \tag{2.29}$$

Note that $\tilde{D}_1$ can also be derived from (2.28). The B.C. at $\rho = 0$ has been propagated to $\rho = 1$ to yield the constraint in (2.29). Given this constraint, one is then only concerned with the dynamics for $\rho \in [2, R-1]$. Substituting (2.29) into the dynamics at $\rho = 2$ yields

$$\begin{aligned} D_2 x_2 &= F_3 x_3 + G_1 \tilde{D}_1^{-1}(F_2 x_2 + \Psi_1) + H_2 u_2 \\ D_2 x_2 &= F_3 x_3 - L_2 F_2 x_2 - L_2 \Psi_1 + H_2 u_2 \end{aligned} \tag{2.30}$$

where L_2 can also be derived directly from (2.27). Letting $\Psi_2 = -L_2 \Psi_1 + H_2 u_2$, one gets from (2.30) the following

$$\tilde{D}_2 x_2 = F_3 x_3 + \Psi_2 \tag{2.31}$$

At the n-th step of the algorithm, one substitutes the constraint

$$\tilde{D}_{n-1} x_{n-1} = F_n x_n + \Psi_{n-1} \tag{2.32}$$

into the dynamics (2.21) at $\rho = n$ to yield

$$\begin{aligned} \tilde{D}_n x_n &= F_{n+1} x_{n+1} + \Psi_n \\ \Psi_n &= -G_{n-1} \tilde{D}_{n-1}^{-1} \Psi_{n-1} + H_n u_n \end{aligned} \tag{2.33}$$

This process continues recursively until one finally has

$$\tilde{D}_{R-1} x_{R-1} = F_R x_R + \Psi_{R-1} \tag{2.34}$$

which can then be combined with the Neumann or Dirichlet conditions to obtain x_R. Before back-substitution begins, one has the following block upper-triangular system

$$Ux = \Psi \tag{2.35}$$

$$\Psi = \begin{bmatrix} \Psi_0 \\ \Psi_1 \\ \Psi_2 \\ \vdots \\ \Psi_{R-1} \\ \Psi_R \end{bmatrix}$$

where U is defined in (2.26) and $\Psi = L^{-1}Hu$ (found by substituting (2.26) into (2.25) and inverting L, although one never explicitly inverts L). The elements of Ψ can also be derived from the recursive relation

$$\Psi_\rho = L_\rho \Psi_{\rho-1} + H_\rho u_\rho \tag{2.36}$$

Once Ψ and U have been computed, the back-substitution follows from Equation (2.35) as

$$x_\rho = \tilde{D}_\rho^{-1}(F_{\rho+1}x_{\rho+1} + \Psi_\rho) \tag{2.37}$$

The block LU algorithm can then be summarized by the two steps: (1) *Factorization* – propagating the B.C. at $\rho = 0$ with the recursions given in Equations (2.27), (2.28), and (2.36) (2) *Back-substitution* – Solving for x_R and using it to initialize the recursion given by Equation (2.37), yielding the rest of the output. The factorization and back-substitution steps together correspond to Block Gaussian Elimination.

The block LU algorithm has many undesirable properties, especially for large mesh sizes. The first problem is that a great deal of storage is required for the factorization. The factorization yields the constraint given by Equation (2.37), which must be stored for all ρ in $[0, R-1]$. Required storage is dominated by storing the matrix $(\tilde{D}_\rho^{-1}F_{\rho+1})$ for all ρ, which can be found in Appendix A to have the dimension $n_\rho \times n_{\rho+1}$. The total number of elements to be stored is then approximately

$$\sum_{\rho=0}^{R} n_\rho^2$$

which for ∞-norm ordering becomes (remember that $R = I/2$)

$$\sum_{\rho=0}^{R}(8\rho+1)^2 \simeq \frac{8}{3}I^3 + 2I^2 = \mathcal{O}(I^3)$$

The reason for this tremendous growth in storage is that, while A and its blocks becomes more sparse for larger I, the matrices $\tilde{D}_\rho$ and their inverses are full; namely, the Block LU algorithms creates a lot of fill-in of the zeros in A. Note that the factorization would also require $O(I^3)$ storage elements if column-stacking were used. For large-sized ROS, these storage requirements can become overwhelming. With double precision arithmetic (8 bytes storage require per matrix entry) and mesh of size 256×256 ($I = 255$), the total amount of storage required is approximately 45 MB.

Another drawback, which is a direct result of the fill-in created in the factorization, is growth in computational complexity. The computational requirements are dominated by inversions of $\tilde{D}_\rho$, which must be found to compute $(\tilde{D}_\rho^{-1}F_{\rho+1})$ in Equation (2.37). Because $F_{\rho+1}$ is extremely sparse, creating the matrix $(\tilde{D}_\rho^{-1}F_{\rho+1})$ essentially amounts to computing the inverse of $\tilde{D}_\rho$. Computing the inverse of a dense matrix of dimension $n \times n$ generally requires $2/3n^3$ flops [16], so to compute the inverse of $\tilde{D}_\rho$ for all $\rho \in [0, R]$ will take on the order of $O(I^4)$ flops. The derivation of this bound uses the fact that n_ρ is a linear function of ρ, and hence

$$\sum_{\rho=0}^{R}\frac{2n_\rho^3}{3} = \mathcal{O}(I^4)$$

Note that, because the state x_ρ becomes very large near the boundary ($\rho = R$), the computations are dominated by inverting $\tilde{D}_\rho$ near the boundary. The fact that the number of arithmetic operations is $O(I^4)$ and the storage required is $O(I^3)$ prevents any practical consideration of the block LU algorithm for large mesh sizes.

A final drawback mentioned earlier in this section is that for some combinations of NNM parameters (N, S, E, W) the matrix $\tilde{D}_\rho$ in Equation (2.27) will become either poorly-conditioned or singular. A small but significant listing of those parameters which lead to singularities is provided in Appendix B, while a complete listing is beyond the scope of this thesis. For most NNM filters of interest, however, $\tilde{D}_\rho$ is well-conditioned for all ρ, while in some other cases, singularities in $\tilde{D}_\rho$ can be avoided by changing the radial ordering, such as from a 1-norm to an ∞-norm ordering. Changing the radial ordering is similar to

pivoting in an LU factorization, since one is changing the order of the equations in $Ax = b$.

In spite of the many drawbacks, the Block LU algorithm is numerically stable for "most" NNM parameters, in which case it produces highly accurate solutions. Furthermore, this algorithm is able to produce solutions for a large class of models in which iterative schemes fail. More general solutions, which side-step the singularity constraint, can most likely be found in the Descriptor Systems literature, such as [24]; yet the Descriptor Systems literature fails to overcome the massive storage and computational requirements needed to implement a 2-D IIR filter given on a mesh of appreciable size. In the remainder of this thesis, a number of algorithms which are much more efficient than the straightforward Block LU algorithm will be discussed, all of which have highly parallel implementations. In particular, a *parallel* Block LU algorithm based upon partitioning the 2-D mesh will be developed in Section 2.3, and a less general yet more efficient approximation to the algorithm is detailed in Chapter 3. These algorithms are then compared to the *nested dissection* algorithm, which is known to be the most efficient algorithm for solving sparse linear systems generated by discretizations of P.D.E.'s. Nested dissection requires as little as $O(I^3)$ arithmetic operations and $O(I^2 \ln I)$ storage elements [13].

2.2.2 An Simple Example

Before motivating the parallel version of the LU algorithm, it is worthwhile to use a simple example to explain in a little more detail how the mesh variables are eliminated radially from the center. From the recursion given by Equations (2.27)-(2.28), one obtains the factorization shown in (2.26). The matrix U in (2.26) effectively yields constraints between neighboring states x_ρ and $x_{\rho-1}$. These constraints are used for the back-substitution recursion. This process is identical to the propagation mentioned in Chapter 1 of B.C.'s from one end of an interval to another for acausal 1-D systems.

Looking a little more closely at the Gaussian elimination process, consider an NNM constrained by Dirichlet conditions on a square region of size n-by-n. For $n = 7$, an ∞-norm radial ordering of output variables is shown in Figure 2-6. State x_0 contains the first element in Figure 2-6, and state x_1 contains elements 2 through 9. The NNM difference equation evaluated at the center of the mesh gives a constraint between x_0 and x_1. The first variable eliminated in the LU factorization is x_0, yielding the new mesh given in Figure 2-7. As was given by Equation (2.29), a constraint now exists solely between x_2 and x_1. The second

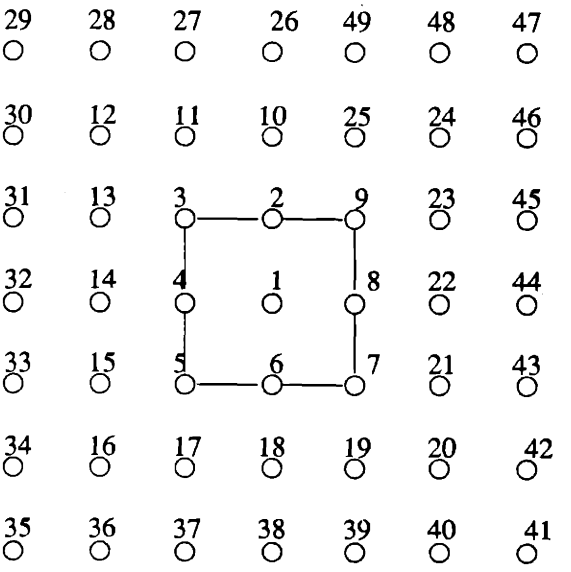

Figure 2-6: Radial ordering of the variables for a square 7-by-7 mesh.

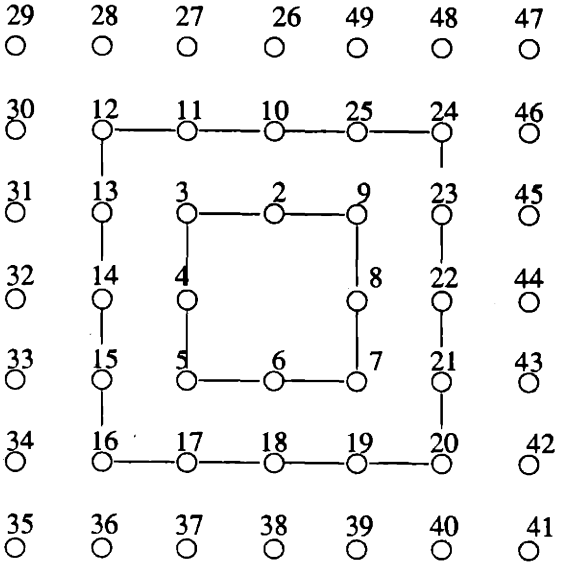

Figure 2-7: Mesh variables remaining after the first variable has been eliminated.

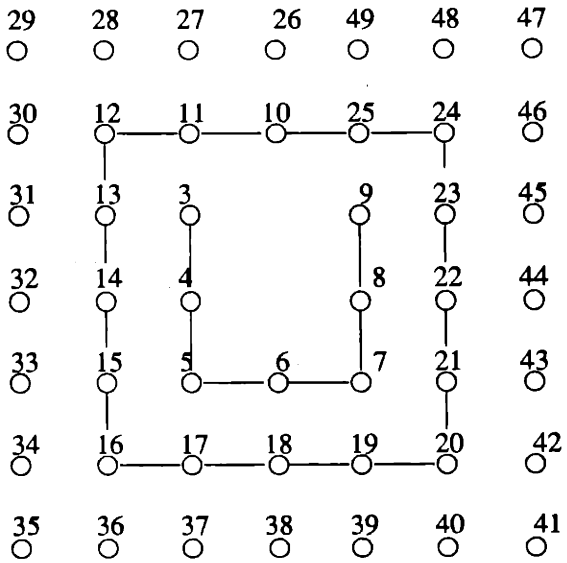

Figure 2-8: Mesh variables remaining after the first two variables have been eliminated.

element is then eliminated from the mesh, with the remaining variables shown in Figure 2-8. Eliminating elements 3 through 9 gives a constraint between x_3 and x_2, shown in Figure 2-9. For a mesh of large size, this process of variable elimination would continue recursively (unless a zero pivot is encountered) in a direction radially outward from the center of the mesh to yield a constraint between the two outermost states x_R and x_{R-1}, which in this example ($R = 3$) are x_3 and x_2, respectively. The outermost radial state, x_3, is given by the Dirichlet condition, and the rest of the solution is obtained by back-substituting in a direction radially inward towards the center of the mesh.

2.3 Parallel Algorithms

2.3.1 Local Factorization

In the Block LU algorithm, the 2-D boundary conditions are needed for only one step of the algorithm – to initialize the back-substitution. For the factorization step, one only needs the filter constraints (dynamics), and for the back-substitution one only needs the the constraints created by the factorization, respectively. The factorization and back-substitution step can thus be done in parallel by giving each processor access to a local set of the dynamic constraints. The mesh can be divided equally among the processors, and each processor

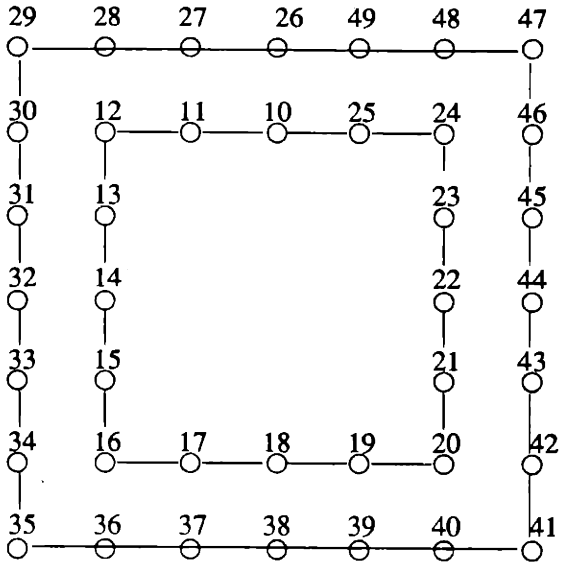

Figure 2-9: Constraint yielded after x_0 and x_1 have been eliminated. For a 7-by-7 mesh, this is also the constraint propagated to the boundary edge.

will order the mesh variables radially with respect to a local center. Given access to the filter constraints on this local mesh, variable elimination then follows in a direction radially outward from the local center, as was done in the previous section with one processor.

For instance, consider an NNM difference equation constrained by Dirichlet conditions on a square mesh of size 14-by-14. A natural partitioning of the filter dynamics is shown in Figure 2-10, where each processor is assigned a local mesh of size 7-by-7. The radial state x_1 is marked in Figure 2-10 for each processor. The factorization step of Gaussian Elimination is then done simultaneously for each local mesh in exactly the same manner as was illustrated in Figures 2-6 through 2-9. The result of the local factorizations is to produce a constraint on the boundary of each processor's mesh, as is illustrated in Figure 2-11, which can be seen as a parallel version of Figure 2-9. Furthermore, a radius L can be defined for the local mesh. In this example, $L = 3$. For the time-invariant NNM filter, the local constraints all have a form identical to that of Equation (2.34), namely

$$\tilde{D}_{L-1} x_{L-1} = F_L x_L + \Psi_{L-1} \tag{2.38}$$

where x_L contains the mesh variables at the boundary edge of a local mesh. The equations stored as a result of the local factorizations will be identical in form to Equation (2.37) for

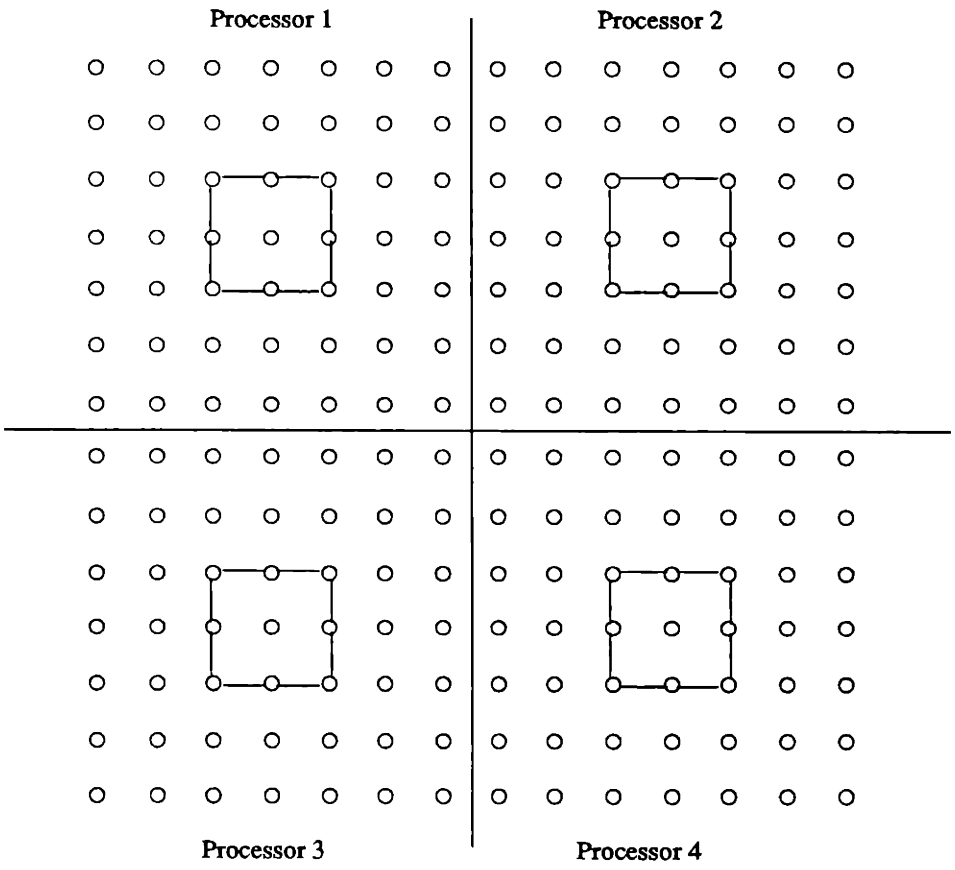

Figure 2-10: Partitioning of 2-D filter dynamics among 4 processors.

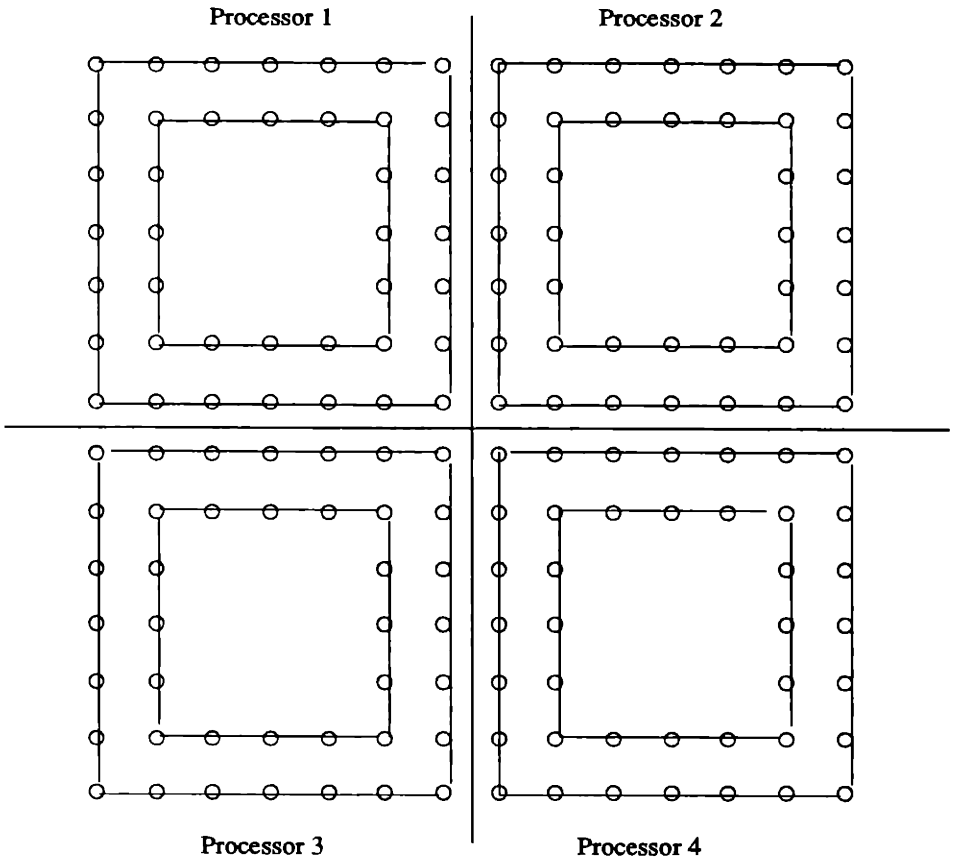

Figure 2-11: Results of eliminating variables ordered radially in each local mesh.

each processor, except that the vectors Ψ_ρ will differ among the processors. The vector Ψ_ρ is a linear combination of the input variables for the local mesh, yet the matrices $(\tilde{D}_\rho^{-1} F_{\rho+1})$ where $\rho \in [0, L-1])$ are a function of the NNM parameters; hence, only one set of these matrices needs to be computed or saved for all of the processors.

2.3.2 Inter-processor Communication

At this stage of the algorithm, back-substitution cannot begin, since the 2-D filter's B.C.'s are given at the global boundary only. Also, the processors cannot eliminate any more variables if they are assumed only to have access to local data. What must follow is communication between the processors, such that the constraints on the local boundaries can be combined into a constraint on the global boundary (2.34); yet, the local constraints alone do not determine the constraint on the global boundary. Instead, the NNM constraints which were not accessed during the local factorizations are used to eliminate the variables $x[i,j]$ in the local boundaries which are not part of the global boundary. A simple parallel algorithm can be derived from [37] which combines the local constraints pairwise.

For instance, consider the set of local constraints illustrated in Figure 2-11. Processor 1 could "communicate" with processor 2 by combining their local constraints with the filter dynamics which form constraints involving mesh variables in both local boundaries. One can then derive a constraint on the boundary of the union of the two local meshes covered by processors 1 and 2. Similarly, processors 3 and 4 can be "communicating" simultaneously with the exchange between processors 1 and 2. This pairwise combination of "east" and "west" local constraints is illustrated in Figure 2-12 for a local mesh of radii $L = 7$. The arrows represent the NNM dynamics which link the two meshes, and the hollow circles indicate the mesh variables which are to be eliminated by the pairwise combination. As a result, a constraint will be produced among the variables represented by dark circles.

After these two pairwise combinations have occurred, the 14-by-14 mesh is effectively divided into two halves, where the mesh variables at the boundaries of both halves are constrained by the results of the first inter-processor communication step. These two constraints can then be combined in the same pairwise fashion, as is illustrated in Figure 2-13. The NNM dynamics which link the boundaries between the north and south halves of the mesh are illustrated by the arrows in Figure 2-13, and the hollow circles represent the variables from the local constraints which are to be eliminated. Once these variables have been

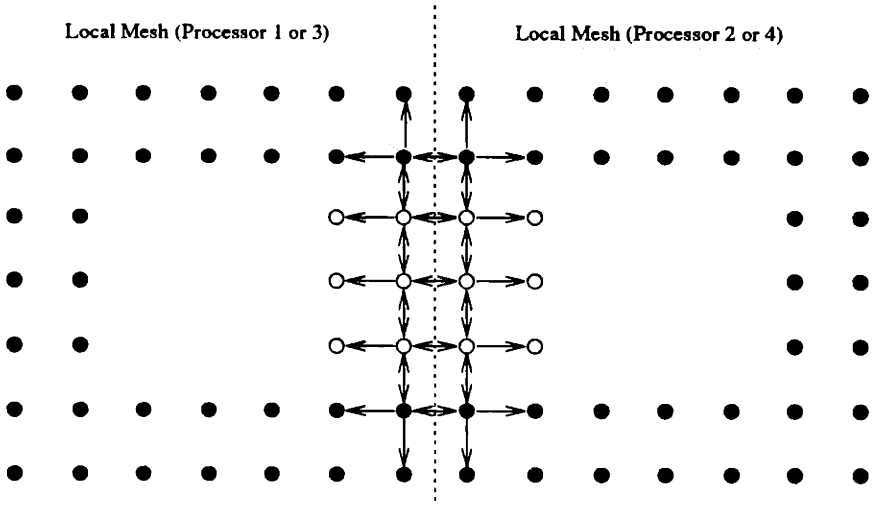

Figure 2-12: Pairwise (East/West) communication between local processors to provide a constraint on a larger boundary.

eliminated, one has a constraint on the two states at the global boundary.

There are a number of ways to implement the inter-processor communication steps of the factorization illustrated in Figures 2-12 and 2-13. A simple option follows, which leads to a more efficient algorithm which is described in Chapter 3. Since any east-west combination is identical in form, only a general derivation is given. For our example 14-by-14 mesh, the local constraint produced in the west half of the mesh by either processor 1 or 3 has the form

$$\tilde{D}_{L-1}x^w_{L-1} = F_L x^w_L + \Psi^w_{L-1} \tag{2.39}$$

and the corresponding constraint produced in the east half of the mesh by either processor 2 or 4 has the form

$$\tilde{D}_{L-1}x^e_{L-1} = F_L x^e_L + \Psi^e_{L-1} \tag{2.40}$$

Combining Equations (2.39) and (2.40) with the NNM dynamics which link the two meshes, one obtains the large matrix equation

$$Zx^{ew} = \Psi^{ew} \tag{2.41}$$

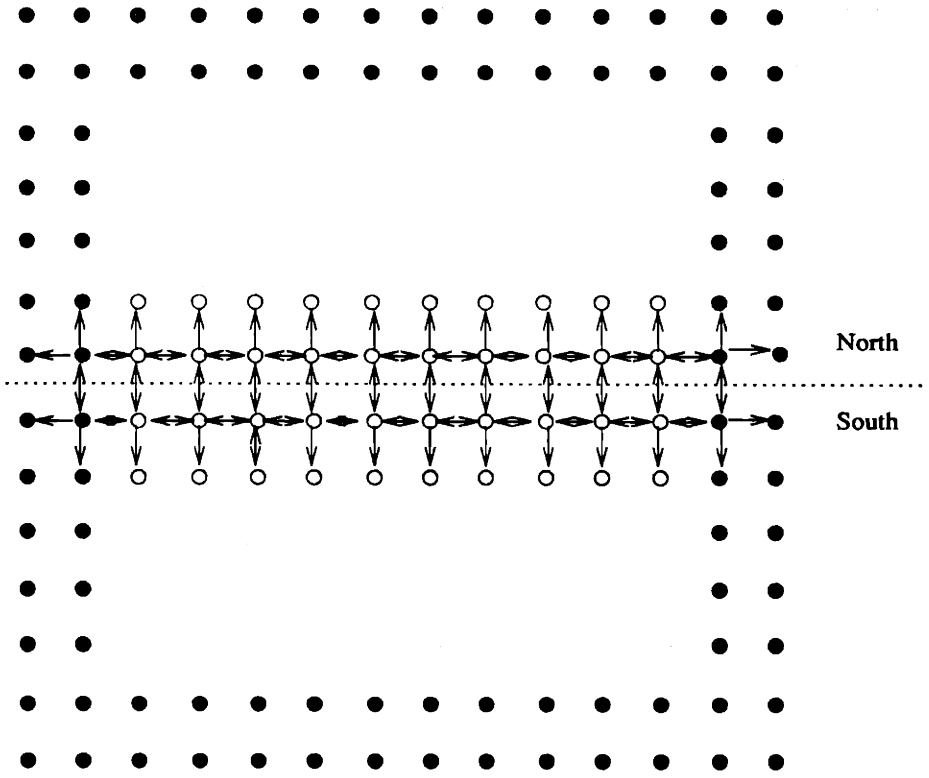

Figure 2-13: Pairwise (North/South) communication between local processors to provide a constraint on the global boundary.

$$\begin{bmatrix} \tilde{D}_{L-1} & -F_R & 0 & 0 \\ 0 & 0 & \tilde{D}_{L-1} & -F_R \\ Z_1 & Z_2 & 0 & Z_3 \\ 0 & Z_4 & Z_5 & Z_6 \end{bmatrix} \begin{bmatrix} x^w_{L-1} \\ x^w_L \\ x^e_{L-1} \\ x^e_L \end{bmatrix} = \begin{bmatrix} \Psi^w_{L-1} \\ \Psi^e_{L-1} \\ diag\{B, B, \ldots, B\}u^w \\ diag\{B, B, \ldots, B\}u^e \end{bmatrix}$$

$$Z \in \Re^{2(10L-9)\times 16(2L-1)} \quad \Psi^e_{L-1}, \Psi^w_{L-1} \in \Re^{8(L-1)\times 1} \quad Z_i \in \Re^{(2L-1)\times 1} \quad i = 1, \ldots, 6$$

$$x^e_L, x^w_L \in \Re^{8L\times 1} \quad x^e_{L-1}, x^w_{L-1} \in \Re^{8(L-1)\times 1} \quad u^w, u^e \in \Re^{(2L-1)\times 1}$$

where the last two block rows of (2.41) correspond to the NNM constraints which link the east and west meshes. These two block rows have the same form as the rows of A in Equation (2.25), where the elements in any row sum to $(1 - N - S - E - W)$. This sum is determined by the coefficients in the difference equation (2.2). The vectors u^w and u^e just correspond to the inputs to the NNM constraints along the west and east local mesh boundaries, respectively, which link the two local meshes. Given Equation (2.41), it is now possible to eliminate those mesh variables which correspond to the hollow circles in Figure 2-12. These variables can be lumped into the state X_{elim}. The two concentric rectangles of mesh variables which remain (illustrated with dark circles) can be lumped into the states X_{in} and X_{out}, corresponding to the inner and outer rectangle, respectively. The elements in X_{in} and X_{out} can be ordered counter-clockwise like those in Appendix A. A permutation matrix P_1 can then be found such that

$$P_1 x^{ew} = \begin{bmatrix} X_{elim} \\ X_{in} \\ X_{out} \end{bmatrix} \tag{2.42}$$

$$P_1^T P_1 = I$$

$$X_{elim} \in \Re^{4(2L-3)\times 1} \quad X_{in} \in \Re^{6(2L-1)\times 1} \quad X_{out} \in \Re^{2(6L+1)\times 1}$$

The permutation given by P_1 is defined such that the variables in X_{in} and X_{out} are ordered counter-clockwise, as are the variables in x_ρ given the orderings in Appendix A. With the

permutation P_1, Equation (2.41) becomes

$$(ZP_1^T)(P_1 x^{ew}) = \Psi^{ew}$$

If the matrix $\tilde{D}_{L-1}$ is diagonally dominant, it is desirable (as will be shown in Chapter 3) to maintain this structure in Z. The permuted matrix ZP_1^T, however, is unlikely to exhibit any diagonal dominance. A row permutation P_2 of Equation (2.41)

$$(P_2 Z P_1^T)(P_1 x^{ew}) = P_2 \Psi^{ew} \tag{2.43}$$

$$\begin{bmatrix} Q_{11} & Q_{12} & 0 \\ Q_{21} & Q_{22} & -F_{out} \end{bmatrix} \begin{bmatrix} X_{elim} \\ X_{in} \\ X_{out} \end{bmatrix} = \begin{bmatrix} \Psi_{elim} \\ \Psi_{in} \end{bmatrix} \tag{2.44}$$

$$Q_{11} \in \Re^{4(2L-3)\times 4(2L-3)} \quad Q_{22} \in \Re^{6(2L-1)\times 6(2L-1)} \quad F_{out} \in \Re^{6(2L-1)\times 6(2L+1)}$$

can be created such that Q_{22} is diagonally dominant if $\tilde{D}_{L-1}$ is diagonally dominant. The permutation P_2 follows from the observation that if the element with maximum magnitude in row i of ZP_1^T is in column j, then row i of ZP_1^T becomes row j of $P_2 Z P_1^T$ (of course, one would never implement these permutations as matrix multiplications!). One should note that because Z is not square, the dimensions of the permutations P_1 and P_2 differ.

$$\dim\{P_1\} = (\dim\{X_{elim}\} + \dim\{X_{in}\} + \dim\{X_{out}\})^2$$

$$\dim\{P_2\} = (\dim\{X_{elim}\} + \dim\{X_{in}\})^2$$

The first step of inter-processor communication attempts to obtain a constraint between X_{in} and X_{out}, which can be obtained by eliminating Q_{21} from (2.43).

Finding the Schur Complement [12] of the matrix

$$\begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \tag{2.45}$$

one gets

$$\tilde{Q}_{22} = Q_{22} - Q_{21}Q_{11}^{-1}Q_{12} \tag{2.46}$$

Using the Schur Complement in Equation (2.43) yields

$$\begin{bmatrix} Q_{11} & Q_{12} & 0 \\ 0 & \tilde{Q}_{22} & -F_{out} \end{bmatrix} \begin{bmatrix} X_{elim} \\ X_{in} \\ X_{out} \end{bmatrix} = \begin{bmatrix} \Psi_{elim} \\ \tilde{\Psi}_{in} \end{bmatrix} \tag{2.47}$$

$$\tilde{\Psi}_{in} = \Psi_{in} - Q_{21}Q_{11}^{-1}\Psi_{elim} \tag{2.48}$$

The bottom block row of (2.41) gives the desired constraint between the two states, X_{in} and X_{out}, which comprise the boundary of the union of the east and west local meshes. Also note that this block row is identical in form and function to Equation (2.38). Note that, since at this stage of the algorithm only those mesh variables interior to the state X_{in} have been eliminated, the block matrix F_{out} in Equation (2.47) is identical in function to the F_ρ in the dynamics given in (2.21)-(2.22), thus the name. The matrix F_{out} can be derived directly from the NNM parameters, as is shown in Appendix A.

After the first inter-processor communication step in our 4-processor example, one has two equations like (2.47), one for the north and one for the south half of the mesh. The second inter-processor communication step is identical to the first, beginning with the constraint produced by communication between processors 1 and 2 and that from processors 3 and 4. The constraint from processors 1 and 2 has the following form, with the superscript n denoting variables in the north half of the mesh.

$$\tilde{Q}_{22}X_{in}^{n} = F_{out}X_{out}^{n} + \tilde{\Psi}_{in}^{n} \tag{2.49}$$

Similarly, the constraint produced by communication between processors 3 and 4 is as follows, with the superscript s denoting variables in the south half of the mesh.

$$\tilde{Q}_{22}X_{in}^{s} = F_{out}X_{out}^{s} + \tilde{\Psi}_{in}^{s} \tag{2.50}$$

These constraints are combined with the NNM dynamics which link the north and south halves of the mesh to produce a constraint on the global boundary. This constraint is iden-

tical to that given in (2.34), since both are constraints on the two outermost states at the global boundary. In particular, $\tilde{Q}_{22}$ for the last inter-processor factorization step must equal to $\tilde{D}_{R-1}$ obtained by the straightforward factorization, since both factorizations manipulate the same set of equations. If one were to divide the mesh into more than four sub-regions, which will be shown to be a more efficient implementation, more inter-processor communication steps are required. In fact, for 2^N local meshes, $(N-1)$ inter-processor steps are needed. With more than 4 local meshes, the inter-processor communication steps continue recursively in the form given by Equations (2.41)-(2.47), alternating between pairwise east-west and pairwise north-south combinations of constraints. When the global boundary is reached, the constraint (2.34) is obtained.

Back-substitution follows easily from the permutations and the results of each inter-processor communication step, stored in the the form of Equation (2.47). Given the values at the outermost state X_{out}, the interior rectangle X_{in} can be found from Equation (2.47) as

$$X_{in} = \tilde{Q}_{22}^{-1}(F_{out}X_{out} + \tilde{\Psi}_{in}) \tag{2.51}$$

and X_{elim} is then given by

$$X_{elim} = Q_{11}^{-1}(-Q_{12}X_{in} + \tilde{\Psi}_{elim}) \tag{2.52}$$

Note that Q_{11} and $\tilde{Q}_{22}$ have been assumed to be non-singular in this algorithm, which is the case for most filters. Also note the similarity between the inter-processor back-substitution steps and the local factorization back-substitution step given by (2.37). The back-substitution steps of inter-processor communication begin at the global boundary and end with the output values at all the points in the local boundaries (those points in Figure 2-11 for our example). Back-substitution for the local processing can then begin, propagating the known values at the local boundaries radially inward toward the local centers.

2.3.3 Computation and Storage

In this thesis, the *number of computations* generally refers to the number of floating point divisions and multiplications required to compute the factorization of the matrix A in $Ax = b$. This convention is used in many numerical linear algebra texts [14, 16], and can be thought of as an on-line measure of complexity for solving 2-D boundary value systems.

However, in many filtering applications, the costs of the factorization can be amortized over a large number of inputs. For these off-line solutions, the computational complexity of the algorithm is determined by the back-substitution processing of the input data.

For any 2-D IIR filter with a mesh of finite size I implemented with the parallel mesh partitioning algorithm, there must be an optimal number of processors which produces the maximum savings in computation and storage over the straightforward uni-processor implementation. This optimal number will undoubtedly depend upon the size of the mesh I and the number of partitions M, but will also depend upon the processor network's characteristics, such as latency and bandwidth. Without knowing these factors *a priori*, a general measure of complexity and storage is difficult to specify. However, we can assume that the number of computations generally takes the form

$$S = \frac{K_1(I)}{M} + K_2(I)\log(M) \tag{2.53}$$

where the first term expresses the fact that for any given mesh size I, the number of computations required for local processing decreases with the number of partitions. However, the number of computations due to inter-processor communication should grow logarithmically with M for a given mesh size I. This contribution is expressed by the second term of S. Two extreme examples of fine and course grain parallelism give a flavor of what computational savings are to be expected from the partitioning.

Assume a square mesh of size $I \times I$. To simplify our analysis without losing any generality, it will be assumed throughout this discussion that the dimension I is such that the local meshes are square and their edges have an odd number of points. This requirement allows for the radial ∞-norm orderings to fit "evenly" within the local mesh, as in Figures 2-6 and 2-10.

Consider a mesh square mesh of size I-by-I where $I = 2(2L + 1)$. The global radius of the mesh is then $R = 2L$, and the mesh can be divided into four local meshes of radius L. As is mentioned in Section 2.2.1, the number of computations for the straightforward single mesh factorization is dominated by the computation of $\tilde{D}_\rho^{-1}$ at each step of the factorization. Summing over $\rho = 1, 2, \ldots, L-1$, it was shown in Section 2.2.1 that the total number of

computations for the factorization is bounded by

$$C_1 = \sum_{\rho=0}^{R} \frac{2n_\rho^3}{3} = \mathcal{O}(I^4) \tag{2.54}$$

Furthermore, the number of storage elements required is dominated by storing the full matrix $\tilde{D}_\rho^{-1} F_{\rho+1}$ at each step of the factorization. The total number of storage elements required was shown to be

$$S_1 = \sum_{\rho=0}^{R} (8\rho + 1)^2 \simeq \frac{8}{3} I^3 + 2I^2 = \mathcal{O}(I^3) \tag{2.55}$$

However, if one decides to implement the Block LU algorithm by dividing the mesh into four equal meshes of radius L, which is an example of course-grain parallelism for large I, the computational and storage requirements are quite different.

With 4 local mesh, the number of computations at each step of the local factorization is dominated by the computation of $\tilde{D}_\rho^{-1}$, where $\rho = 1, 2, \ldots, L = R/2$. Summing over the local factorizations (note that for a linear shift-invariant difference equation, the matrix $\tilde{D}_\rho^{-1}$ is the same for each local set of data) one obtains

$$S_4^L = \sum_{\rho=0}^{L} \frac{2n_\rho^3}{3} = \mathcal{O}(L^4) = \frac{1}{16}\mathcal{O}(I^4) \tag{2.56}$$

computations. If the filter is not shift-invariant, one would multiply the number S_4^L by four to indicate that local factorizations are unique for each local mesh. For each of the two inter-processor steps, the total number of computations is dominated by computing the inverses of the matrices $\tilde{Q}_{22}$ and Q_{11} in Equation (2.47). The dimensions of these matrices are given after Equation (2.44). We note that for the first inter-processor factorization step

$$\dim\{\tilde{Q}_{22}\} \simeq \dim\{\tilde{D}_{3R/4}\} \qquad \dim\{Q_{11}\} \simeq \dim\{\tilde{D}_{R/2}\}$$

There would be two sets of these matrices—one for the north half of the mesh and one for the south—if the filter is not shift-invariant. For the second inter-processor communication step, since $\tilde{Q}_{22}$ must be equal to $\tilde{D}_{R-1}$, the dimension of the matrices follow as

$$\dim\{\tilde{Q}_{22}\} = \dim\{\tilde{D}_{R-1}\} \qquad \dim\{Q_{11}\} \simeq \dim\{\tilde{D}_{R-1}\}$$

These dimensions are illustrated in Figures 2-12 and 2-13. Note that the size of the matrix $\tilde{Q}_{22}$ grows much more rapidly with each inter-processor factorization step than does the size of $\tilde{D}_\rho$ with each local factorization step (the same can be said in the growth of the state X_{in} vs. the growth of the state x_ρ). Hence, the total number of computations is

$$S_4 \simeq S_4^L + \frac{2(8(R-1))^3}{3} + \frac{2(8(3R/4))^3}{3} + \frac{2(8(R/2))^3}{3} = \mathcal{O}(I^4) \qquad (2.57)$$

The total number of computations still grows at the same rate as it does for the single mesh implementation, yet the total number of computations drastically decreases since one only needs to perform four matrix inversions between $\rho = L$ and the global boundary, $\rho = 2L$. When R is large, these savings can be tremendous.

The total number of storage elements required for the 4 mesh algorithm can similarly be shown to be considerably less than in the single processor case, yet the number is still $O(I^3)$.

To see if the savings in storage and computations increases with the number of local meshes, consider a boundary value system with a constant-coefficient difference equation supported on a mesh of size $I \times I$, where $I = 3(4^M)$. The mesh can be partitioned into at most 4^M local regions, where each of the 4^M meshes is a square of size 3-by-3. In this case, there is no local processing to be done, since the filtering constraint gives a relationship between the two outermost states x_0 and x_1 of each local mesh. This implementation is in fact very similar to the nested dissection algorithm given in [14, 13] and discussed in Chapter 3, although there is more fill in the factorization for the algorithm given in this chapter. The parallel implementation in this case consists entirely of inter-processor factorization (combining boundaries) and back-substitution steps. Such a partitioning of the mesh requires M east-west and M north-south inter-processor communication steps before reaching the global boundary. In the first step of the algorithm, 4^M local constraints are combined pairwise (east-west) into $2 * 4^{M-1}$ constraints. The north-south combination step then reduces the the number of constraints to 4^{M-1}. This process continues recursively until the global boundary is reached. Note that, as can be seen from Figures 2-11 through 2-13, the size of the boundary increases approximately by a factor of two with each succession of east-west and north-south combinations. The dimension of the constraints also increases by a factor of two with each two steps. In particular, if before an east-west combination

step X_{in} has dimension L, then after the east-west combination it will have dimension $\frac{3L}{2}$. Yet after the proceeding north-south combination it will have dimension $2L$.

As was the case with the inter-processor steps for the previous example in which the mesh was divided into 4 local regions, the computation requirements of each inter-processor factorization step are dominated by inverting the matrices Q_{11} and $\tilde{Q}_{22}$ found in Equation (2.47). Because the difference equation is constant-coefficient, these coefficient matrices are the same for each processor. Let the dimension of $\tilde{Q}_{22}$ at each east-west step of the algorithm be $n_m \times n_m$ $(m = 1, 2, 3, \ldots, M)$. Then at each north-south factorization step the matrix $\tilde{Q}_{22}$ will have dimension $\frac{3n_m}{2} \times \frac{3n_m}{2}$ $(m = 1, 2, 3, \ldots, M)$. At the boundary, $m = M$ and $\tilde{Q}_{22}$ has dimension proportional to the size of the boundary. Inverting $\tilde{Q}_{22}$ at the boundary will thus take $O(I^3)$ computations. Noting that the dimension of the boundary doubles with each two inter-processor steps, the matrix dimension n_m will also double with each two steps. Hence, the total number of computations needed to invert $\tilde{Q}_{22}$ over all the factorization steps can be approximated as

$$\begin{aligned} S_{fine} &= \sum_{m=0}^{M} (1 + (3/2)^2)\frac{2}{3} n_m^3 \qquad n_m = 1, 2, 4, \ldots, 2^M \\ &= \frac{26}{27}\left(\sum_{m=0}^{M} (2^m)^3\right) \\ &\simeq \sum_{m=0}^{M} 8^m \\ &\simeq \left(\frac{1 - 8^M}{1 - 8}\right) \\ &\simeq 8^M \end{aligned}$$

where $M = \log I$. The number of computations required to invert Q_{11} must also be included in the approximation, yet these computations will be less than double S_{fine}; hence, the total number of computations is $O(I^3)$. Storage requirements are more difficult to analyze, but can similarly be argued to take $O(I^2)$ storage elements. These results compare favorably with nested dissection, which takes as little as $O(I^3)$ computations and $O(I^2 \ln(I))$ storage elements.

These measures of computational complexity reflect the *total* number of computations required for the fine-grain parallel mesh partitioning algorithm when the difference equation is constant-coefficient. One must then ask how the number of computations will change if the

factorization is implemented in parallel and/or the 2-D difference equation is shift-varying.

First consider a parallel implementation of the mesh partitioning algorithm for the constant-coefficient difference equation. For the on-line solution, because the matrices D_ρ, Q_{22}, and Q_{11} are the same for each local mesh, the time to create and factor these matrices will be the same if one processor factors them or if all the local processors factor them independently. The only real increase in performance resulting from a parallel implementation will be in the manipulation of the local input data during the back-substitution steps. For the on-line filter implementation, these computations are a minor contribution to the total number of computations required for the factorization. However, for an off-line implementation, the back-substitutions and similar manipulations of the input data comprise the bulk of the computations, and a parallel implementation will produce sizable savings. For digital filtering applications, in which the matrix factorization can generally be done off-line, the computational savings will be even greater. For such applications, the number of computations is determined by the number of storage elements required for the back-substitution step.

If the difference equation is not constant-coefficient, the computational and storage requirements for a single processor implementation differ from those given for the constant-coefficient filter. For instance, because the matrices D_ρ and Q_{22} differ for each local mesh, a set of coefficient matrices must be stored for each local mesh during the factorization. The number of storage requirements for the factorization of a space-varying filter system is thus significantly more than the $O(I^2)$ required for the constant-coefficient system. Similarly, the total number of computations for the factorization will increase beyond that given for constant-coefficient case due to the fact that the local factorizations and inter-processor communication factorizations must be calculated for each local mesh. However, the performance of the algorithm will increase significantly when implementing the algorithm in parallel. The computational complexity of implementing the shift-varying filter system in parallel is essentially identical to that of the constant-coefficient filter due to the fact that all the local factorizations and inter-processor communication steps can be done in parallel, even though the coefficient matrices differ for each mesh. Note that there are sizable savings in a parallel implementation of both the off-line and the on-line solutions.

The mesh partitioning algorithm has thus been shown to be a vast improvement over the single processor block LU algorithm (obtained by one radial ordering of the mesh).

Chapter 3 details an approximate solution to the mesh partitioning algorithm which can be shown for some cases to yield significant improvements in performance.

Chapter 3

Efficient Algorithms

It was demonstrated in Chapter 2 that the parallel Block LU algorithm is a significant improvement over the algorithm given in Section 2.2. Even when implemented on a single processor, the method of dividing the mesh into local regions, solving for a constraint on the boundary of each of these regions, and then combining the constraints into a constraint on the global boundary is superior to the straightforward algorithm in terms of both computation and storage requirements. However, despite the number of local regions or processors used, the majority of the computations occur during the last few steps of the factorization, when large states of mesh variables comparable in size to the global boundary are "eliminated". In the local processing described in Sections 2.2.1 and 2.3.1, the number of computations is dominated by the construction of the inverse of $\tilde{D}_\rho$ at each step of the factorization in Equation (2.37), and each inter-processor step described in Section 2.3.2 is dominated by inverting $\tilde{Q}_{22}$ in (2.51). For a large number of filters, these full matrices can be approximated by sparse matrices which can be inverted much more efficiently while having negligible or marginal impact on the solution.

For those interested in "exact" numerical solutions to P.D.E.'s or linear systems of equations, the approximate solutions produced by the algorithms developed in this chapter might not be palatable; yet, for many signal processing an estimation applications, one might be willing to make approximations that those solving P.D.E.'s would not. For instance, approximations are inevitably made when transferring from desired frequency response characteristics to the FIR or IIR implementation as a difference equation. It seems reasonable to expect, especially in 2-D, that one might then sacrifice some additional accuracy in the

solution for computational savings.

In this chapter, the *approximate* LU algorithm is developed from the algorithms given in Chapter 2 and can be used to provide highly accurate solutions to 2-D boundary value systems which lead to diagonally dominant systems of equations. The performance of the algorithm on these difference equations is then compared to nested dissection.

3.1 An Approximate LU Algorithm

Given once again the 2-D acausal NNM filter system described in Section 2.2, assume that the mesh variables have been ordered into concentric squares and that the factorization given by the recursion in Equations (2.32)-(2.33) has proceeded for N steps, where $N < R-1$. After N steps, one has the constraint

$$\tilde{D}_N \boldsymbol{x}_N = F_{N+1} \boldsymbol{x}_{N+1} + \Psi_N \tag{3.1}$$

between the states at radii $\rho = N$ and $\rho = (N+1)$. The matrix F_{N+1} is sparse and can be derived directly from the filter coefficients (see Appendix A). The matrix $\tilde{D}_N$ is full (a majority of its elements are non-zero), since eliminating the mesh variables interior to $\boldsymbol{x}_N$ creates fill. This fullness is undesirable, since the next step of the factorization requires that $\tilde{D}_N$ be inverted. When $\boldsymbol{x}_N$ is approximately the size of the global boundary $\boldsymbol{x}_R$ ($R = I/2$), this inversion alone will take $O(I^3)$ operations. Fortunately, for many sets of filter coefficients, $\tilde{D}_N$ can be replaced with a sparse approximation. In fact, for all radii ρ except those near zero, D_ρ can be replaced with a sparse approximation. Thus the approximate algorithm involves replacing the full matrix $\tilde{D}_\rho$ at each step of the factorization for $\rho \geq N$ with a sparse and easily invertible approximation.

The coupling given by Equation 3.1 between the states $\boldsymbol{x}_N$ and $\boldsymbol{x}_{N+1}$ is illustrated in Figure 3-1. The variables contained in $\boldsymbol{x}_N$ are connected in the figure with a solid-line square, whereas those in $\boldsymbol{x}_{N+1}$ are connected with a dotted-line square. The arrows extending from the inner to the outer square in the figure show which variables in $\boldsymbol{x}_N$ are coupled through F_{N+1} with those in $\boldsymbol{x}_{N+1}$. Because $\tilde{D}_N$ is full, however, every variable in $\boldsymbol{x}_N$ is directly coupled with every other variable in $\boldsymbol{x}_N$; yet, for most filters, one would expect that each variable $x[i,j]$ in $\boldsymbol{x}_N$ is coupled most strongly to the other variables in $\boldsymbol{x}_N$ which are geographically closest to $x[i,j]$. In fact, in some cases, the coupling decays

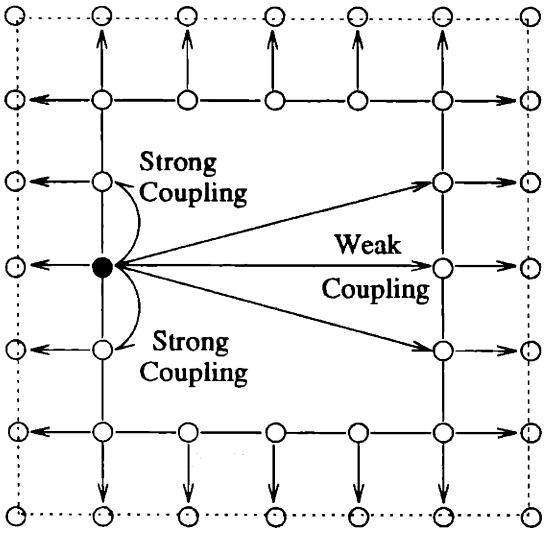

Figure 3-1: Structure of the constraint between x_ρ and $x_{\rho+1}$ after each factorization step of the Block LU Algorithm.

extremely fast with the distance between variables in the mesh. This decay is illustrated in Figure 3-1, where the variable represented by the shaded circle is strongly coupled with its nearest neighbors in x_N, yet is only very weakly coupled to those variables farther away. As N increases, so will the maximum distance from one variable to another in x_N, and the coefficients in $\tilde{D}_N$ will decrease accordingly.

The approximate algorithm treats those coefficients in $\tilde{D}_N$ which represent couplings between variables separated by more than a "small distance" in the mesh as zero. One can think of this approximation as modeling $\tilde{D}_N x_N$ in (3.1) as a low-order 1-D IIR filter around x_N. After the approximation, $\tilde{D}_N$ is sparse and its inverse can be found much more quickly. Furthermore, if we make a similar approximation at each step of the factorization where $\rho \geq N$, the total number of computations should drop dramatically. Before discussing the implementation and accuracy of the approximate algorithm, however, we need to first decide how to make the approximation (i.e. if only those coefficients corresponding to coupling between variables separated by less than a small distance are kept, what distance metric is used?). In this work, the approximation to $\tilde{D}_N$ was developed based upon heuristic ideas and empirical evidence, and its structure can be best illustrated by example.

A class of filters for which the approximation algorithm works well results from a dis-

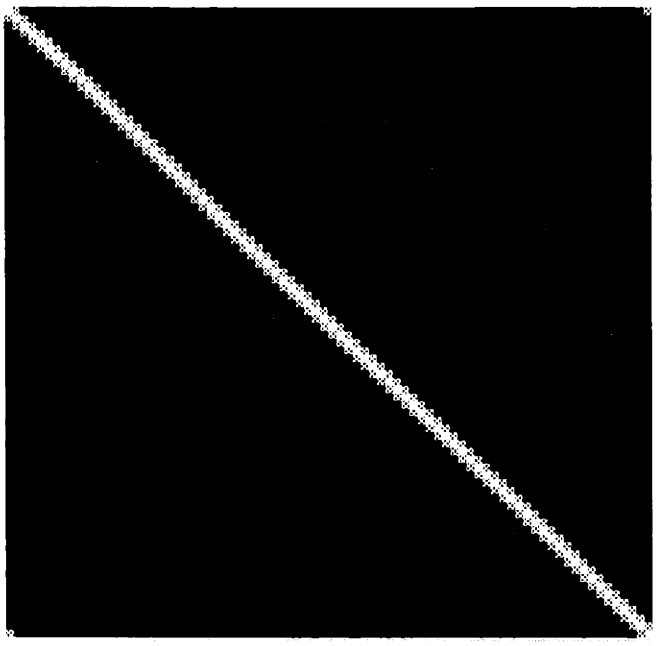

Figure 3-2: Grey scale magnitude plot of the elements in $\tilde{D}_{10}$ for an NNM with $n=s=e=w=.15$. The magnitude of the diagonal elements relative to the elements just of the diagonal is: $\tilde{D}_{10}(i, i \pm 1) \approx \tilde{D}_{10}(i, i) * (-.154)$.

cretization of the following PDE

$$\nabla^2 x(q, v) + kx(q, v) = u(q, v) \tag{3.2}$$

where q and v are continuous spatial coordinates. Equation (3.2) will be referred to in this chapter as a damped Poisson, although it is better known as the Helmholtz equation when $k > 0$. Discretizing (3.2) by finite-difference methods ($q = h_q i, \quad v = h_v j$), one obtains an NNM difference equation (2.2). The difference equations which result from a discretization of Helmholtz's equation are low-pass filters, yet the approximation works well for other classes of frequency-selective filters, some of which are analyzed in Section 3.3.

As k increases from zero, the overall filter system $Ax = b$ becomes more *diagonally dominant* and the approximation's accuracy increases proportionately to k. (A matrix is diagonally dominant if the sum of the absolute values of the off-diagonal elements for every row and column is less than or equal to the absolute value of the element along the diagonal. If the inequality is strict, the matrix is strictly diagonally dominant.) For instance, when $k = 0$ and $h_q = h_v$, one obtains an NNM difference equation corresponding to a discrete

Poisson ($n = s = e = w = 1/4$). In this case the overall system is diagonally dominant (not strictly diagonally dominant), but significant errors are introduced by the approximation. However, as will be shown in Section 3.3, when k increases slightly such that (3.2) discretizes to an NNM filter with the coefficients $n = s = e = w = .15$, a simple approximation to $\tilde{D}_N$ can be found which has minimal effects on the accuracy of the solution. For this difference equation, a grey-scale plot of the magnitude of the elements in $\tilde{D}_N$ for $N = 10$ is given in Figure 3-2. The matrix $\tilde{D}_{10}$ has dimension 80-by-80 (see Appendix A), and note that the variables in x_N have been ordered counter-clockwise, as is illustrated in Figure A-1. From Figure 3-2, one can see that the off-diagonal elements decay quickly to zero with distance from the diagonal. The "large" values in the upper-right and lower-left corners of $\tilde{D}_{10}$ represent the fact that the first element in x_{10} is physically located next to the last element in x_{10} (see Figure A-1), hence, they are strongly coupled. In other words, because x_N is a closed square in the mesh, the matrix $\tilde{D}_N$ has circular structure, and one would expect the approximation to also be circular.

From Figure 3-2, one can see that a natural approximation to the full matrix $\tilde{D}_{10}$ is to zero those elements which are more than a distance of one from the main diagonal or from the upper-right and lower-left corners of the matrix. The accuracy of the approximation could be increased by setting to zero those elements which are more than a distance of B from the diagonal or from the upper-right and lower-left corners of the matrix. Call B the bandwidth of the approximation.

As can be found in Appendix A, the matrix $\tilde{D}_N$ has dimension $8N$-by-$8N$, so its approximation D_N^{apx} of bandwidth B can be derived from the following relation

$$D_N^{apx}(i,j) = \left\{ \begin{array}{cc} \tilde{D}_N(i,j), & |i-j| \leq B \text{ or } |i-j| > 8N-B \\ 0, & \text{otherwise} \end{array} \right\} \tag{3.3}$$

To illustrate the structure of the approximation matrix, take the nearly full 32-by-32 matrix $\tilde{D}_4$, which is produced after four steps in the LU factorization. Its approximation D_4^{apx} has the structure shown in Figure 3-3 for $B = 1$ and that shown in Figure 3-4 for $B = 3$. (Note that there are zeros within the bandwidth B of the main diagonal for D_4^{apx} in Figure 3-4. These elements were zero in $\tilde{D}_4$ before the approximation was made.)

Given the non-zero sparse structures shown in Figures 3-3 and 3-4, inverting D_N^{apx} will take much less than $O(I^3)$ operations. Using the formula given in [13] for determining the

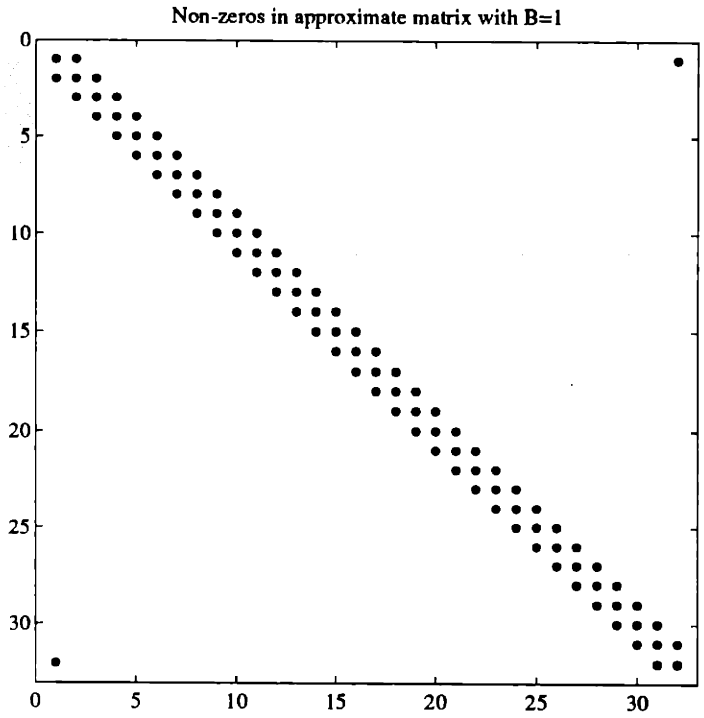

Figure 3-3: Location of non-zero elements in approximate matrix with $B = 1$.

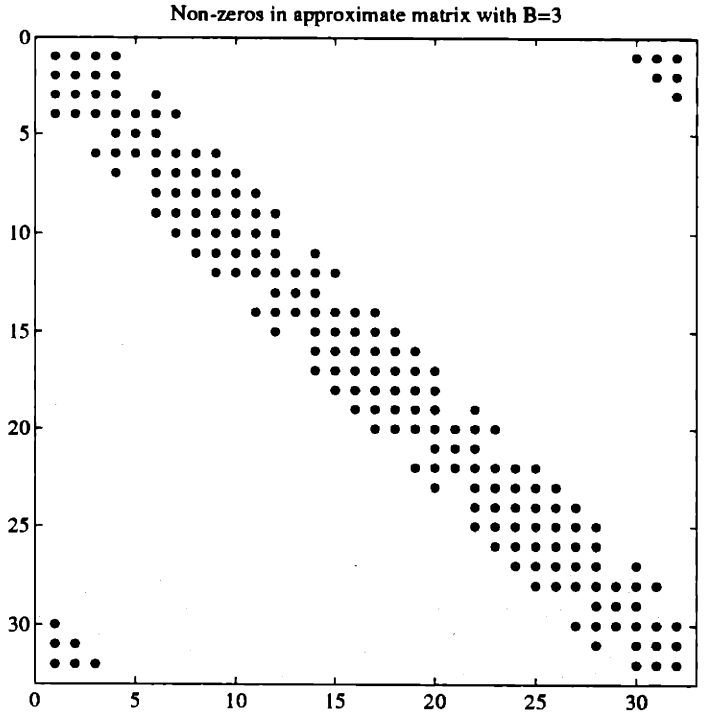

Figure 3-4: Location of non-zero elements in approximate matrix with $B = 3$.

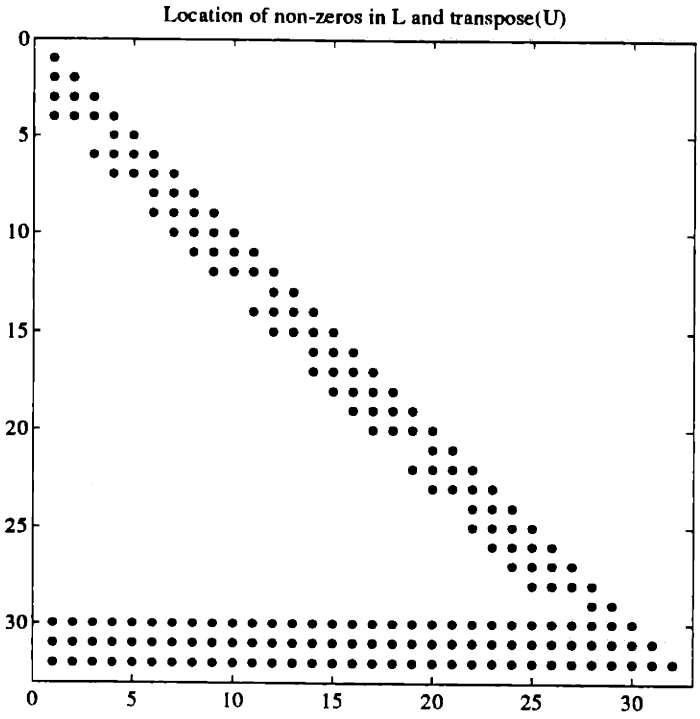

Figure 3-5: Location of non-zero elements in L and U^T of LU factorization of approximate matrix with $B = 3$.

number of operations required to factor a sparse matrix while not operating on the zero elements, the number of operations required to factor D_N^{apx} is

$$\theta = (N-1)(2B(2B+3)) \tag{3.4}$$

which asymptotically reduces to $O(B^2N)$ computations. Such matrices can also be factored very efficiently in parallel with a variant of cyclic block reduction given in [16], or the inward/outward processes given in [33]. For $B = 3$, the non-zero structure of L and U^T in a straightforward LU factorization of D_4^{apx} is given in Figure 3-5. One can see that fill from even the simplest factorization occurs in only the last B rows of D_4^{apx}. In fact, for any N, the fill from factoring D_N^{apx} will only occur in the last B rows of L and U^T. Therefore, the factors L and U of D_N^{apx} will become more sparse (a greater percentage of the matrix's elements will be zero) as N increases. When B is small, inverting D_ρ^{apx} ($\rho \geq N$) will essentially take $O(\rho)$ operations.

Although using the approximation given by Equation 3.3 at each step of the LU factorization does not significantly perturb the solution to a number of filter systems, even when B is equal to one, there exists a modification which increases the accuracy of the

approximate algorithm by an order of magnitude without significantly increasing the computational requirements. In other words, there is still a significant amount of information in the "dark" area of the plot in Figure 3-2. The approximation can be improved by finding which coefficients in the dark area significantly affect the accuracy of the solution.

The approximation given by Equation 3.3 keeps those coefficients in $\tilde{D}_N$ which represent a coupling between variables in x_N separated by a distance less than or equal to B, where distance in this case is the minimum number of mesh points along the square x_N that one must travel to get from one variable to another. However, the decay of the coefficients $\tilde{D}_N$ also depends upon the 2-D distance between points in x_N. Variables in the "corners" of x_N are closer to more points in x_N than those which lie in the middle of the edges of the square; hence, variables at the corner are tightly coupled to more variables in x_N. The approximation is improved by adding more coefficients around the diagonals of $\tilde{D}_N$ at indices which correspond to corners in x_N. One such improvement to (3.3) which will be shown in the next section to work well is given by

$$D_N^{apx2} = D_N^{apx} + D_N^{improve} \tag{3.5}$$

where $D_N^{improve}(i,j) = \tilde{D}_N(i,j)$ if and only if (i,j) falls into one of the following "squares" of indices:

$$\begin{aligned}(i,j) \in\ & [c_1-(B+1), c_1+(B+1)] \times [c_1-(B+1), c_1+(B+1)] \\ & [c_2-(B+1), c_2+(B+1)] \times [c_2-(B+1), c_2+(B+1)] \\ & [c_3-(B+1), c_3+(B+1)] \times [c_3-(B+1), c_3+(B+1)] \\ & [c_4-(B+1), c_4+(B+1)] \times [c_4-(B+1), c_4+(B+1)]\end{aligned}$$

where c_1, c_2, c_3, and c_4 are the indices of variables of the four corners in x_N, and D_N^{apx} is given by (3.5); otherwise $D_N^{improve}(i,j)$ is zero. The indices of the four corners in x_N resulting from the counter-clockwise ordering given in Appendix A are

$$\begin{aligned}c_1 &= N+1 \\ c_2 &= 3N+1 \\ c_3 &= 5N+1\end{aligned}$$

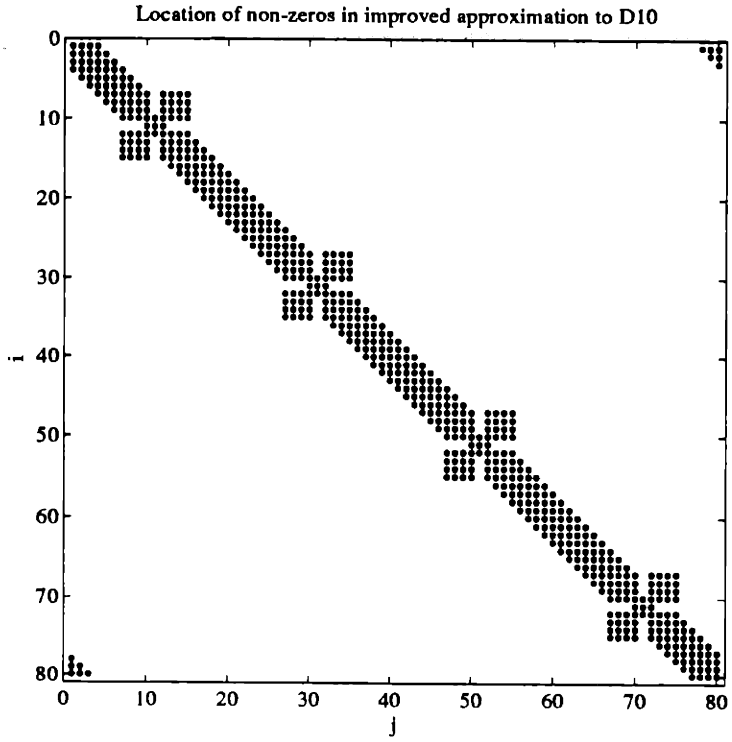

Figure 3-6: Location of non-zero elements in approximate matrix with $B = 3$, including extra coefficients near the corners of x_N.

$$c_4 = 7N+1$$

$$x_N \in \Re^{8N\times 1}$$

As an example, approximating the elements in $\tilde{D}_N$ with Equation (3.5), the structure of the non-zero elements in D_N^{apx2} for $N = 10$ is shown in Figure 3-6. In this figure one can see the squares of non-zeros centered about $(i,j) = (c_1,c_1)$, (c_2,c_2), (c_3,c_3), and (c_4,c_4). These squares give improved accuracy by extending the coefficients in the approximation which constrain elements at the corners of x_{10}. Note that the number of additional non-zero elements in D_N^{apx2} attributable to $D_N^{improve}$ is independent of N, meaning the inversion of D_N^{apx2} will still require $O(B^2N)$ computations.

3.1.1 Approximating the Inter-processor Communication Step

Fortunately, an identical approximation exists for each inter-processor communication step detailed in Section 2.3.2. The majority of the computations at each step of the factorization given by Equations (2.39)-(2.47), excluding reordering and inter-processor communication costs, is dominated by the computation of $\tilde{Q}_{22}$ in Equation (2.46). Furthermore, the inverse of both $\tilde{Q}_{22}$ and Q_{11} must be computed for the back-substitution steps (2.51) and (2.52).

These operations will be computationally expensive, since both Q_{22} and Q_{11} in the left-hand side of Equation (2.44) are full and have very large dimension. Equation (2.44), however, is just a permutation of Equation (2.41). Without using the approximation given by (3.5) in the previous section, all the block elements of Z in (2.41) are sparse, except for $\tilde{D}_{L-1}$. But as we have just seen, there are filters for which the matrix $\tilde{D}_{L-1}$ can be approximated accurately by the sparse matrix D_{L-1}^{apx2} when $L-1 \geq N$ (we are assuming that no approximation is made for the first few steps of the LU factorization, since the matrices have small dimensions). After approximating $\tilde{D}_{L-1}$, all the block elements of Z are now sparse and, hence, so are the block elements Q_{22} and Q_{11} in the permutation of Z. The matrix $\tilde{Q}_{22}$ in (2.46) can then be computed efficiently. The last remaining bottleneck for the inter-processor communication step is that $\tilde{Q}_{22}$ will in general be full. This fullness results from the fact that the inverse of a sparse matrix, which in this case is Q_{11}^{-1} in Equation (2.46), is generally full.

To compute the inverse of $\tilde{Q}_{22}$ in (2.51) while still maintaining an efficient algorithm, an approximation to $\tilde{Q}_{22}$ follows by noting that the last two block rows of Equation (2.47)

$$\tilde{Q}_{22}X_{in} = F_{out}X_{out} + \tilde{\Psi}_{in} \tag{3.6}$$

are identical in form and function to the constraint given by (3.1). Like the matrix F_{N+1} in (3.1), the matrix F_{out} is very sparse and can be generated from the filter coefficients (see Appendix A). More importantly, the full matrix $\tilde{Q}_{22}$ has a sparse structure identical to that of $\tilde{D}_{\rho}$, where the coupling between variables in X_{in} decreases with the 2-D distance between the variables.

The elements X_{in} and X_{out} for the first inter-processor communication step are illustrated in Figure 2-12, where X_{in} contains the mesh elements in the inner dark rectangle and X_{out} contains those in the outer rectangle. Furthermore, the permutation P_1 in (2.42) is defined such that the mesh variables in X_{in} and X_{out} are ordered counter-clockwise in the mesh. Since for most filters the coefficients in $\tilde{Q}_{22}$ decrease rapidly in magnitude with distance from the diagonal and upper-right and lower-left corners of the matrix, the problem of approximating $\tilde{Q}_{22}$ is identical to that of approximating $\tilde{D}_N$ in (3.1).

Consider again the NNM difference equation with $n = s = e = w = .15$, where the local radius L is defined to be 10. A grey-scale plot of the magnitude of the elements in $\tilde{Q}_{22}$ for the

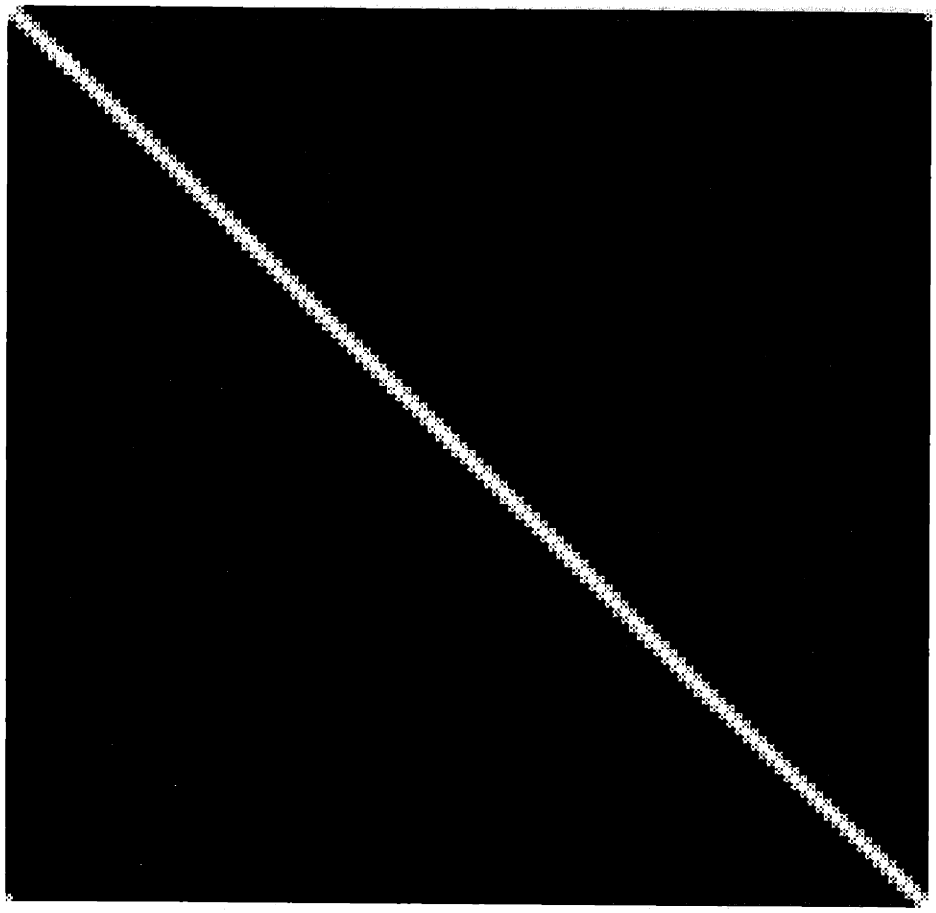

Figure 3-7: Grey scale magnitude plot of the elements in $\tilde{Q}_{22}$ with $L = 10$ for an NNM with $n = s = e = w = .15$.

first inter-processor step is given Figure 3-7. As with $\tilde{D}_\rho$, the most natural approximation to $\tilde{Q}_{22}$ is given by (3.3), yet again one must correct for the increased coupling at the corners of the state X_{in}. The only difference, then, in approximating $\tilde{Q}_{22}$ in lieu of $\tilde{D}_\rho$ is that the state X_{in} is rectangular and not square; hence, the indices of the corners in (3.5) differ from those given for x_N. The non-zero structure of the approximation to the matrix $\tilde{Q}_{22}$ is shown in Figure 3-8. The fact that X_{in} is rectangular and not square is evidenced in Figure 3-8, where the "corners" of X_{in} are not evenly spaced as they are in Figure 3-6.

In order to state the approximation of $\tilde{Q}_{22}$ more formally, one has to define an ordering of the mesh variables in X_{in}, which in turn determines the structure of the equations in $\tilde{Q}_{22}$. However, whether the elements in X_{in} and X_{out} are ordered clockwise or counter-clockwise is arbitrary, as is the determination of which mesh variable in each state should be indexed first. For this chapter, assume that the permutation P_1 for the first inter-processor

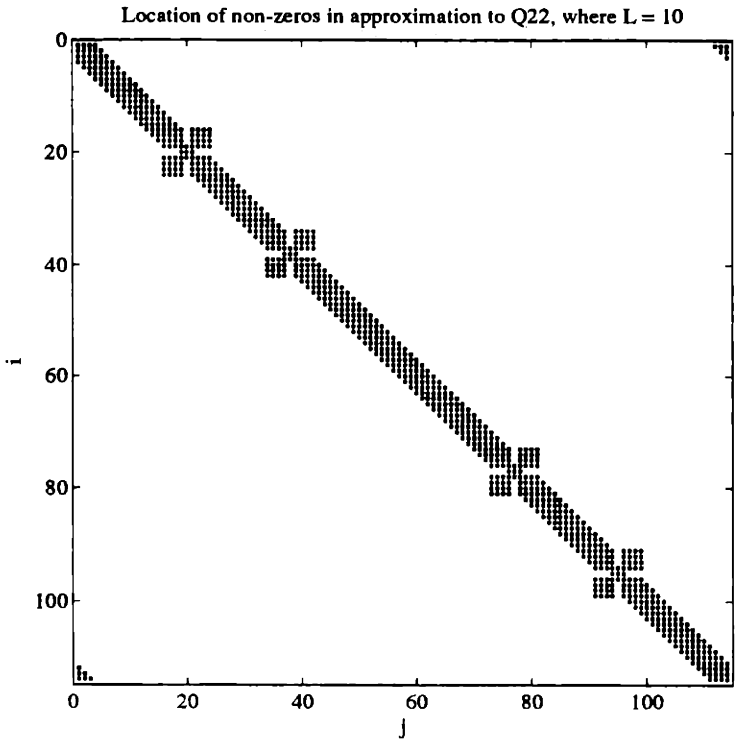

Figure 3-8: Location of non-zero elements in matrix approximating $\tilde{Q}_{22}$, with $B = 3$ and $L = 10$.

communication step is defined such that the first element in X_{in} is the centermost element (geographically) which lies on the northern edge of X_{in} in Figure 2-12 (actually, there are two elements in the center of each edge, so we can choose the element lying in the west half of X_{in}). Relabeling the variables X_{in} and X_{out} from the first inter-processor communication step as X_{in}^1 and X_{out}^1, the indices of the corner elements in X_{in}^1 follow as

$$\begin{aligned} c_1 &= 2L \\ c_2 &= 4L-2 \\ c_3 &= 8L-3 \\ c_4 &= 10L-5 \\ X_{in}^1 &\in \Re^{6(2L-1)\times 1} \end{aligned}$$

For the second inter-processor communication step, which is illustrated in Figure 2-13, the first element of X_{in}^1 is also a member of the interior shaded square of Figure 2-13. Call the interior square of mesh variables X_{in}^2 and the outer square X_{out}^2, which for the four processor example in Section 2.3 is the global boundary. Note that unlike X_{in}^1 from the

first inter-processor factorization step, X_{in}^2 is square. Choosing the first variable in X_{in}^1 to be the first variable in X_{in}^2, and again ordering the rest of the variables counter-clockwise, the corner indices follow as

$$\begin{aligned} c_1 &= 2L \\ c_2 &= 6L-1 \\ c_3 &= 10L-2 \\ c_4 &= 14L-3 \\ X_{in}^2 &\in \Re^{(16L-4)\times 1} \end{aligned}$$

Let X_{in}^i be the interior rectangular state of mesh variables generated by the i-th inter-processor communication step. When $i > 2$, the corner indices of X_{in}^i can be found similarly. The approximation Q_{22}^{apx} to the matrix $\tilde{Q}_{22}$ generated by each inter-processor communication step then follows from (3.3) and (3.5) as $Q_{22}^{apx}(i,j) = \tilde{Q}_{22}(i,j)$ if and only if one of the following is satisfied:

$$\begin{aligned} |i-j| &\leq B \\ |i-j| &> \dim\{X_{in}\} - B \\ (i,j) &\in [c_1-(B+1), c_1+(B+1)] \times [c_1-(B+1), c_1+(B+1)] \\ &\in [c_2-(B+1), c_2+(B+1)] \times [c_2-(B+1), c_2+(B+1)] \\ &\in [c_3-(B+1), c_3+(B+1)] \times [c_3-(B+1), c_3+(B+1)] \\ &\in [c_4-(B+1), c_4+(B+1)] \times [c_4-(B+1), c_4+(B+1)] \end{aligned}$$

otherwise $Q_{22}^{apx}(i,j) = 0$. If more than four processors are used, east/west and north/south combinations of constraints follow recursively until the global boundary is reached, so the corner indices can be easily derived from those given above by just redefining L.

After this somewhat laborious discussion of how to approximate the matrices $\tilde{Q}_{22}$ and $\tilde{D}_\rho$ in the block LU algorithm, we now turn to the performance of the algorithm. The next section discusses possible implementations of the approximate algorithm and gives some estimates of the computational and storage requirements, which are shown to depend

heavily upon the implementation.

3.1.2 Implementing the Approximate LU Algorithm

Consider again the NNM filter given in Section 2.1 defined on a square mesh. Also assume, without loss of generality, that the mesh can be divided evenly into 4^M square sub-regions of local radius L. After ordering the local mesh variables and filter dynamics radially into the form of Equations (2.21)-(2.23), the parallel block LU algorithm with approximations follows as:

- **Local Factorizations** – Perform the following recursive procedure on each of the 4^M sets of local data for $\rho = 1, 2, \ldots, L-1$, initializing the recursions with $\Psi_0 = H_0 u_0$ and $\tilde{D}_0 = 1$:

  $$\Psi_\rho = G_{\rho-1}\tilde{D}_{\rho-1}^{-1}\Psi_{\rho-1} + H_\rho u_\rho \quad \text{[Equation (2.36)]}$$

  If $\rho \geq N$, replace $\tilde{D}_\rho$ with D_ρ^{apx2} given by Equation (3.5).

  $$\tilde{D}_\rho = D_\rho - G_{\rho-1}\tilde{D}_{\rho-1}^{-1}F_\rho \quad \text{[Equations (2.27)-(2.28)]}$$

  (where the matrices used in each of the 4^M processors are the same, assuming a constant coefficient difference equation, although, of course, each processor has its own set of local data (input data) in Equation (2.36))

- **Inter-processor Factorizations** – Given the constraints produced on the boundaries of the local meshes, it takes M east-west and M north-south inter-processor factorization steps to form a constraint on the global boundary. The first inter-processor step is detailed in Section 2.3.2, yet a general form is not given. Because the form of the equations does not change from those given in Section 2.3.2 for inter-processor steps after the first, the superscript j is added to denote the states and coefficients at step j. Without detailing the permutations (reordering of the equations) involved in between inter-processor communication steps, the following recursive procedures are performed for $i = 1, \ldots, M$:

  $$j = (2i - 1)$$

  As is illustrated in Figure 2-12, 4^{M-i+1} constraints are combined pairwise east/west with the unused constraints which link each pair of constraints to yield $2 * 4^{M-i}$ sets

of equations of the form

$$\begin{bmatrix} Q^j_{11} & Q^j_{12} & 0 \\ Q^j_{21} & Q^j_{22} & -F^j_{out} \end{bmatrix} \begin{bmatrix} X^j_{elim} \\ X^j_{in} \\ X^j_{out} \end{bmatrix} = \begin{bmatrix} \Psi^j_{elim} \\ \Psi^j_{in} \end{bmatrix} \quad \text{[Equation (2.44)]}$$

One then computes:

$$\tilde{\Psi}^j_{in} = \Psi^j_{in} - Q^j_{21}(Q^j_{11})^{-1}\Psi^j_{elim} \quad \text{[Equation (2.48)]}$$

$$\tilde{Q}^j_{22} = Q^j_{22} - Q^j_{21}(Q^j_{11})^{-1}Q^j_{12} \quad \text{[Equation (2.46)]}$$

Replace $\tilde{Q}^j_{22}$ with its approximation $(Q^j_{22})^{apx}$. One then has $2 * 4^{M-i}$ constraints

$$\tilde{Q}^j_{22}X^j_{in} = F^j_{out}X^j_{out} + \tilde{\Psi}^j_{in}$$

which are to be combined north-south pairwise with the unused constraints which link the states in each pair as in Figure 2-13 to yield 4^{M-i} constraints of the form

$$\tilde{Q}^{j+1}_{22}X^{j+1}_{in} = F^{j+1}_{out}X^{j+1}_{out} + \tilde{\Psi}^{j+1}_{in} \tag{3.7}$$

If $i = M$, then Equation (3.7) is equivalent to (2.34).

- **Inter-processor Back-Substitution** – The recursion follows for $i = M, \ldots, 2, 1$ and is initialized by the Dirichlet conditions which specify $x_R = X^{2M}_{out}$:

$$j = (2i - 1)$$

North-south back-substitution

$$X^{j+1}_{in} = (\tilde{Q}^{j+1}_{22})^{-1}(F^{j+1}_{out}X^{j+1}_{out} + \tilde{\Psi}^{j+1}_{in}) \quad \text{[Equation (2.51)]}$$

$$X^{j+1}_{elim} = (Q^{j+1}_{11})^{-1}(-Q^{j+1}_{12}X^{j+1}_{in} + \tilde{\Psi}^{j+1}_{elim}) \quad \text{[Equation (2.52)]}$$

The east-west back-substitution step has the identical form

$$X^j_{in} = (\tilde{Q}^j_{22})^{-1}(F^j_{out}X^j_{out} + \tilde{\Psi}^j_{in}) \quad \text{[Equation (2.51)]}$$

$$X^j_{elim} = (Q^j_{11})^{-1}(-Q^j_{12}X^j_{in} + \tilde{\Psi}^j_{elim}) \quad \text{[Equation (2.52)]}$$

- **Local Back-Substitution** – For each of the M sets of local data, the solution is found for $\rho = L-1, ..., 2, 1$ with the following recursion, where x_L is given after some rearrangement of the variables by the last inter-processor back-substitution step:

$$x_\rho = \tilde{D}^{-1}_\rho(F_{\rho+1}x_{\rho+1} + \Psi_\rho) \quad \text{[Equation (2.37)]}$$

The storage and computational requirements of this algorithm depend upon a variety of parameters, such as the bandwidth of the approximation and how the coefficient matrices are stored for the back-substitution step. Note that the back-substitution steps given in Equations (2.51), (2.52), and (2.37) require the matrices $[(\tilde{Q}^j_{22})^{-1}F^j_{out}]$, $[(Q^j_{11})^{-1}Q^j_{12}]$, and $(\tilde{D}^{-1}_\rho F_{\rho+1})$, respectively, at each step. Both $(\tilde{D}^{-1}_\rho F_{\rho+1})$ and $[(Q^j_{11})^{-1}Q^j_{12}]$ are computed during each local and inter-processor factorization step in Equations (2.27) and (2.46), respectively, and thus can be stored during the factorization. However, the matrix $[(\tilde{Q}^j_{22})^{-1}F^j_{out}]$ is not computed during the factorization steps. One only needs to store the sparse approximation to $\tilde{Q}^j_{22}$, since F^j_{out} can be generated from the filter parameters during the back-substitution. The algorithm's storage requirements are then dominated by saving the dense matrices $[(Q^j_{11})^{-1}Q^j_{12}]$ and $(\tilde{D}^{-1}_\rho F_{\rho+1})$ after each factorization step.

A minor question regarding implementing the algorithm is then whether to store the dense matrices $[(Q^j_{11})^{-1}Q^j_{12}]$ and $(\tilde{D}^{-1}_\rho F_{\rho+1})$ during the factorization steps, or to save the sparse matrices $\tilde{D}_\rho$ (after it is approximated) and Q^j_{11} and recompute the relevant matrices during the back-substitution step. The former method is obviously more computationally efficient and is the method used for all the examples in Section 3.3, yet the latter method might be preferred when storage is a serious concern. The storage requirements demanded by the former implementation can become particularly acute when the filter is space-varying.

If storing the large, dense matrices is a serious bottleneck, one can think of two alternative implementations which reduce the memory requirements; one implementation sacrifices computational efficiency, and the other sacrifices accuracy. The first alternative implementation has already been discussed, in which the approximation to $\tilde{D}_\rho$ given by (2.27)-(2.28) is stored after each factorization step, in lieu of storing $(\tilde{D}^{-1}_\rho F_{\rho+1})$. Also, after each inter-processor communication step, one stores Q^j_{11} instead of $(Q^j_{11})^{-1}Q^j_{12}$. The added

computational requirements of this algorithm are alleviated somewhat by a second alternative implementation. For this implementation we note that if an approximation exists for $\tilde{D}_\rho$, a similar approximation should exist for $(\tilde{D}_\rho^{-1} F_{\rho+1})$. Yet, because the most efficient implementations use a large number of partitions, meaning that $(\tilde{D}_\rho^{-1} F_{\rho+1})$ will only need to be computed for a small number of steps, this additional approximation to $(\tilde{D}_\rho^{-1} F_{\rho+1})$ is unlikely to be worth pursuing.

Dropping implementation intricacies related to storage, it should now be apparent from the discussion in this section that the approximate algorithm offers significant improvements in storage and computation requirements over the Block LU algorithm, given any degree of parallelism. Exact performance measures are beyond the scope of this thesis, since there are a number of variables which must be considered (e.g. the degree of parallelism, the bandwidth of the approximation, in addition to the size of the 2-D input). One can however, make some significant observations about the total number of computations required by the approximate algorithm.

For the algorithms given in Chapter 2, the computational requirements are dominated by producing the inverse to $\tilde{D}_\rho$ at each local factorization step and producing the inverse to Q_{11} and $\tilde{Q}_{22}$ at each inter-processor back-substitution step. Without any approximations, these matrices are dense. After approximating $\tilde{D}_\rho$ and $\tilde{Q}_{22}$ with the algorithms given in this section, however, all the inversions involve sparse, structured matrices. For any of these matrices which has dimension n-by-n, the number of computations required to compute its inversion reduces from $O(n^3)$ to $O(B^2 n)$, after an approximation of bandwidth B. Consider now the example given in Section 2.3.3 in which only local factorizations are performed. This implementation was shown to require $O(I^4)$ computations, yet with the approximations the number of computations reduces to

$$\sum_{\rho=0}^{R} B^2 n_\rho = \mathcal{O}(B^2 I^2) \tag{3.8}$$

In other words, for a mesh of size I-by-I and a small approximation bandwidth, the approximate block LU factorization requires essentially $\mathcal{O}(I^2)$ computations, which is an order of magnitude less than that required by nested dissection. With the approximation, the back-substitution steps and the sparse matrix multiplications needed to compute $\tilde{D}_\rho$ now comprise a significant amount of the total computations for both the on-line and off-line

implementations.

For implementations in which the mesh is partitioned, the computational savings due to the approximations are more difficult to quantify. Consider the finest-grain partitioning which involves only inter-processor communication steps. Without approximations, the factorization is dominated by the computation of the factorization of $\tilde{Q}_{22}$ and Q_{11} at each inter-processor step. For each of the matrices, these computations were shown in Section 2.3.3 to require

$$\begin{aligned} S_{comp} &= \sum_{m=0}^{M} (1 + (3/2)^2)\frac{2}{3} n_m^3 \qquad n_m = 1, 2, 4, \ldots, 2^M \\ &\simeq 8^M \\ &\simeq \mathcal{O}(I^3) \end{aligned}$$

computations. With approximations, the number of computations required for inverting $\tilde{Q}_{22}$ at each step of the algorithm becomes

$$\begin{aligned} S_{comp} &= \sum_{m=0}^{M} (1 + (3/2)^2) B^2 n_m \qquad n_m = 1, 2, 4, \ldots, 2^M \\ &\simeq B^2 2^M \\ &\simeq \mathcal{O}(B^2 I) \end{aligned}$$

which is a significant improvement. However, the sparse matrix Q_{11} does not have the simple banded structure of $\tilde{Q}_{22}$; thus the complexity of its factorization is difficult to measure. After the approximation, the matrix $\tilde{Q}_{22}$ is banded because it represents the approximate coupling among the variables in the inner rectangle of Figure 2-12. The matrix Q_{11} represents the coupling among the variables represented by hollow circles in Figure 2-12, which are to be eliminated by the inter-processor communication step. This set of variables, however, has thickness, suggesting that a tailored ordering be used for Q_{11} similar to that developed for $\tilde{Q}_{22}$ and $\tilde{D}_\rho$. For large mesh sizes, Q_{11} will essentially be banded about the diagonal after a few inter-processor communication steps. In any case, the approximation yields significant computation savings for the fine-grain partitioning case.

The next section discusses nested dissection, a well-known direct algorithm for solving large, sparse systems of equations, such as those which result from discretizing elliptic P.D.E.'s by finite difference methods. The performance of the algorithms developed in this

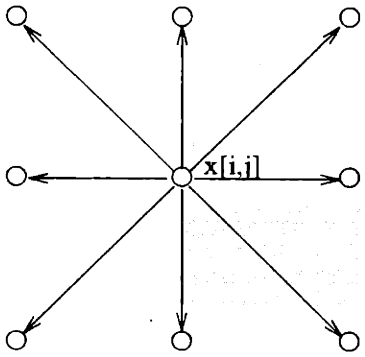

Figure 3-9: Dependence structure of a difference equation leading to linear system of equations to be solved by the nested dissection algorithm.

thesis is then compared to that of nested dissection in Section 3.3.

3.2 Nested Dissection

Since the development of *nested dissection* in [13] in 1973, the algorithm has been known as the most elegant and efficient (asymptotically) direct solution to matrix problems arising from finite difference applications, where the system of equations $Ax = b$ is such that the matrix A has symmetric structure with a localized dependence graph. One example might be the set linear system which results from a 2-D IIR filter described by a low-order symmetrically structured difference equation. Because nested dissection is clearly detailed and simulated in [14, 13], the objective of this section is foremost to give a brief synopsis of the algorithm and to contrast is with the algorithms given in Chapter 2 and Section 3.1. We also hope to be suggesting how nested dissection, which has traditionally been used to solve P.D.E.'s, allows for efficient implementations of 2-D IIR filters described by difference equations and boundary conditions. Some low-order filter applications are given in Section 3.3, but the larger question of whether or not one can design meaningful 2-D IIR filters with boundary conditions is pondered in the conclusions.

It is shown in [14, 13] that the factorization of any matrix which is the result a boundary value system with a second-order 2-D difference equation, such as the NNM or that given by the dependence graph in Figure 3-9, applied to a domain of size N-by-N must take *at least* $O(N^3)$ computations and $O(N^2 \log_2 N)$ storage elements. Nested dissection is optimal in that it asymptotically matches these lower bounds, and no other direct algorithm will

do asymptotically better, unless one is willing to make approximations. For some systems, the approximations given in Section 3.1 allow one to beat these bounds while bearing small errors in the solution.

In obtaining these lower bounds, nested dissection orders the elements in A such that the amount of fill which occurs during its LU factorization is minimal. It has been mentioned throughout this thesis that the amount of fill which occurs during a matrix's factorization to a large extent determines the number of storage elements and computations required to perform the factorization. Minimizing the amount of fill is accomplished with the use of separators (dissectors), which dissect the mesh into disjoint sets (disjoint in the sense that the difference equation does not couple variables in one set with those in the other). For a difference equation with the dependence shown in Figure 3-9 applied to a 2-D region with 9-by-9 elements, a separator (dissector) is illustrated in Figure 3-10. The dissector, S_1, in effect disconnects the variables in C_1 from those in C_2. Namely, if the factorization is performed such that the variables in S_1 are ordered last, the matrix A takes the form

$$A = \begin{bmatrix} A_1 & 0 & Z_1 \\ 0 & A_2 & Z_2 \\ W_1 & W_2 & A_3 \end{bmatrix} \tag{3.9}$$

where the matrices A_1, A_2, and A_3 represent coupling between variables in the same set. The significance of the dissector is that if A in (3.9) is factored, the block zeros will be preserved during the factorization (i.e. no fill will occur in these locations).

Since sets C_1 and C_2 can be viewed as meshes by themselves, the dissection can continue recursively by placing separators in the middle of both C_1 and C_2, as is illustrated in Figure 3-11. If variables in the the separators S_2 and S_3 are ordered last in sets C_1 and C_2, respectively, both A_1 and A_2 will also have the form of A in (3.9). One can see that for any domain size, the elements in A can be ordered by recursively dissecting the mesh until no more dissections are feasible. The matrix A then takes a fractal structure like that given in Figure 3-12. The nested dissection ordering of the 10-by-10 element mesh used to order A in Figure 3-12 is given in Table 3.1.

Because the block zeros created by each separator are preserved during the factorization, it is shown in [14, 13] that the nested dissection ordering of A requires $O(N^3)$ computations and $O(N^2 \log_2 N)$ storage elements to factor.One might now ask how the nested dissection

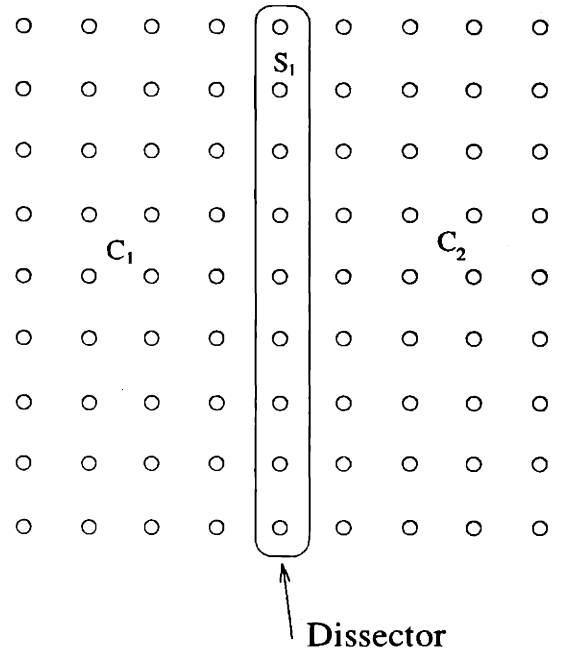

Figure 3-10: The dissector S_1 divides the mesh into two disjoint sets, one which contains the variables in C_1 and the other those variables in C_2.

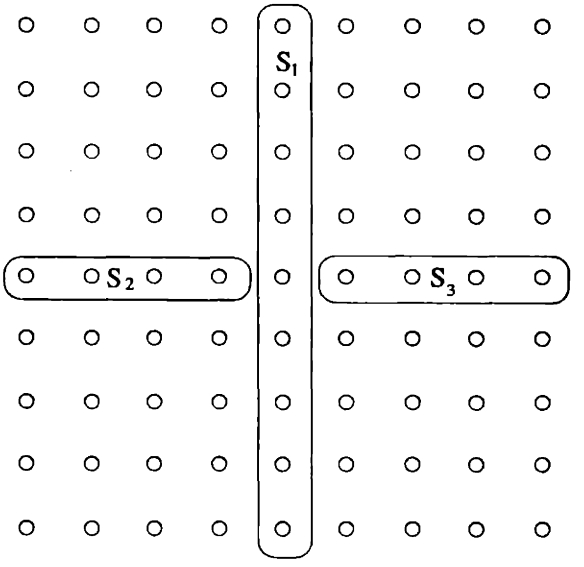

Figure 3-11: Adding a second (nested) level of separators.

1	13	5	6	91	41	42	57	49	50
2	14	11	12	92	47	48	58	55	56
3	15	9	7	93	45	43	59	53	51
4	16	10	8	94	46	44	60	54	52
37	38	39	40	95	86	87	88	89	90
17	23	21	19	96	63	61	81	73	71
18	24	22	20	97	64	62	82	74	72
33	34	35	36	98	69	70	83	79	80
25	31	29	27	99	67	65	84	77	75
26	32	30	28	100	68	66	85	78	76

Table 3.1: Nested dissection ordering of a 10-by-10 mesh.

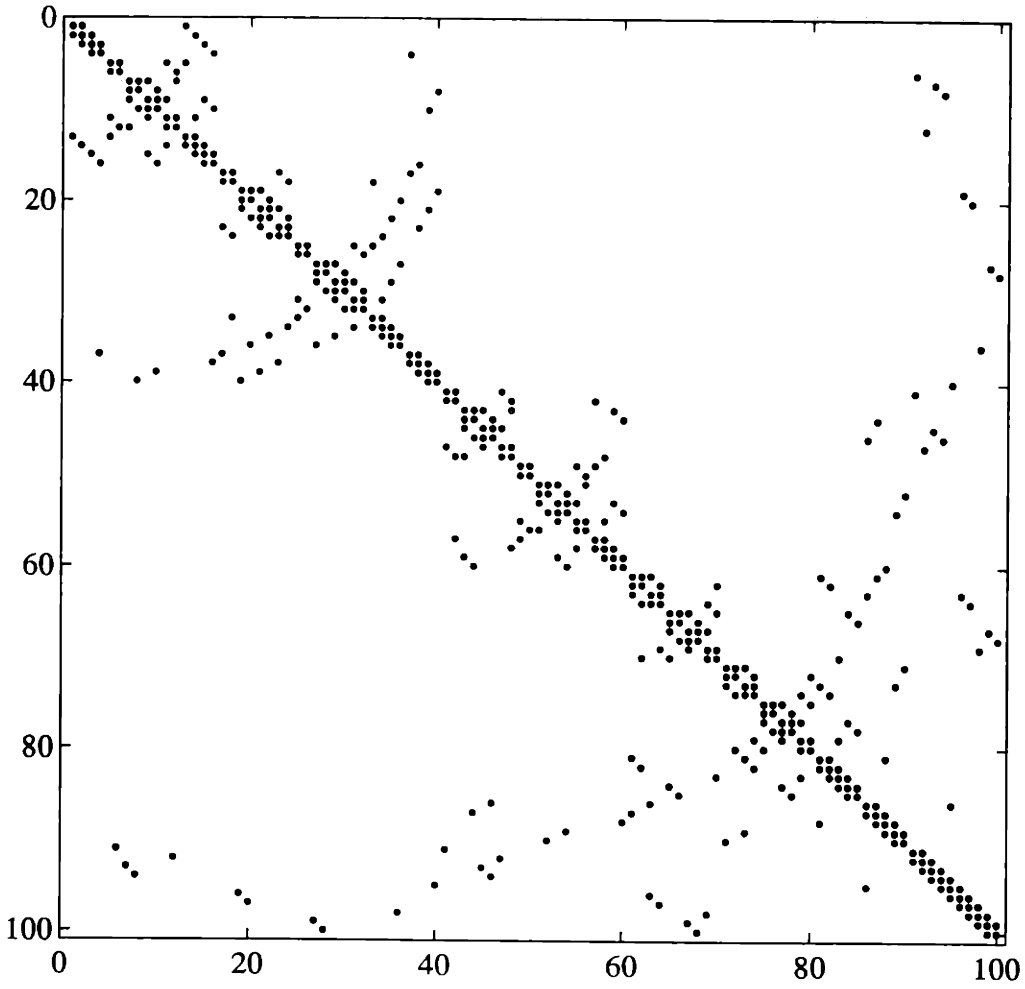

Figure 3-12: Structure of the non-zero elements in A after nested dissection ordering of a 10-by-10 mesh. (The underlying boundary value system is a Discrete Poisson with Dirichlet conditions.)

ordering compares to the finest-grain partitioning of the mesh given in Section 2.3. The last variables eliminated in a nested dissection factorization are the N variables which dissect the mesh into two halves, while the last variables eliminated for the mesh partitioning algorithm are the $4N-2$ variables on the boundary of the $N \times N$ mesh. Yet as the dissection and mesh partitionings continue recursively, one can see from Figure 3-13 that orderings begin to look somewhat alike. The orderings, however, produce significantly different algorithms. For instance, the nested dissection ordering must only be done once before the factorization begins, yet as was shown in Section 2.3 for mesh partitioning, the remaining mesh variables must be reordered after each factorization step. These differences suggest that performance measures in addition to computational complexity and storage requirements must be considered when comparing the two algorithms.

3.2.1 An approximate solution derived from Nested Dissection

By noting another similarity between nested dissection and mesh partitioning, an approximate nested dissection solution can be developed. It is likely that the approximate algorithm developed in Section 3.1 can be extended to nested dissection for 2-D boundary value applications which are equivalent to diagonally dominant matrix equations.

The approximate algorithm for nested dissection begins with the observation that just before the variables in the primary dissector S_1 (see Figure 3-10) are eliminated during the matrix factorization, the block matrices W_1 and W_2 in (3.9) have been zeroed. In other words, Gaussian Elimination has proceeded to the point where (3.9) becomes

$$A = \begin{bmatrix} U_1 & 0 & Z_1 \\ 0 & U_2 & Z_2 \\ 0 & 0 & \tilde{A}_3 \end{bmatrix} \tag{3.10}$$

where U_1 and U_2 are upper-triangular. The block element $\tilde{A}_3$ will be full due to fill-in during the factorization. Because the dissector S_1 contains N elements for an N-by-N mesh, factoring $\tilde{A}_3$ alone will take approximately $\frac{2N^3}{3}$ computations [16]. Hence, one would like a sparse approximation to $\tilde{A}_3$ similar to that given for $\tilde{D}_\rho$ and $\tilde{Q}_{22}$ in Section 3.1. Because $\tilde{A}_3$ represents the coupling between elements in the dissector S_1 after all the other variables in the mesh have been eliminated, one would expect for diagonally dominant systems of

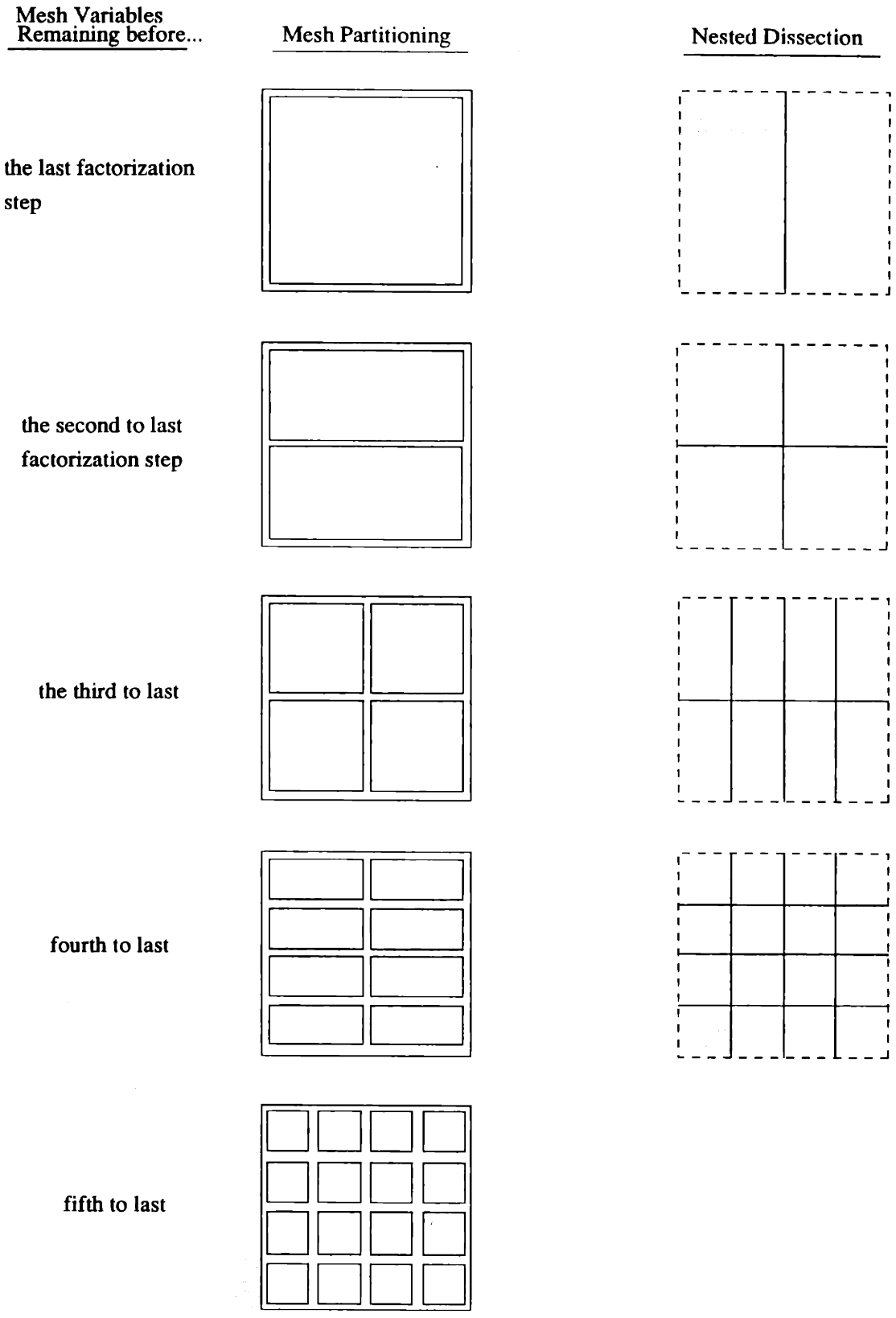

Figure 3-13: Comparison of nested dissection and mesh partitioning (see Section 2.3). Mesh variables lie along solid lines in figure.

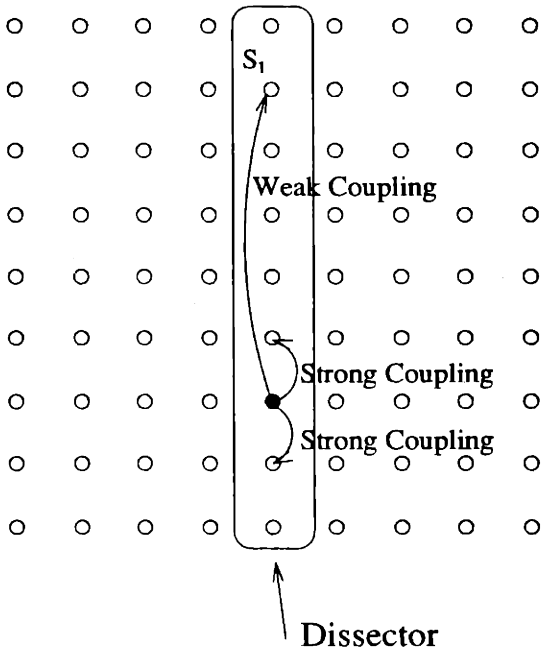

Figure 3-14: Coupling between variables in a dissector, just before the dissector is to be eliminated in the factorization.

equations that the coupling decays rapidly with physical distance between variables in S_1. This decay is illustrated in Figure 3-14, which conveys a message identical to that given in Figure 3-1.

Unlike the radial states described in Chapter 2, S_1 is not circular and $\tilde{A}_3$ will not be circular. The approximation then follows as a simple modification to the approximations for $\tilde{D}_\rho$ and $\tilde{Q}_{22}$ as

$$\text{approx}(\tilde{A}_3) = \left\{ \begin{array}{ll} \tilde{A}_3(i,j), & |i-j| \leq B \\ 0, & \text{otherwise} \end{array} \right\} \tag{3.11}$$

where B is again the bandwidth of the approximation. Furthermore, since A in (3.9) has a self-similar structure after nested dissection ordering, the blocks A_1 and A_2 have the same form as A. The blocks A_1 and A_2 represent coupling between elements in C_1 and C_2 of Figure 3-10, whose dissectors will contain approximately $N/2$ elements. Hence, at some point in the factorization, the matrices resulting from factoring A_1 and A_2 will both contain full matrices with dimension $N/2 \times N/2$. These matrices and all similar matrices resulting from dissections with significant dimension can be approximated by (3.11). Because factoring these full matrices obviously dominates the $O(N^3)$ computations required for the nested

dissection factorization, the approximation should significantly reduce the total number of computations. The questions which must then be answered are

- How good is the approximation for a given bandwidth and degree of diagonal dominance?
- How do we determine the resulting approximate algorithm's computational and storage complexity?
- How does the nested dissection approximation perform relative to the mesh partitioning approximation given in Section 3.1?

Full answers to such questions are beyond the scope of this thesis. However, since nested dissection is more efficient (in terms of both computation and storage) than mesh partitioning, the approximate nested dissection algorithm will also be more efficient than the approximate mesh partitioning algorithm for any given approximation bandwidth; but the relative accuracy of the approximations is unknown. Furthermore, the approximate nested dissection is much easier to implement, since only the initial ordering is required, and the approximation consists of a simple banding of a matrix about the main diagonal.

3.3 Examples

The purpose of this thesis is to develop a direct, efficient, parallelizable algorithm for 2-D acausal IIR filters. The block LU algorithm, as well as a parallel version, were derived in Chapter 2 as a framework for solving 2-D acausal IIR filters. These algorithms led to a more efficient approximate algorithm which works well for 2-D boundary value systems equivalent to diagonally dominant matrix equations. Such systems are a common by-product of discretizing many elliptic P.D.E.'s with finite difference methods. This section illustrates some possible applications of the approximate algorithm, such as digital filtering, and compares the results to the solution obtained by the nested dissection algorithm.

3.3.1 Setting up the examples

For the examples, we again consider a filter described by an NNM difference equation

$$x[i,j] = nx[i,j+1] + sx[i,j-1] + ex[i+1,j] + wx[i-1,j] + u[i,j] \qquad (3.12)$$

constrained by boundary conditions. Unless stated otherwise, the boundary conditions are Dirichlet-type and are derived from the input as follows: if $u[i,j]$ is given over the square $\Omega = [1,I] \times [1,I]$, then $x[i,j] = u[i,j]$ along the boundary of Ω. One must then solve for $x[i,j]$ over the square $\Omega_x = [2,I-1] \times [2,I-1]$.

The applicability of the NNM difference equation was detailed in Chapter 2. However, it is also worth noting that when $n = s$ and $e = w$, the NNM corresponds to a zero-phase filter. One of the reasons that FIR filters are preferred to IIR filters is captured in [10] where it is written that "One of the biggest advantages that FIR filters enjoy over IIR filters is the fact that implementable FIR filters can be designed to have purely real frequency responses", hence, zero-phase. Yet, ignoring the filter design issue, some efficient algorithms have already been developed in this thesis which implement the solution to a zero-phase (which implies acausal) IIR filter. Nested dissection is one such solution which has been known by those solving P.D.E.'s since the early '70's.

Although some very general results are obtained regarding the performance of the approximate algorithm, it is beyond the scope of this thesis to develop any analytic bounds on the performance of the filter implementations developed in Chapters 2 and 3. The lack of any closed-form measures of performance is in part due to the large number of parameters which must be considered when measuring the algorithm's performance. Those parameters which impact the efficiency and accuracy of the algorithms developed in this thesis are as follows:

- the filter coefficients n, s, e, and w
- the input characteristics (e.g. the 2-D DFT of the input)
- the size of the filter domain, I
- the number of local regions, 2^M, which can also be seen as the degree of parallelism in the algorithm
- the radius N at which the approximations begin, i.e. approximate $\bar{D}_\rho$ for all $\rho \geq N$

The coefficients chosen for the filter are particularly important, since they determine the filter's properties (e.g. frequency response or the possibility of diagonal dominance in the augmented linear system of equations); yet, they span and infinite set of possibilities. The following examples attempt to evaluate the performance of the filter as these parameters

vary. It should be noted that the value chosen for N in the approximate algorithm has surprisingly little effect on the performance of the algorithm. For ease in programming, the following examples use the convention $N = B + 1$.

All of the examples are implemented on a SUN Sparc2 with 64MB of RAM using MATLAB 4.0. Nested dissection is implemented using the sparse matrix package of MATLAB 4.0 (which is a very nice programming environment for handling sparse matrix operations, particularly inversions). All algorithms are thus implemented on a single processor.

Error Measures

Three measures of error are used in this section, although the utility of any given error measure depends heavily upon the nature of the application. One measure, the maximum percentage error is defined to be

$$d_1(x_1, x_2) = \max_{1 \le i,j \le I} \frac{|x_1[i,j] - x_2[i,j]|}{|x_1[i,j]|}$$

where x_1 will be considered in this section to be the output obtained from nested dissection and x_2 is the output obtained from one of the algorithms developed in this thesis. This error measure is particularly demanding, and will produce spurious results when $x_1[i,j]$ is either 0 or very small.

Another measure of error attempts to reduce the affect of outliers on d_1 due to very small values of $x_1[i,j]$. This error measure uses the 2-norm, but first we stack the variables $x_1[i,j]$ and $x_2[i,j]$ over the rectangle Ω into the column vectors x_1 and x_2, respectively. The second error measure d_2 is then

$$d_2(x_1, x_2) = \frac{||x_1 - x_2||_2}{||x_1||_2}$$

which is similar to a signal to noise ratio.

A third error classification is

$$d_3(x_1, x_2) = \frac{||x_1 - x_2||_\infty}{||x_1||_\infty}$$

This measure is similar to the first, except that it does not take into consideration large percentage errors at pixels which have insignificant values. Note that all three error measures

are in some sense invariant to the amplitude of the system input.

Inputs

In addition to error measures, we also define a set of inputs which attempt to demonstrate the algorithms immunity to variations in the input characteristics. Define the following inputs:

- **DC:** $u_1[i,j] = 1$ for all $(i,j) \in \Omega$
- **IID Gaussian Noise:** $u_2[i,j]$ is an independent, identically distributed, zero-mean normal process where each pixel has a variance of 1.0.
- **High Frequency Noise:** $u_3[i,j]$ is obtained by taking the second difference of U_2 (where U_2 is the $I+2$-by-$I+2$ matrix of the input $u_2[i,j]$) in the j direction followed by the second difference in the i direction. The MATLAB operation is (1) "U_2 = randn$(I+2, I+2)$", followed by (2) "U_3 = diff(diff$(U_2, 2)', 2)$".
- **"Diagonal" Sinusoid:** $u_4[i,j] = \sin(2\pi(i/N_i + j/N_j))$, for varying N_i and N_j.
- **"Rectangular" Sinusoid:** $u_5[i,j] = \sin(2\pi i/N_i)\sin(2\pi j/N_j)$, for varying N_i and N_j.

For $\Omega = [1,64] \times [1,64]$, $N_i = 2$, and $N_j = 32$, the a grey-scale plot of the diagonal and rectangular sinusoid are illustrated in Figures 3-15 and 3-16, respectively. These inputs are combined to form a variety of other inputs.

Difference Equations

Finally, a set of 2-D difference equations must be given. Since one of the motivations for this research is to develop direct solutions to 2-D IIR filters constrained by boundary values, a set of second-order frequency-selective 2-D filters is given. For instance, if one desired to remove high-frequency noise from a 2-D image while passing low frequencies and preserving phase, a possible low-pass filter is given by the NNM with the coefficients $n = s = e = w = .15$. The magnitude of the 2-D DFT for such a filter is given in Figures 3-17 and 3-18. A 2-D low-pass filter whose DFT has a more elliptic contour plot is given by an NNM with the coefficients $n = s = .2$ and $e = w = .1$. The frequency response of this

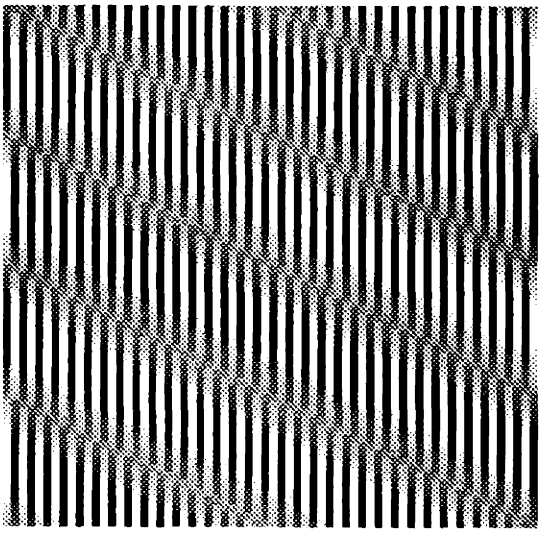

Figure 3-15: Grey scale magnitude plot the diagonal sinusoid for $N_i = 2$ and $N_j = 32$.

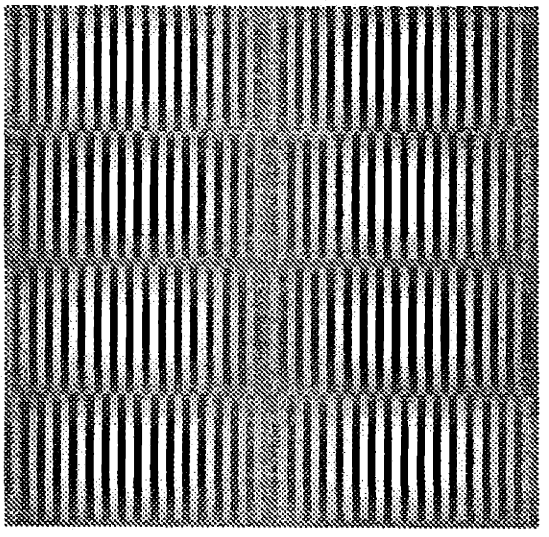

Figure 3-16: Grey scale magnitude plot the rectangular sinusoid for $N_i = 2$ and $N_j = 32$.

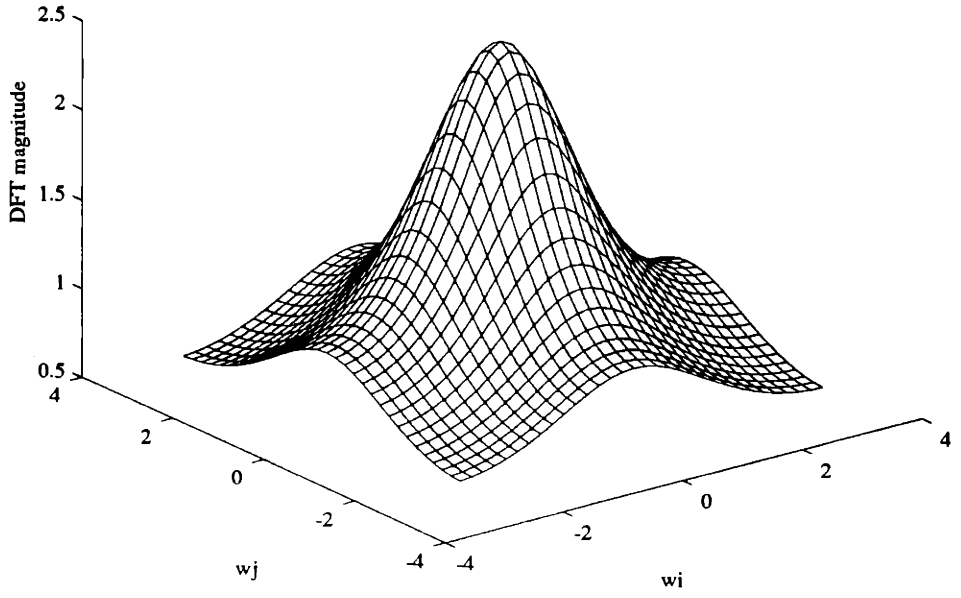

Figure 3-17: The magnitude plot of the 32-by-32 point DFT of a low pass filter described by an NNM with $n = s = e = w = .15$.

elliptic low-pass filter appears similar to that in Figure 3-17, yet its contour plot given in Figure 3-19 displays an asymmetry not present in Figure 3-18. Another possible low-pass filter is that given by the coefficients $n = e = .13$ and $s = w = .18$, also has a frequency response similar to that given by Figure 3-17. The contour plot of this skewed LPF (low pass filter) is shown in Figure 3-20. Note that unlike the previous two low-pass filters, the skewed filter of Figure 3-20 does not have zero-phase. One final low pass filter is that given by the NNM with $n = s = e = w = .05$, whose frequency response is shown in Figure 3-21 to have nearly linear roll-off on the interval $[-\pi, \pi]$.

A crude high pass filter can also be implemented with the simple NNM difference equation. When $n = s = e = w = -.15$, the filter has the frequency response given in Figure 3-22. A mixed filter can also be designed using the NNM coefficients $n = s = -.14$ and $w = e = .16$ which essentially passes high frequencies in ω_i and low frequencies in ω_j, as is shown in Figure 3-23. Note that both the mixed filter is not zero-phase.

Two other systems which will be considered in this section are used primarily to examine the numerical stability of the algorithms developed in Chapter 2 for difference equations not equivalent diagonally dominant systems. As is shown in Appendix B, the block LU algorithms developed in Chapter 2 can become singular for a countable set of NNM parameters,

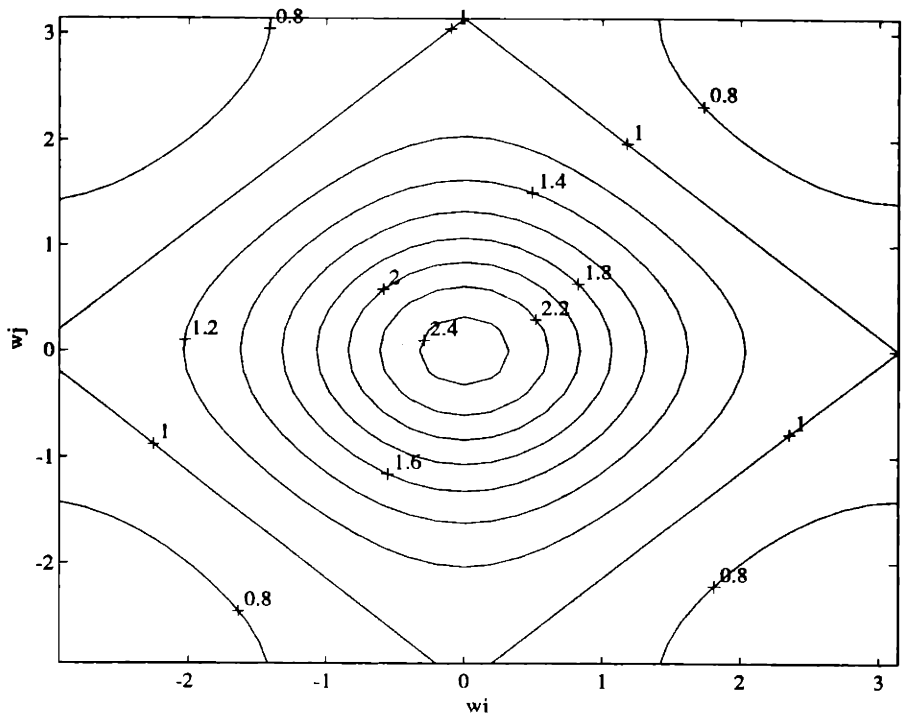

Figure 3-18: The contour plot of the magnitude of the 32-by-32 point DFT of a low pass filter described by an NNM with $n = s = \epsilon = w = .15$.

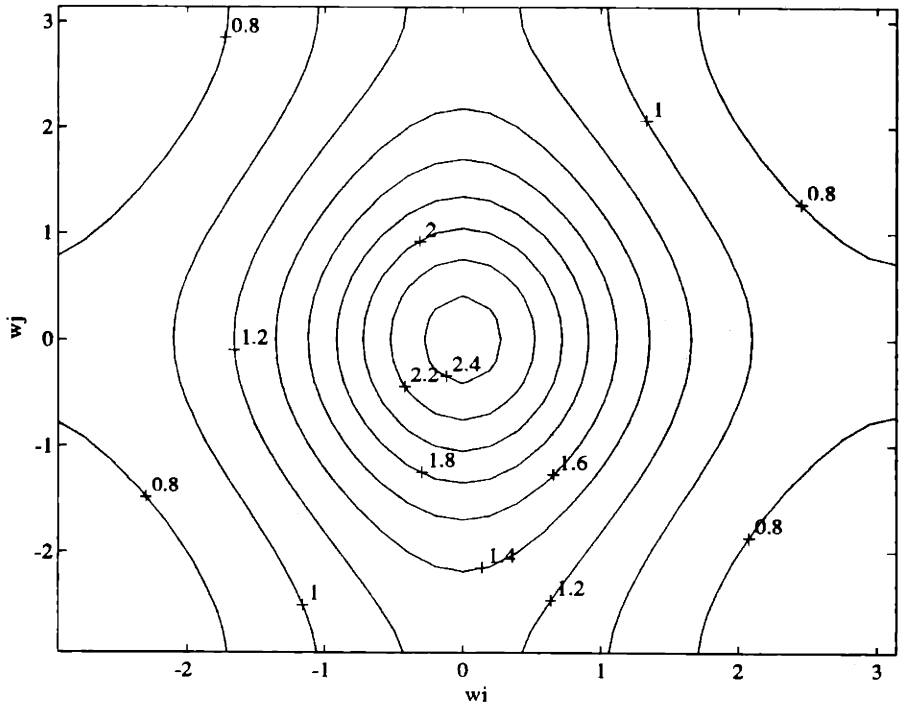

Figure 3-19: The contour plot of the magnitude of the 32-by-32 point DFT of a low pass filter described by an NNM with $n = s = .2$ and $\epsilon = w = .1$.

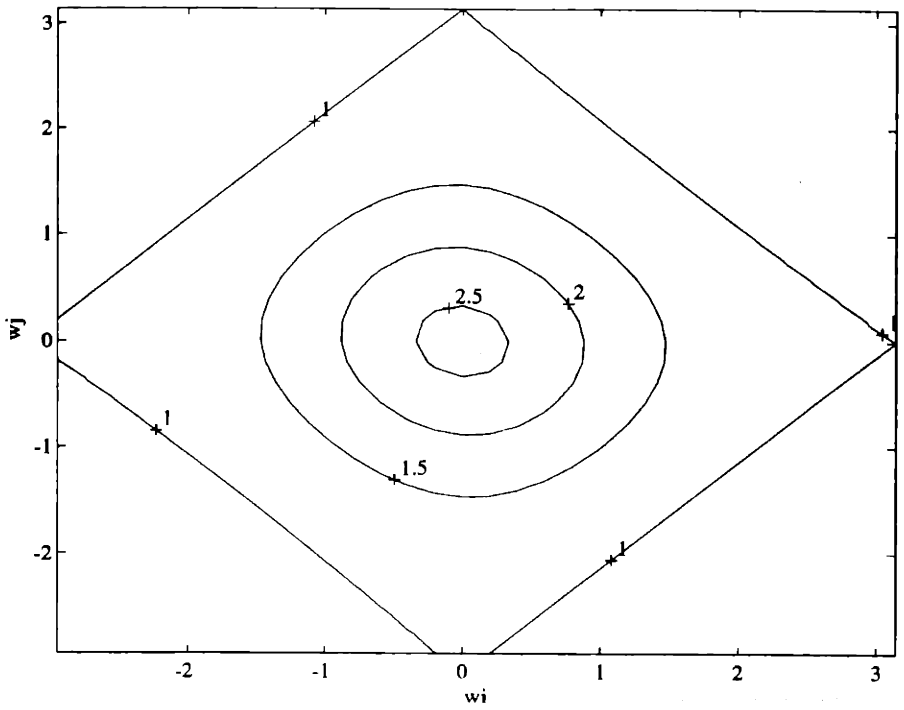

Figure 3-20: The contour plot of the magnitude of the 32-by-32 point DFT of a low pass filter described by an NNM with $n = e = .13$ and $s = w = .18$.

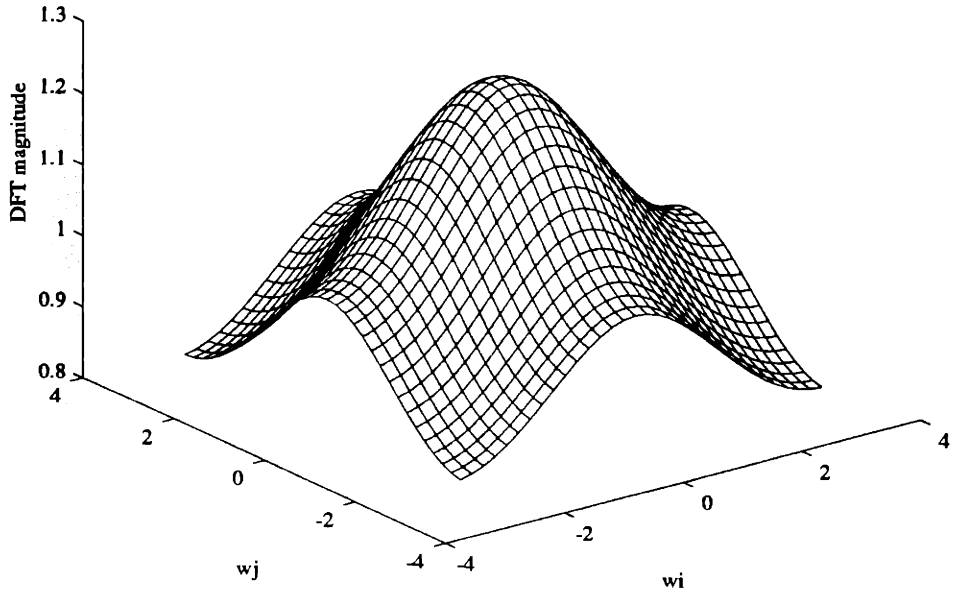

Figure 3-21: The magnitude plot of the 32-by-32 point DFT of a low pass filter described by an NNM with $n = e = s = w = .05$.

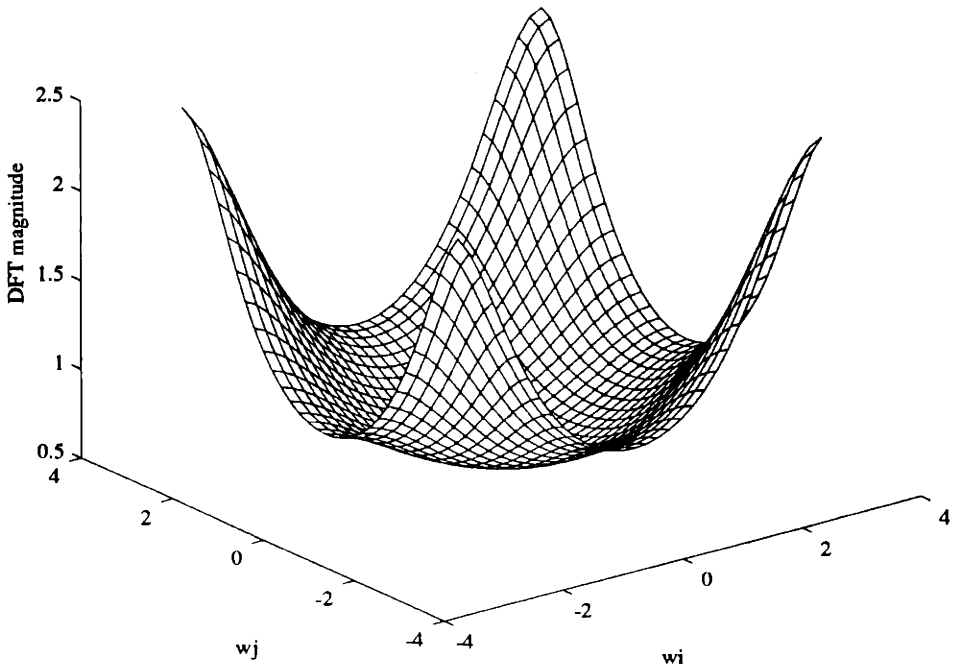

Figure 3-22: The magnitude plot of the 32-by-32 point DFT of a high pass filter described by an NNM with $n = s = e = w = -.15$.

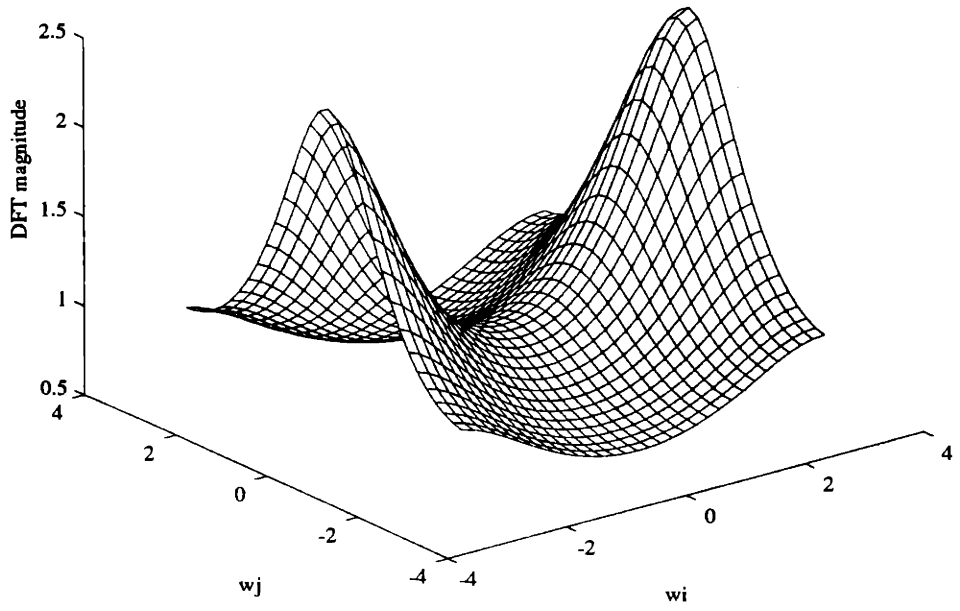

Figure 3-23: The magnitude plot of the 32-by-32 point DFT of a filter described by an NNM with $n = s = -.14$ and $e = w = .16$.

Table of Systems/Filters using the NNM		
System Class	Description	NNM coefficients $\{n,s,e,w\}$
Low-Pass Filter	Poisson/Laplacian	{.25,.25,.25,.25}
	circular contour	{.15,.15,.15,.15}
	elliptic contour	{.2,.2,.1,.1}
	skewed contour	{.13,.18,.13,.18}
	linear roll-off	{.05,.05,.05,.05}
High-Pass Filter	circular contour	{-.15,-.15,-.15,-.15}
Mixed Filter	High pass along i-axis	{-.14,-.14,.16,.16}
"Near" Singular System		{.49,.49,.49,.49}
Non-physical system		{10,10,10,10}

Table 3.2: Systems described by NNM difference equations to be used in the examples.

such as when $n = s = e = w = .5$. However, one would like to show that the algorithms are numerically stable for difference equations "near" these ill-conditioned problems, such as when $n = s = e = w = .49$. In addition, the system given by the NNM difference equation $n = s = e = w = 10$ is also considered, although such a system appears to have no physical interpretation (as a finite difference approximation). A summary of the preceding difference equations is given in Table 3.2.

The numerical results of this chapter can essentially be divided into two parts. Note that when no approximations are made, nested dissection is the choice of implementation. However, in the first part the boundary value systems are implemented with the algorithms of Chapter 2 making no approximations in order to demonstrate the numerical stability of these algorithms. The stability of both the local factorizations (block LU factorization for a single radial ordering) and the inter-processor factorization steps is given. The second part of the results uses the approximations made in Section 3.1 and describes the accuracy of the solution both when the mesh is partitioned and when it is not.

3.3.2 Numerical stability of mesh partitioning algorithm

In demonstrating the numerical stability of the mesh partitioning algorithm, we essentially seek to demonstrate empirically that the errors do not increase appreciably as the number of local and inter-processor factorization steps increase. Let x_1 correspond to the filter output obtained by nested dissection and x_2 contain that produced by the mesh partitioning algorithm of Section 2.3. Observations on numerical stability can then be drawn from

Table 3.3, which shows the solution errors obtained when a number of systems (sampled from Table 3.2) are driven by the input $u = (0.5u_2 + u_4)$ with $N_i = N_j = I/8$. Note that when $M = 0$—when there are no partitions—the simulations are only done for meshes of size $I \leq 61$, because the memory and computational requirements increase rapidly with the number of local factorization steps.

It is apparent from Table 3.3 that the errors do not increase appreciably as the number of local or inter-processor communication steps increase. In other words, even for those NNM parameters near those which cause singularities, the mesh partitioning algorithm is numerically stable. One should also note from Table 3.3 that for a given mesh size I, the errors are independent of the number of partitions, or equivalently the number of inter-processor communication steps.

Knowing that the magnitude of the errors is independent of the size of the domain Ω—independent in the sense that the errors $d_2(x_1, x_2)$ and $d_3(x_1, x_2)$ do not increase significantly when increasing from $I = 32$ to $I = 128$—one would also like the errors to be invariant of the form of the input for any given system. As an example, we investigate for some low-order filters how the errors vary as the input moves from the stop band to the pass band of the filter. Consider the circular contour low-pass filter given by an NNM with coefficients $\{n, s, e, w\} = \{.15, .15, .15, .15\}$, whose 2-D Fourier Transform is illustrated in Figures 3-17 and 3-18. A possible input is the diagonal sinusoid u_4 with $N_i = N_j$, whose Fourier Transform consists of two impulses along the line $\omega_i = \omega_j$ equidistant from the center of the frequency plane at $(\omega_i, \omega_j) = (0, 0)$. When $N_i = N_j$ varies from I to 2, the impulses move from the low-pass filter's pass band near $(\omega_i, \omega_j) = (0, 0)$ along the line $\omega_i = \omega_j$ to the stop band near $(\omega_i, \omega_j) = (\pm\pi, \pm\pi)$. The Fourier transform of the diagonal sinusoid is shown for varying frequencies in Figures 3-24 to 3-26 when $I = 64$. The rectangular sinusoid u_5 has a Fourier Transform which can be derived directly from that of the diagonal sinusoid with the same N_i and N_j by adding the two impulses obtained by reflecting the DFT of the diagonal sinusoid about the ω_i or ω_j axis. The transform of the rectangular sinusoid obtained from Figure 3-25 is given in Figure 3-27.

Fixing the size of the filter domain to $\Omega = [1, 64] \times [1, 64]$, the errors are given in Table 3.4 for both the low-pass filter of Figure 3-17 and the high-pass filter of Figure 3-22 as the inputs u_4 and u_5 move from the pass to the stop bands of both filters. For both the high-pass and low-pass filters, the value of the error is both small and relatively independent of the

NNM Parameters $\{n, s, e, w\}$	I Mesh Size	M (2^M Partitions)	$d_1(x_1, x_2)$	$d_2(x_1, x_2)$	$d_3(x_1, x_2)$
$\{.25, .25, .25, .25\}$	9	0	1.07e-14	3.02e-16	4.35e-16
Poisson	31	0	3.05e-13	9.55e-16	1.01e-15
LPF	61	0	2.16e-11	2.74e-14	2.06e-14
	16	1	5.75e-14	2.22e-16	1.86e-16
	32	1	9.32e-12	4.05e-15	2.45e-15
	64	1	4.53e-10	7.17e-14	1.36e-14
	32	2	9.02e-11	7.37e-15	1.00e-14
	64	2	1.31e-09	6.68e-15	1.22e-14
	128	2	1.47e-10	1.27e-14	3.61e-14
$\{-.15, -.15, -.15, -.15\}$	9	0	2.09e-15	3.14e-16	3.10e-16
HPF	31	0	3.54e-13	4.96e-16	9.19e-16
	61	0	4.02e-13	5.51e-16	1.21e-15
	16	1	7.77e-15	2.13e-16	9.06e-16
	32	1	6.04e-14	5.00e-16	1.17e-15
	64	1	8.73e-13	5.53e-16	1.21e-15
	32	2	4.84e-13	4.64e-16	8.69e-16
	64	2	4.76e-12	5.53e-16	1.21e-15
	128	2	1.40e-12	5.62e-16	1.18e-15
$\{.49, .49, .49, .49\}$	9	0	1.57e-13	1.31e-15	1.50e-15
	31	0	2.40e-12	5.87e-14	4.59e-14
	61	0	5.25e-10	2.58e-12	2.96e-12
	16	1	6.63e-13	5.90e-15	4.18e-15
	32	1	3.04e-12	7.34e-13	1.76e-13
	64	1	3.97e-11	1.35e-12	9.02e-13
	32	2	1.61e-11	1.63e-14	1.99e-14
	64	2	7.10e-11	1.33e-13	1.27e-13
	128	2	1.15e-07	5.48e-12	6.13e-12
$\{10, 10, 10, 10\}$	9	0	5.60e-14	3.94e-15	1.11e-14
	31	0	3.39e-11	5.99e-14	4.58e-14
	61	0	3.82e-09	8.57e-13	8.65e-13
	16	1	5.74e-14	4.04e-14	1.81e-14
	32	1	3.40e-11	4.45e-14	2.28e-14
	64	1	4.56e-09	6.36e-13	4.55e-13
	32	2	1.08e-11	3.46e-14	4.39e-14
	64	2	6.10e-09	4.71e-13	2.96e-13
	128	2	2.12e-08	3.82e-12	4.19e-12

Table 3.3: Changes in error with the size of the domain $\Omega = [1, I] \times [1, I]$ and the number of local and inter-processor factorization steps. The input is $u = 0.5u_2 + u_4$ with $N_i = N_j = I/8$.

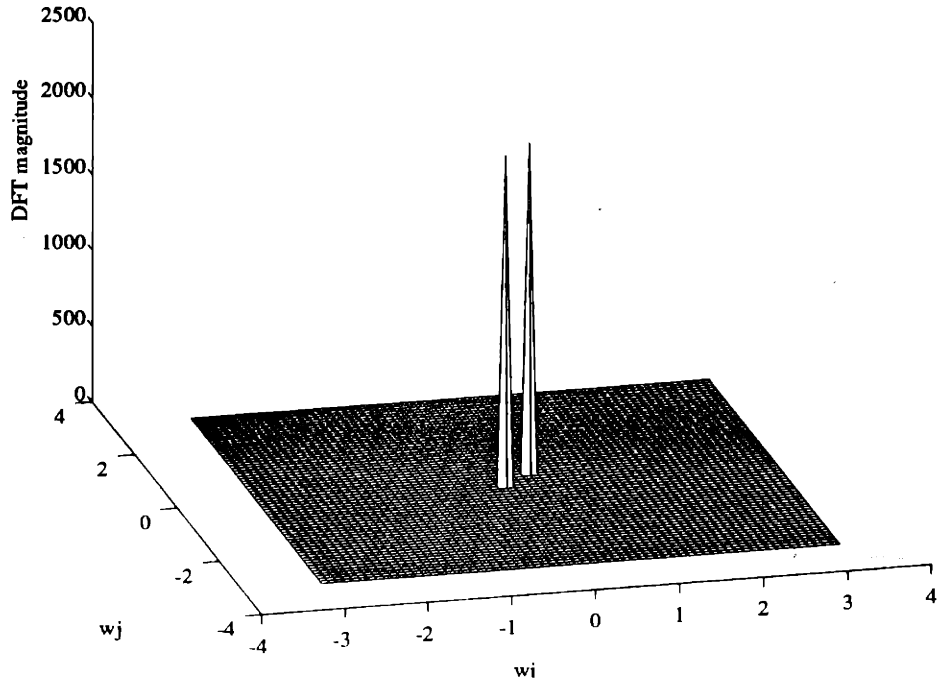

Figure 3-24: Fourier Transform of the diagonal sinusoid for $N_i = N_j = I/2 = 32$ and $\Omega = [1, 64] \times [1, 64]$.

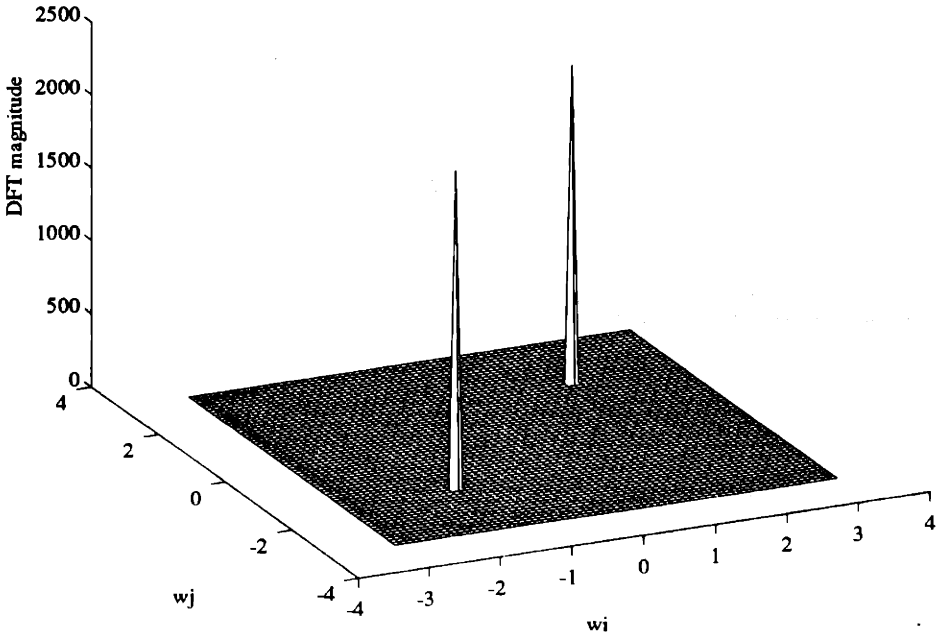

Figure 3-25: Fourier Transform of the diagonal sinusoid for $N_i = N_j = I/16 = 4$ and $\Omega = [1, 64] \times [1, 64]$.

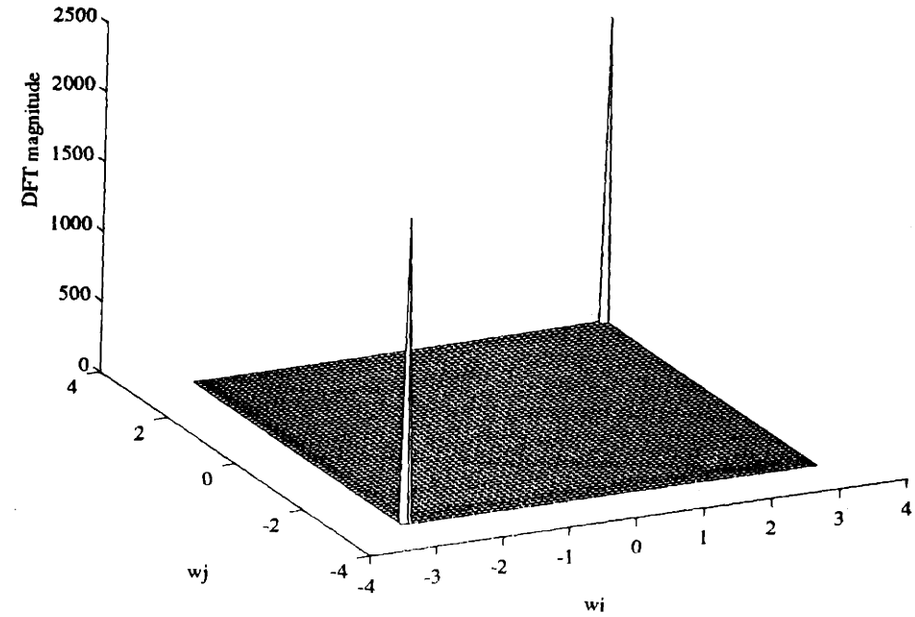

Figure 3-26: Fourier Transform of the diagonal sinusoid for $N_i = N_j = I/31 = 64/31$ and $\Omega = [1, 64] \times [1, 64]$.

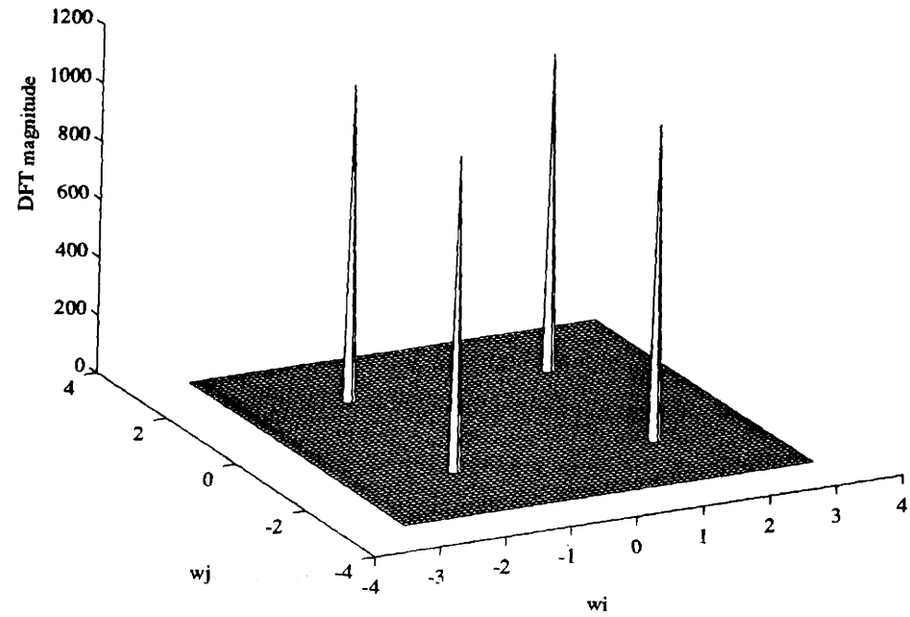

Figure 3-27: Fourier Transform of the rectangular sinusoid for $N_i = N_j = I/16 = 4$ and $\Omega = [1, 64] \times [1, 64]$.

NNM parameters	Input	$N_i = N_j$	$d_1(x_1, x_2)$	$d_2(x_1, x_2)$	$d_3(x_1, x_2)$
{.15, .15, .15, .15}	u_4 diagonal sinusoid	I/2	0.136	4.37e-16	1.82e-15
		I/8	2.02	4.90e-16	1.76e-15
Low Pass Filter		I/16	0.137	5.02e-16	2.37e-15
		I/31	0.007	6.59e-16	2.45e-15
	u_5 rectangular sinusoid	I/2	0.385	5.27e-16	1.82e-15
		I/8	2.02	4.90e-16	1.76e-15
		I/16	1.83	4.77e-16	1.72e-15
		I/31	1.63	6.79e-16	1.94e-15
$\{-.15, -.15, -.15, -.15\}$	u_4 diagonal sinusoid	I/2	1.55	6.51e-16	2.45e-15
		I/8	0.416	5.99e-16	2.54e-15
High Pass Filter		I/16	0.183	4.92e-16	1.60e-15
		I/31	0.111	4.39e-16	1.82e-15
	u_5 rectangular sinusoid	I/2	1.25	6.80e-16	1.94e-15
		I/8	3.037	5.83e-16	2.07e-15
		I/16	0.478	5.36e-16	1.60e-15
		I/31	1.25	5.19e-16	1.73e-15

Table 3.4: Error in response of high and low pass filter to diagonal and rectangular sinusoids when $M = 2$ (four partitions) and $I = 64$.

frequency of the input. The large errors in Table 3.4 for error measure $d_1(x_1, x_2)$ are outliers due to this error measure's sensitivity to near zero output values.

The mesh partitioning algorithm has been shown for a number of systems to be immune to input variations, and exhaustive simulations indicate that this invariance is a property of most NNM boundary value systems. However, in the context of filtering, one is not only concerned with how closely x approximates the exact solution to $Ax = b$, but one also would like to know the form of x when a filter system (which is a difference equation *and* boundary conditions) is driven by a complex exponential input $e^{j(\omega_1 n_1 + \omega_2 n_2)}$. The filter system frequency responses shown in Figures 3-17 through 3-23 only have significant meaning if complex exponentials are eigenfunctions of the filter system. Complex exponentials, such as diagonal sinusoid, are eigenfunctions to systems described by linear constant-coefficient difference equations when the input is defined over the entire discrete plane in $\Re^2$, but we are using boundary conditions to window the region over which the output is to be computed. The question then is whether or not the boundary conditions can be chosen such that the filter output corresponds to a weighted sum of the individual harmonics of the input. Namely, do the frequency responses of the filters, which are derived from the 2-D

difference equations, correspond to the eigenvalues of the system.

As was mentioned in Section 3.3.1, all of the previous examples used Dirichlet conditions in which the output values along the boundary were set to the input values continued to these locations in the mesh. Such boundary conditions are appropriate if all the input harmonics lie entirely within the pass band of the filter (and the filter pass-band has a magnitude of one), yet would be inappropriate if all of the harmonics are outside the pass band, since one would then expect the output to be near-zero along the boundary and in the interior. Dirichlet conditions which zero the filter output along the boundary would be inappropriate for similar reasons. The following example, however, illustrates that a sinusoidal input to a 2-D IIR filter with boundary conditions is seen at the filter output as essentially a sinusoid of the same frequency and desired attenuation with slight distortion at the boundaries.

Figures 3-28 to 3-33 show the contour lines and corresponding Fourier Transform of the input and output to the low-pass filter defined by an NNM with the coefficients $\{n, s, e, w\} = \{.15, .15, .15, .15\}$ when the input is a diagonal sinusoid in the filter's pass-band. It can be seen from the figures that if the boundary conditions are chosen properly, the sinusoidal output will have little distortions. Yet even for the zero Dirichlet conditions in which the distortions are greater, the distortions are present primarily at the boundaries of the 64-by-64 mesh. Furthermore, as the size of the mesh increases, the distortions become less noticeable. This decrease is illustrated in Figure 3-34, which shows the output from the filter computed over a 250-by-250 mesh with zero Dirichlet conditions.

The Fourier Transforms of the outputs when $I = 64$, illustrated in Figure 3-31 and 3-33, confirm that the outputs are basically diagonal sinusoids at the same frequency as the input, and with magnitudes amplified by the magnitude of the low-pass filter's Fourier Transform (see Figure 3-17) at these frequencies. Hence, even for these primitive boundary conditions, the outputs to the 2-D IIR filters are the desired diagonal sinusoids with small distortions near the boundaries.

These cases in which the inputs are confined entirely to the pass or stop bands of a filter, however, is unrealistic. More appropriate boundary conditions for filtering applications might be Neumann conditions, which are detailed in Section 2.1. With large sized mesh, one would expect that the output from a low-pass filter will change "slowly" in any spatial direction, so a reasonable boundary condition is to assume that the gradient of the output

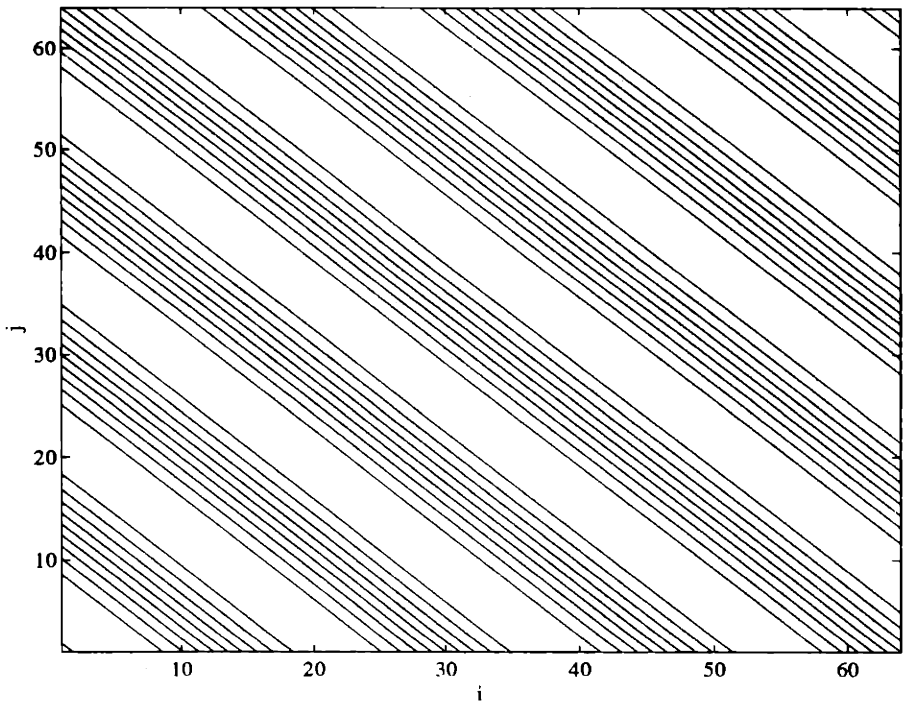

Figure 3-28: Contour plot of a diagonal sinusoid (truncated at the boundaries) with $N_i = N_j = I/2$ and $\Omega = [1, 64] \times [1, 64]$.

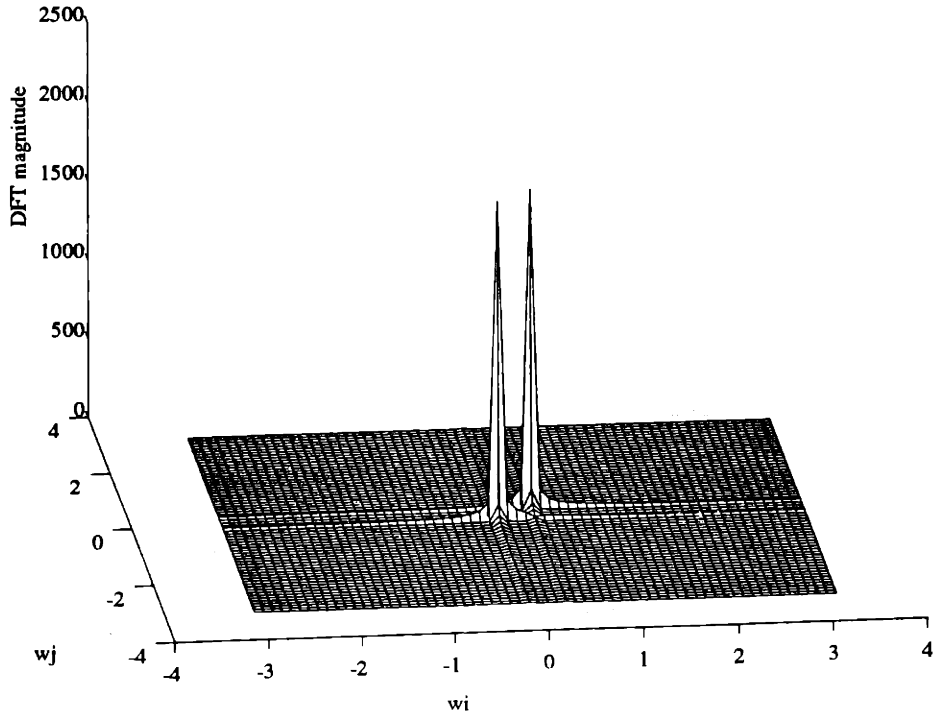

Figure 3-29: Fourier Transform of a diagonal sinusoid (truncated at the boundaries) with $N_i = N_j = I/2$ and $\Omega = [1, 64] \times [1, 64]$.

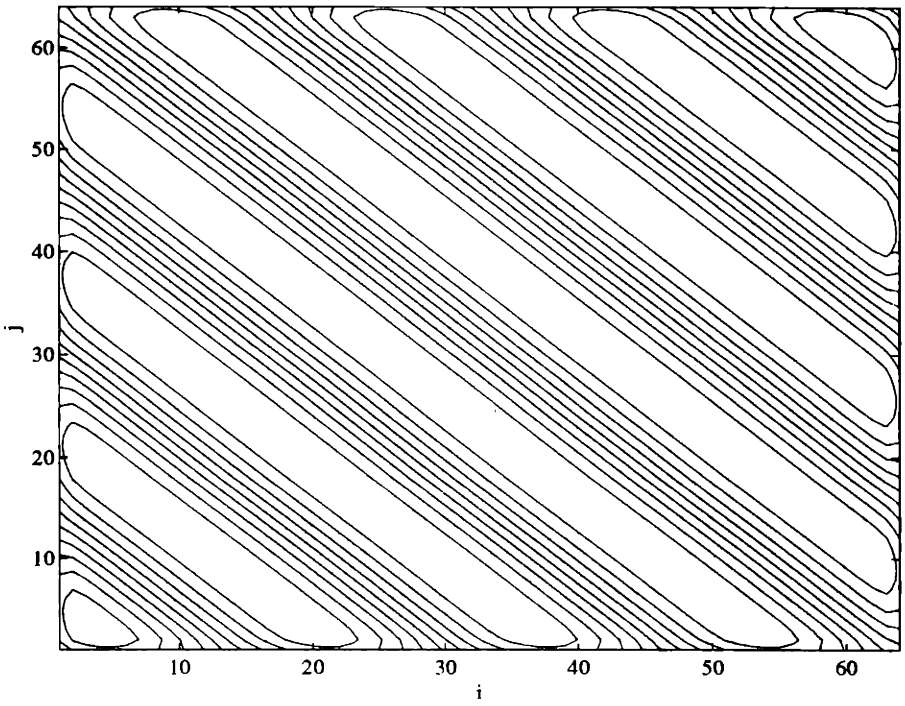

Figure 3-30: Contour plot of the output to a low-pass IIR filter with zero Dirichlet conditions and an input of a diagonal sinusoid (truncated at the boundaries) with $N_i = N_j = I/2$ and $\Omega = [1, 64] \times [1, 64]$.

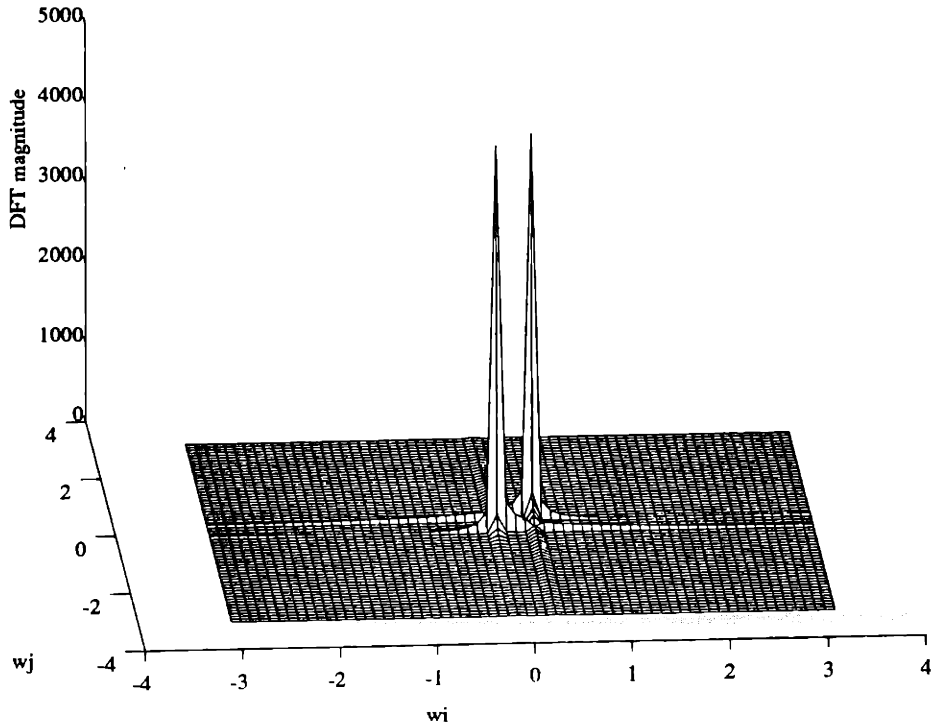

Figure 3-31: Fourier Transform of the filter output with zero Dirichlet conditions.

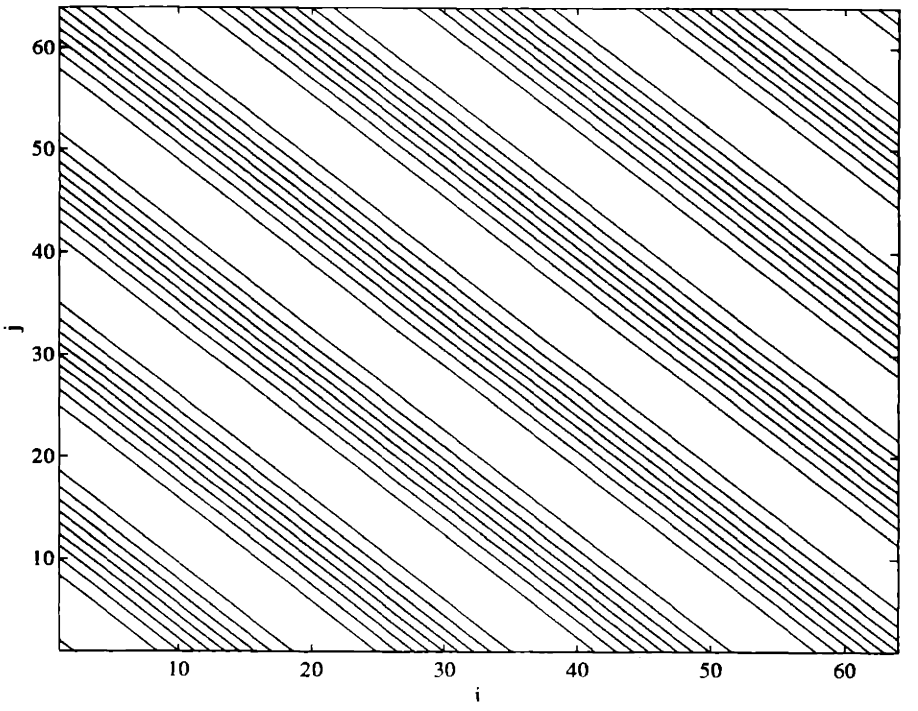

Figure 3-32: Contour plot of the output to the low pass filter when the Dirichlet conditions are derived from the filter input.

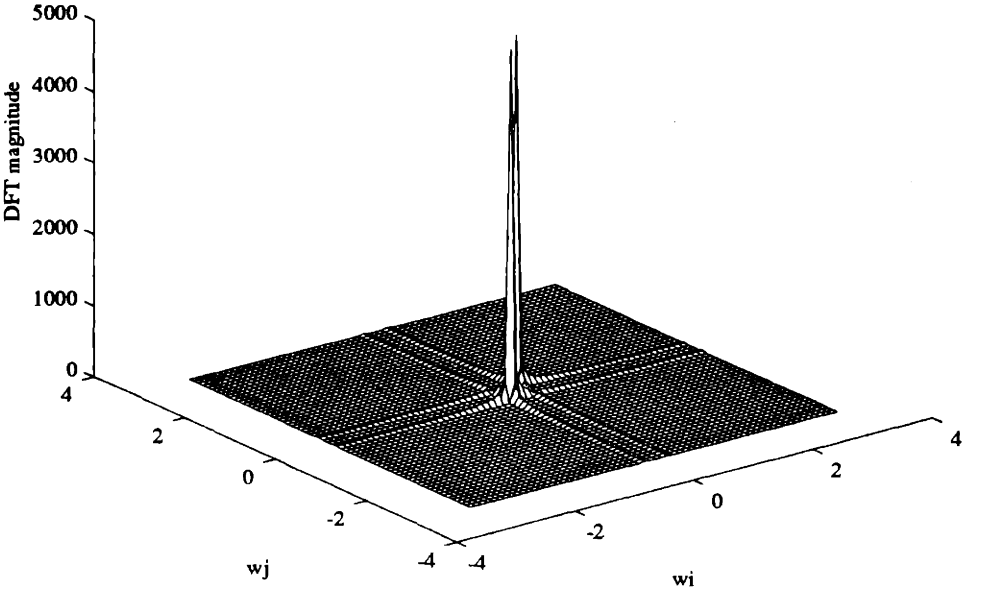

Figure 3-33: Fourier Transform of the filter output with Dirichlet conditions derived from the filter input.

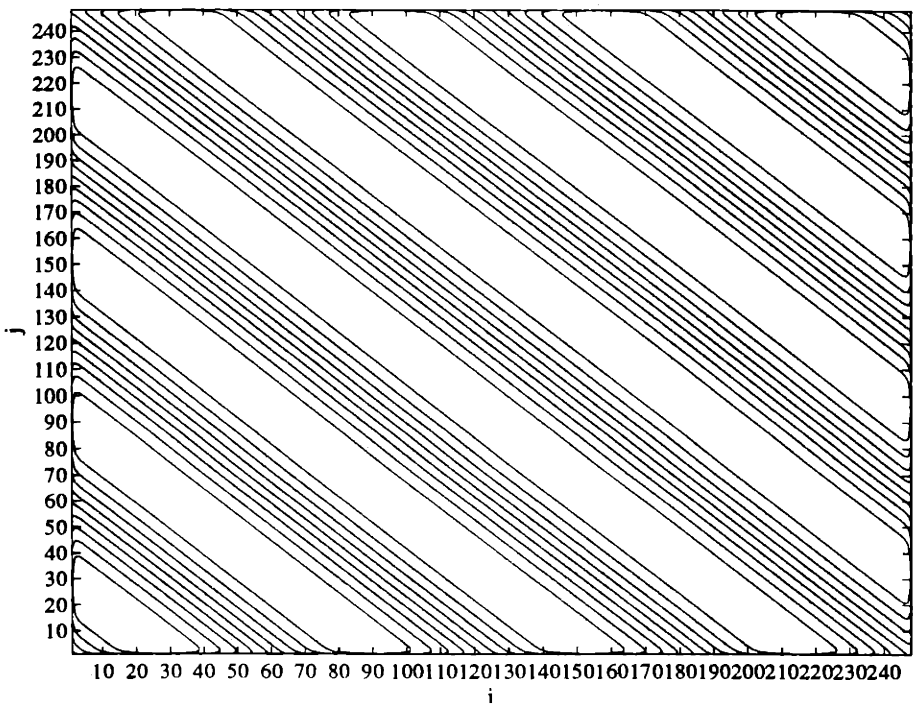

Figure 3-34: Contour plot of the output to a low-pass IIR filter with zero Dirichlet conditions and an input of a diagonal sinusoid (truncated at the boundaries) with $N_i = N_j = I/2$ and $\Omega = [1, 250] \times [1, 250]$.

normal to the boundary of the mesh is zero. In other words, the output values along the boundary equal their nearest neighbor just inside the boundary. For a high-pass filter, an appropriate boundary condition might be to constrain the output values along the boundary to be equal the negative of their nearest neighbors just inside the boundary. In general, the appropriate choice of boundary conditions will depend upon the filtering application.

3.3.3 Performance with approximations

Assuming that one is willing to sacrifice accuracy for computational savings, this section then details by example the level of accuracy which can be achieved for a given savings in computation. This section also attempts to define the class of filters for which the errors are acceptable when implemented with the approximate mesh partitioning algorithm. Note that for the examples in this section, x_1 is still the solution obtained by nested dissection, and x_2 is now that obtained by the approximate mesh partitioning algorithm.

Since the only filters discussed in this thesis are those implicitly defined by a NNM difference equation with scalar coefficients (n, s, e, w), it would suffice in this context to define the set of NNM coefficients for which the approximation performs well. Ideally, one

would like an analytic expression of the form

$$f(n, s, e, w, B) \leq d_2(x_1, x_2)$$

from which one could determine the accuracy of an approximate implementation with bandwidth B for any given difference equation. Such a rigorous derivation is beyond the scope of this thesis, yet very strong statements can be made about the applicability of the approximation for any NNM difference equation based upon the degree of diagonal dominance of the filter (if the 2-D filter is augmented into the large systems of equations $Ax = b$, any row in A other than those given by the boundary conditions will sum to $(1 - n - s - e - w)$).

The accuracy of the approximation algorithm, for any fixed bandwidth, turns out to be proportional to the ratio

$$q = \frac{1}{|n| + |s| + |e| + |w|}$$

which is simply a measure of the diagonal dominance of the system. When $q \geq 1$, the filter system is defined to be diagonally dominant, and when $q > 1$ the filter system is defined to be strictly diagonally dominant. The errors introduced by the approximation algorithm are essentially discontinuous at $q = 1$. Namely, for $q \geq 1$, the errors decrease rather smoothly as q increases. Yet for any reasonably small approximation bandwidth, the errors introduced by the approximation jump dramatically near $q = 1$. The relationship between diagonal dominance and errors in the solution due to approximation is given in Figure 3-35 for NNM filters in which $n = s = e = w$. (The errors in Figure 3-35 were obtained without partitioning the mesh ($M = 0$), with the approximation bandwidth $B = 5$, $\Omega = 40 \times 40$, and the input is given by u_1). The data in this figure is obtained for filters in which all the filter coefficients are equal. Let $c = n = s = e = w$, then $q = \frac{1}{4|c|}$. Note that in this case, $q = 1$ corresponds to the Discrete Poisson equation—a system for which the approximation algorithm was stated in Section 3.1 to produce unacceptable errors in the solution. Note that although the results given in Figure 3-35 are for filters in which all the coefficients are equal, a similar relationship is characteristic of diagonally dominant systems which do not exhibit this symmetry.

Again, since this thesis is partly concerned with filtering applications, one again would like to know how well suited these approximate solutions are for filtering applications. Namely, one would like a sinusoidal input to appear at the output as scaled version of

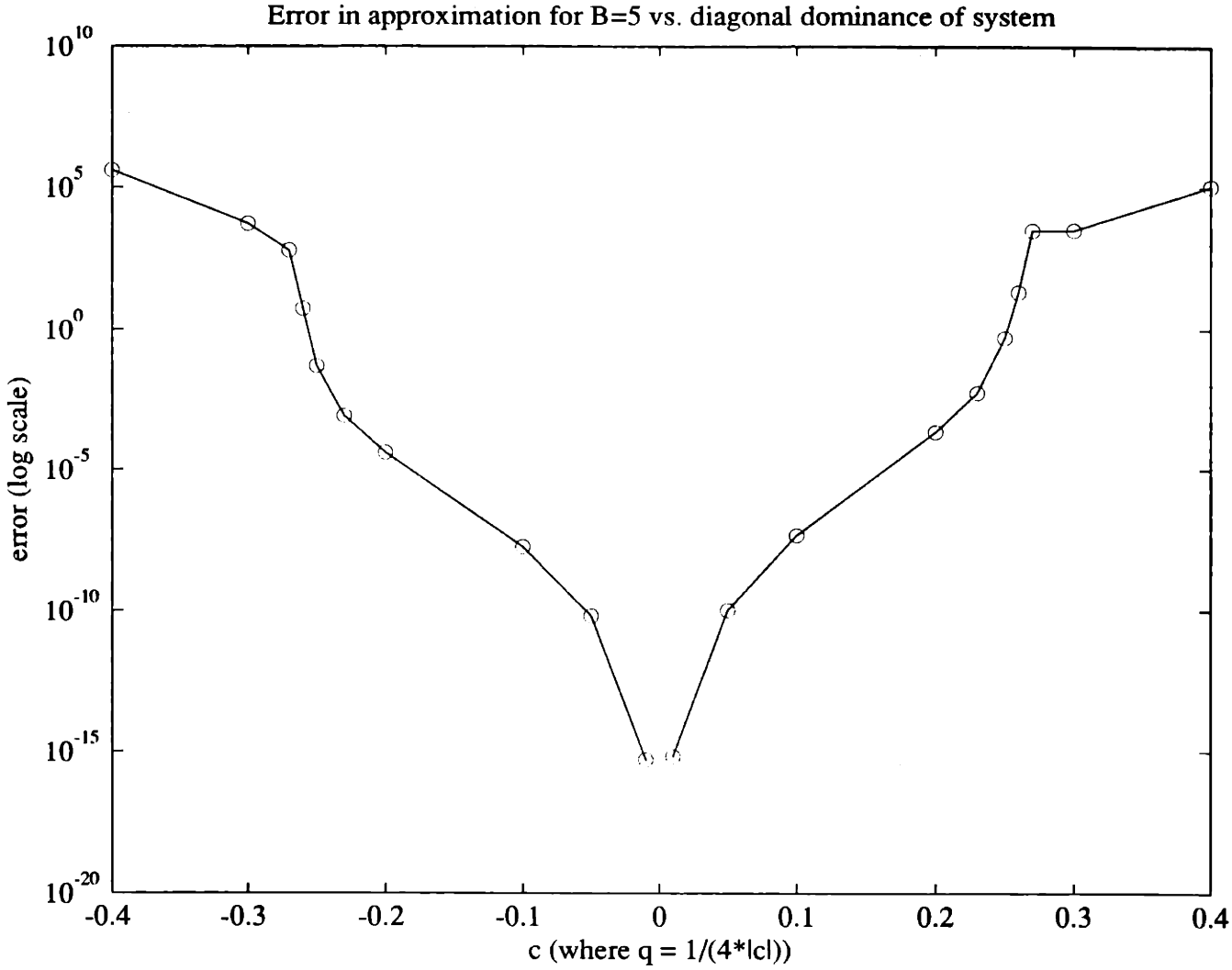

Figure 3-35: Log plot of $d_2(x_1, x_2)$ when $c = n = s = \epsilon = w$ and $B = 5$. The diagonal dominance ratio is then $q = \frac{1}{4|c|}$.

the input, hoping that the approximation does not introduce any distortions beyond those mentioned in the previous section. The following example investigates how well a NNM filter can extract noise from a pure sinusoid without corrupting the sinusoid. Consider the zero-phase (in the sense that the filter frequency response is zero-phase without considering boundary conditions) mixed filter given in Table 3.2, which has the frequency response illustrated in Figure 3-23, driven by the input $u = 0.4u_3 + u_5$ with $N_i = I/31$ and $N_j = I/2$ on a mesh of size $\Omega = [1, 64] \times [1, 64]$. The frequency response of this input is given in Figure 3-36, which consists of a rectangular sinusoid embedded in high frequency noise. Because the rectangular sinusoid is in the pass-band of the mixed filter, the filter can be used to remove a significant portion of the noise. The filter output obtained by the mesh partitioning algorithm ($M = 2$) without any approximations is given in Figure 3-37, and that obtained by the approximate algorithm with $B = 5$ is given in Figure 3-38. The transforms of the output testify to the fact that the mesh partitioning algorithms, both with and without approximations, essentially attenuate the high frequency noise and scale the rectangular sinusoid by the magnitude of the filter in the pass band. The B.C.'s used in this case were again Dirichlet conditions derived from a continuation of the input. The errors in the approximation for this example are $d_2(x_1, x_2) = 1.20 \times 10^{-6}$ and $d_3(x_1, x_2) = 2.10 \times 10^{-6}$.

With the high frequency noise added to the input, it is difficult to detect the distortions in the output sinusoid introduced by the Dirichlet conditions when implementing the filter with the approximate mesh partitioning algorithm. Instead, consider the again the high-pass filter from Table 3.2. For an input which again is the low-frequency rectangular sinusoid given in Figures 3-28 and 3-29, the output is found with an approximation bandwidth of $B = 5$. The contour of the approximate output and its Fourier Transform are given in Figures 3-39 and 3-40. One can see again from the contour plot and Fourier Transform that the output is essentially a scaled version of the input. If the boundary conditions were not chosen to be a properly scaled extension of the sinusoidal input, as would be the case for any general application in which the input is composed of a superposition of complex exponential at many frequencies, more distortion would be evident at the boundaries. One would instead design more general boundary conditions for the application, such as Neumann conditions for a low-pass filter.

In these filtering examples it was shown that even for small approximation bandwidths, low-order filters can be implemented which essentially attenuate the harmonics in the input

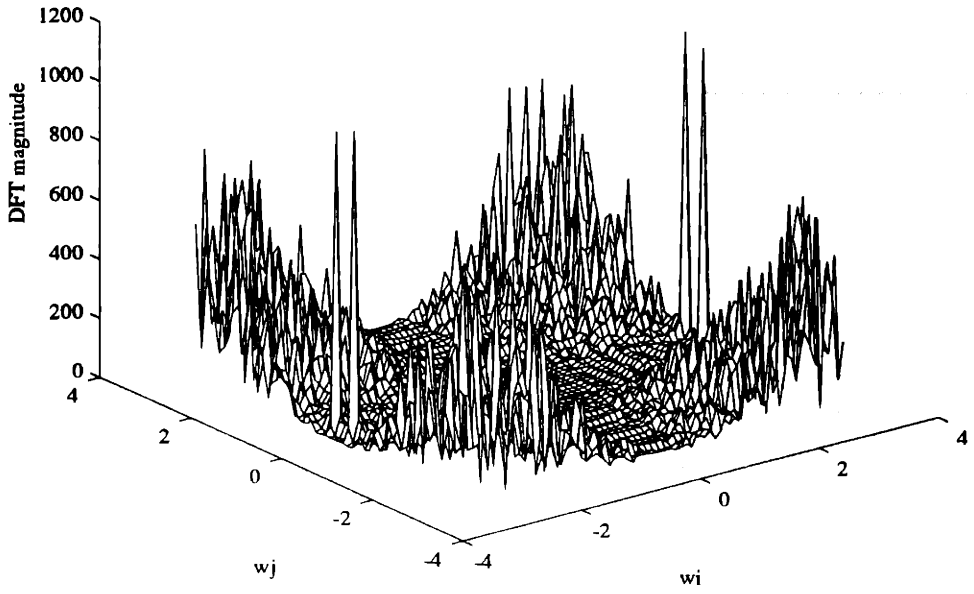

Figure 3-36: DFT of $u = 0.2u_3 + u_5$ with $N_i = I/31$ and $N_j = I/2$ which is used as an input to the mixed filter

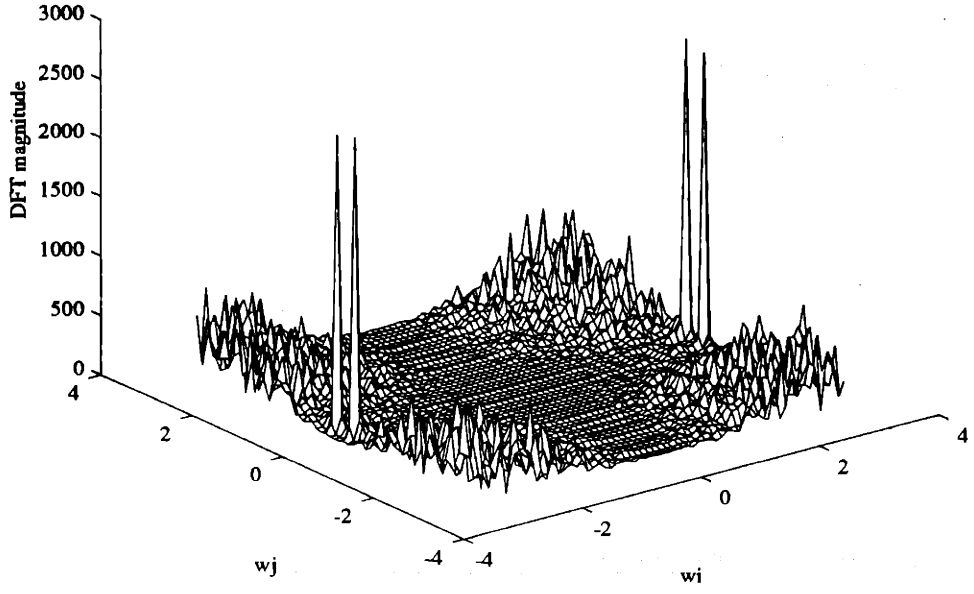

Figure 3-37: DFT of the "exact" output of the mixed filter driven by $u = 0.2u_3 + u_5$ with $N_i = I/31$ and $N_j = I/2$.

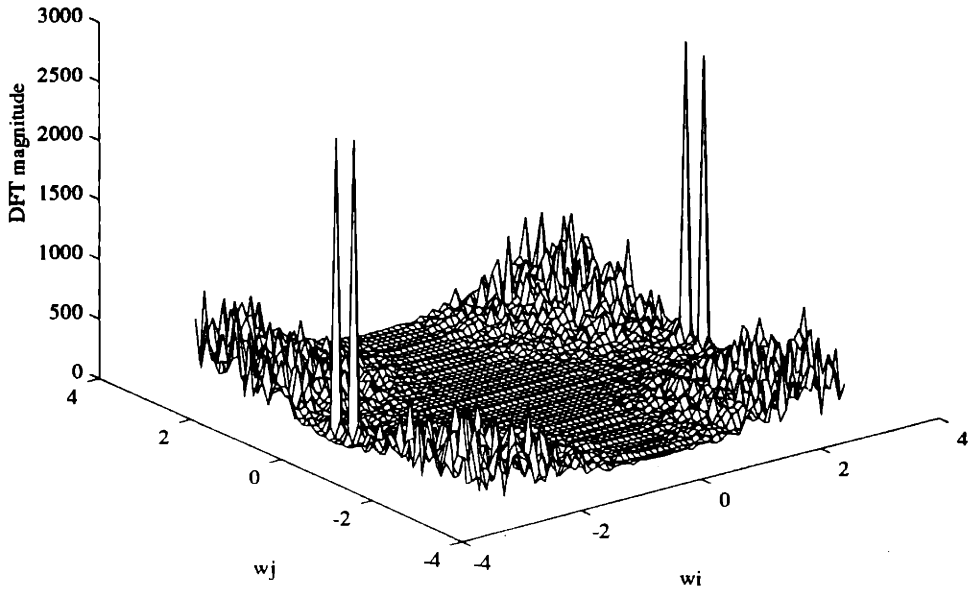

Figure 3-38: DFT of approximate output of the mixed filter driven by $u = 0.2u_3 + u_5$ with $N_i = I/31$ and $N_j = I/2$ with $B = 5$.

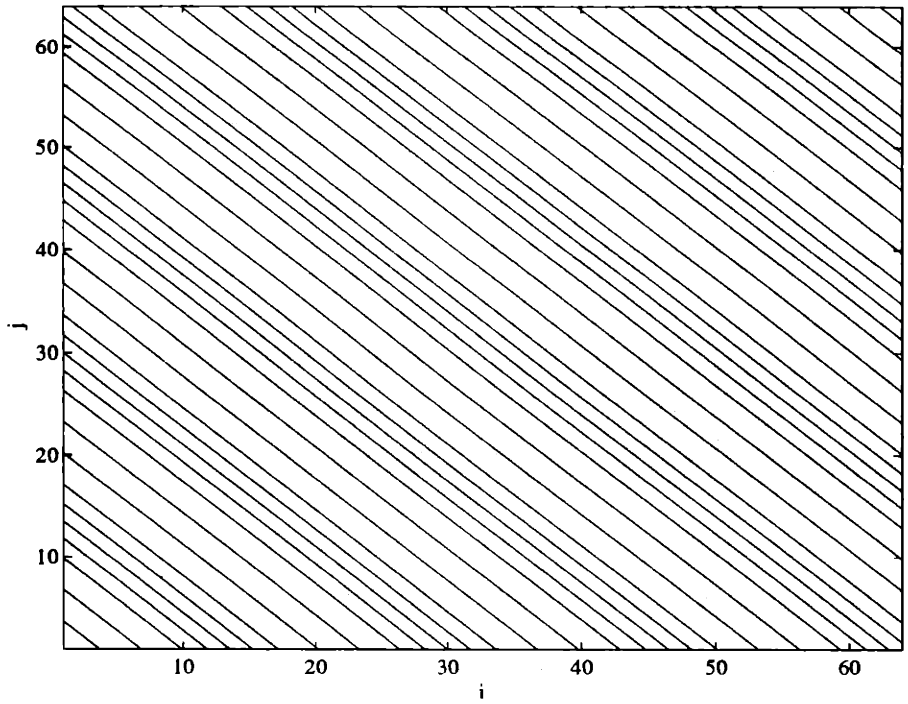

Figure 3-39: Contour of high-pass filter output (approximated with $B = 5$) when driven by the input $u = u_4$ with $N_i = N_j = I/2$.

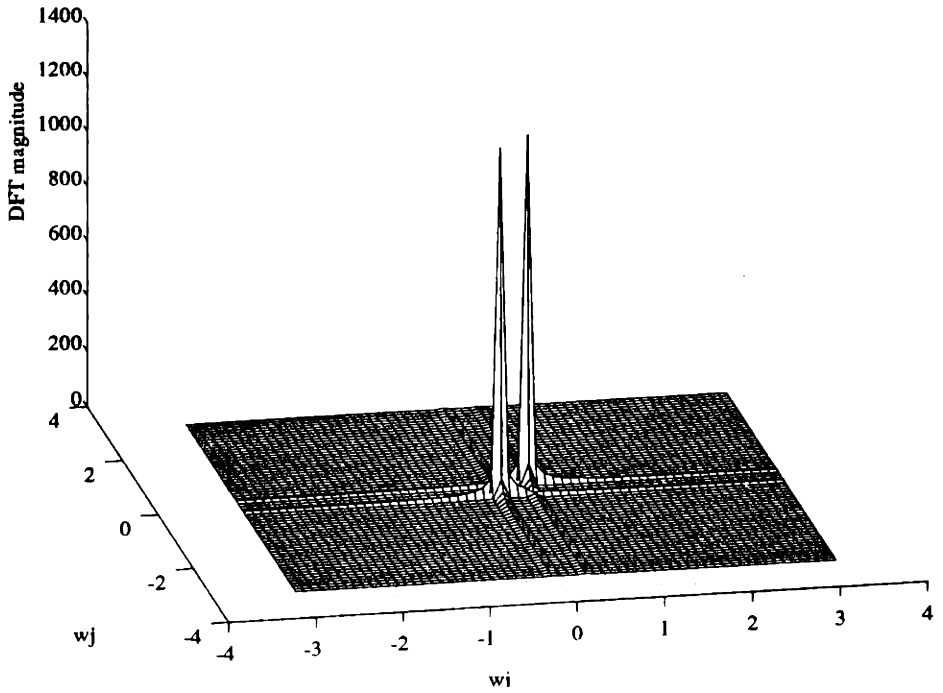

Figure 3-40: DFT of high-pass filter output (approximated with $B = 5$) when driven by the input $u = u_4$ with $N_i = N_j = I/2$.

by the amounts given by the Fourier Transform of the system function. However, the larger question of how the approximation errors vary with the bandwidth and size of the mesh has not been answered. The following example illustrates that errors in the approximate solution drop exponentially with the bandwidth of the approximation.

Fix the size of the filter domain to $\Omega = 128 \times 128$, and consider the low-pass filter with $\{n, s, e, w\} = \{.15, .15, .15, .15\}$ driven by the diagonal sinusoid with $N_i = N_j$. For sinusoidal inputs which vary between the pass and stop band of the low-pass filter, the error $d_2(x_1, x_2)$ is plotted in Figure 3-41 versus the bandwidth of the approximation. Note that the figure shows that in general, for the same bandwidth approximation, the errors are slightly more pronounced in the pass-band than in the stop-band. Yet the most striking conclusion to be drawn is that for any input the errors decrease exponentially with the size of the approximation bandwidth, and that very small errors can be obtained for bandwidths as low as $B = 3$. Similar results can be found for other diagonally dominant filters.

The final example gives a relationship among input domain size, the number of local factorization and inter-processor communication steps, and the approximation error. Varying the mesh from size $I = 32$ to $I = 256$, Table 3.5 gives the errors incurred with an approximation bandwidth of $B = 5$ when two filters are driven by the input $u = 0.2u_3 + u_{5_1} + u_{5_2}$

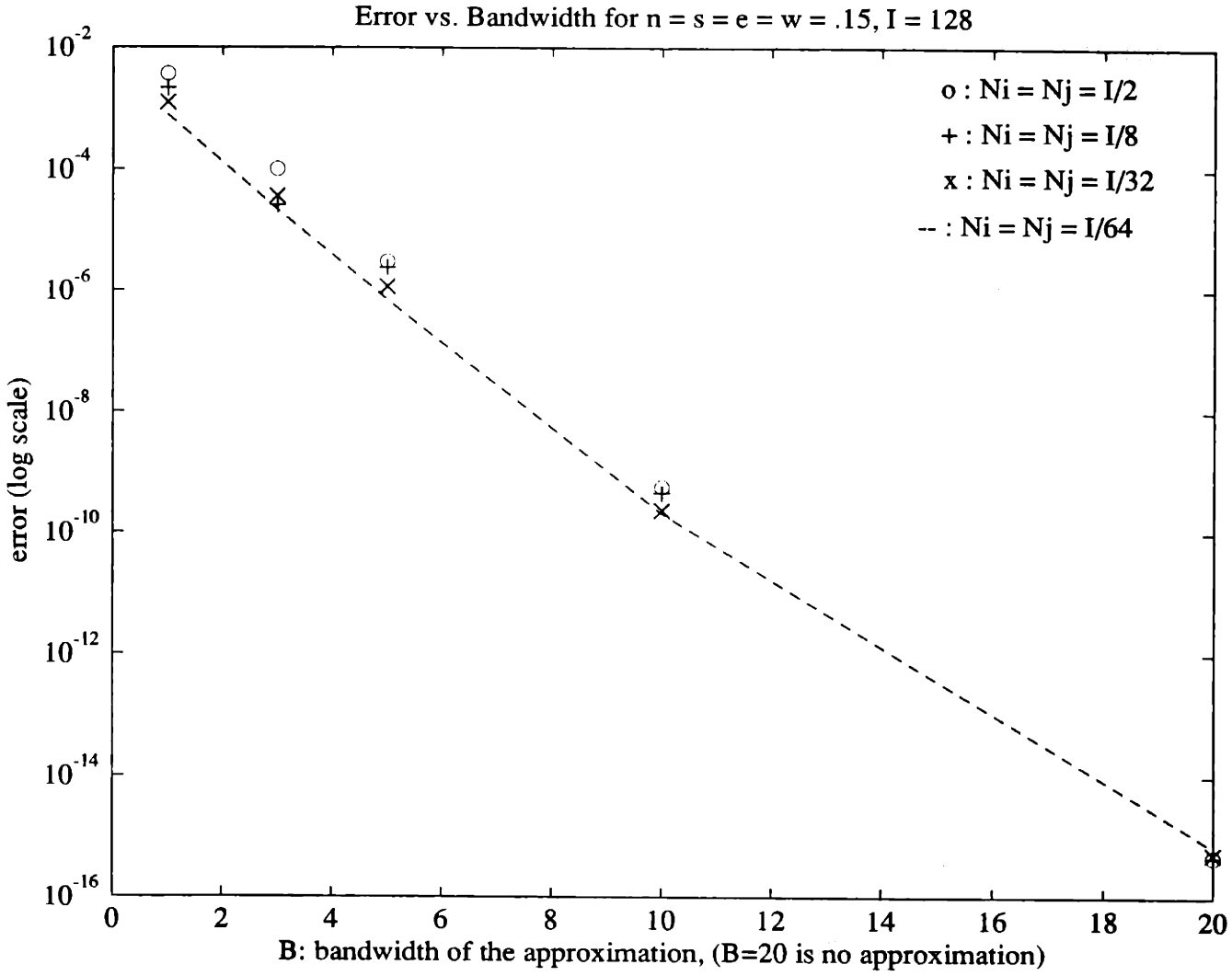

Figure 3-41: Log plot of $d_2(x_1, x_2)$ for the NNM $n = s = e = w = .15$.

NNM Parameters	I	M	$d_1(x_1, x_2)$	$d_2(x_1, x_2)$	$d_3(x_1, x_2)$
{.13,.18,.13,.18} Skewed LPF	32	0	0.0026	4.72e-06	5.88e-06
	61	0	0.0076	5.09e-06	8.19e-06
	32	2	0.0020	1.92e-06	3.08e-06
	64	2	0.0302	2.80e-06	4.41e-06
	128	2	0.0140	3.12e-06	5.84e-06
	256	2	0.0315	4.13e-06	6.44e-06
{-.15,-.15,-.15,-.15} HPF	32	0	0.0011	1.81e-06	3.24e-06
	61	0	0.0499	2.01e-06	2.08e-06
	32	2	0.0037	1.30e-06	3.32e-06
	64	2	0.0160	1.80e-06	3.24e-06
	128	2	0.0207	1.92e-06	5.19e-06
	256	2	0.0460	1.99e-06	2.40e-06

Table 3.5: Errors vs. I and M, which together determine the number of local and inter-processor factorization steps. $B = 5$.

where u_{5_1} is a rectangular sinusoid with $N_i = N_j = I/2$ and u_{5_2} is a rectangular sinusoid with $N_i = N_j = I/8$. One can see from Table 3.5 that as the mesh size varies from 32-by-32 to 256-by-256 the errors remain relatively constant. The large error given by d_1 can be shown to be due to a very small number of outliers at mesh locations where the output is small.

These examples have demonstrated the general numerical accuracy of the approximate algorithm for the second-order NNM filter, and also how the approximation improves with the degree of diagonal dominance and the bandwidth. In fact, even for small bandwidths, such as $B = 3$, the approximation produces a highly accurate solutions for meshes with large size and up to four partitions. Similar results should hold for a more finely partitioned mesh. It also seems reasonable that similar relationships hold for higher-order 2-D filters which exhibit diagonal dominance.

Chapter 4

Conclusions and Future Research

The objective of this work was to develop and investigate efficient, direct, and parallelizable algorithms for 2-D boundary value systems, particularly in the context of digital filtering. A few algorithms were discussed which appear to be provide a suitable framework for implementing 2-D IIR filters. One such algorithm, the mesh partitioning algorithm developed in Chapter 2, was shown to be similar in form and performance to nested dissection—a common algorithm used to solve P.D.E.'s and finite element applications. Although neither of these direct solutions to 2-D boundary value problems is a novel approach, using these algorithms to implement 2-D IIR filters is undoubtedly new to those in signal processing. These implementations are very efficient when the costs of the matrix factorization can be amortized over a large number of inputs. Furthermore, for those willing to make approximations, an approximate factorization requiring significantly less computations was given for both nested dissection and mesh partitioning.

The mesh partitioning algorithm developed in Chapter 2 is based upon the parallel estimation algorithms given in [37]. By partitioning the elements in a mesh of size I-by-I, one obtains an ordering of the matrix equations which allows for the factorization of the matrix in $O(I^3)$ computations requiring $O(I^3)$ storage elements. The examples of Chapter 3 and extensive simulations beyond those explicitly illustrated in this thesis show that the performance of the mesh partitioning algorithm is excellent for a large class of 2-D boundary value systems and a large number of inputs.

After developing the mesh partitioning algorithm in Chapter 2 and its approximation at the beginning of Chapter 3, the focus of this thesis then shifted nested dissection. Although

mesh partitioning appears to be very similar to nested dissection, nested dissection results in a different ordering of the system of equations which can be factored in $O(I^3)$ computations requiring $O(I^2 \ln I)$ storage elements. It is shown in [14, 13] that these asymptotic performance measures are optimal for a direct solution to any low-order boundary value system similar those defined in this thesis. Nested dissection is also easier to implement than mesh partitioning, since only an initial ordering of the equations is needed, yet it is unknown how well nested dissection performs when making approximations.

When one is willing to accept small errors in the solution, as is the case in many filtering applications, an approximate yet more efficient algorithm might be preferred. For difference equations resulting in diagonally dominant systems, it was shown in Chapter 3 for the mesh partitioning algorithm that approximations can be made during the factorization which significantly reduce the required number of computations. The mesh partitioning algorithm orders the variables along the mesh boundary last, meaning that they are the last variables eliminated. Therefore, boundary conditions are invoked only at the last step of the factorization, such that the property of diagonal dominance depends only upon the coefficients of the difference equation. To measure the accuracy of the approximation, the degree of diagonal dominance q (determined by the difference equation coefficients) and an approximation bandwidth B were defined in Chapter 3. The examples showed that, as q^{-1} decreases linearly to zero from one, the errors in the approximate solution decrease exponentially. Also, as B increases linearly, the errors decrease exponentially, while the computational complexity of the full matrix inversions within the mesh partitioning algorithm increase as B^2. The errors in the approximate solution were found to be robustly invariant to the form of the inputs and the size of the 2-D mesh.

These two direct factorizations, both with and without approximations, provide efficient solutions to low-order 2-D boundary value systems and are well-suited to cases in which the costs of the factorization can be amortized over a number of inputs. The approximate mesh partitioning algorithm applies to a smaller class of 2-D boundary value systems than those solved by mesh partitioning without approximations. Yet this class appears to include a number of important applications, such as physical oceanography [36] and 2-D signal processing. The examples focused on some simple 2-D signal processing applications in which the filters were implemented with NNM difference equations constrained by boundary conditions. The examples showed that the response of a filter to a pure sinusoid is a sinusoid

at the same frequency with attenuated magnitude and small distortions along the boundary. However, these distortions are small and become less noticeable when the size of the mesh increases. In other words, as the boundaries extend in all directions towards infinity, the boundary conditions appear to have less effect on the output values near the center of the mesh. This observation is very similar to the notion of stability given for 1-D two-point boundary value systems in [34], except that for the systems in [34] the dimension of the boundary systems remains constant as the size of the filter output region increases.

The filtering examples lead to two possible areas of future research. First, all of the frequency responses given in the examples were for diagonally dominant systems, a class whose system properties might account for the observed "decay" in the effect of the boundary conditions on the filter output. This "decay" is analyzed in [34] as a notion of stability for 1-D boundary value systems which have constant-coefficient dynamics with constant state dimension. One would like to extend this framework to 2-D boundary value systems, where the dimension of the boundary now grows with the size of the computed output. Such a framework would allow one to determine the effect of a 2-D IIR filter system's boundary conditions on the filter output for large mesh sizes. Secondly, 2-D IIR filter applications which are more complex than those found in this thesis call for boundary conditions more complex than the simple Neumann and Dirichlet conditions described in the examples. One could investigate how suitable boundary conditions can be chosen for general 2-D IIR filters. These two areas of research essentially seek to answer how appropriate direct implementations of 2-D IIR filters are for 2-D signal processing applications.

When extending the mesh partitioning and nested dissection algorithms to boundary value problems in higher dimensions, the computational complexity and storage requirements of the direct factorizations increase dramatically. For these reasons, iterative solutions are strongly preferred over direct solutions to 3-D boundary value problems [29, 30]. For a cube with $I \times I \times I$ elements, nested dissection requires $O(I^6)$ computations, which is an unreasonable number even for small cubes. Iterative solutions can be found requiring much less computational complexity. Even in 2-D, simple iterative schemes like Gauss-Jacobi and Gauss-Seidel [16] are guaranteed to converge for strictly diagonally dominant systems of equations (proven with a simple application of the Gerschgorin Circle Theorem), which then begs the question regarding 2-D IIR filters as to how the direct implementations given in this thesis compare to iterative implementations, such a Preconditioned Conjugate

Gradient methods [16]. One would also like to know how the 2-D IIR implementations compare to their FIR counterparts, which can be implemented efficiently with the 2-D FFT. Finally, the feasibility of implementing 3-D boundary value systems with the approximate direct algorithms could be investigated.

There are a few more questions which were raised but unanswered in the body of the thesis, yet could be the subject of future research. First, it was emphasized in Chapter 2 that for some difference equations the block factorization will fail due to singularities in the leading principle sub-matrices of the large matrix A in Equation (2.25). These systems are singular in the sense that the boundary conditions cannot be causally propagated through the dynamics due to singularities encountered during the propagation. The system's dynamics (see Equations (2.21)-(2.23)), however, are extremely difficult to analyze since the state has growing dimension. There has been a significant amount of research [28, 27, 34, 33, 32] for singular systems (often referred to as *Descriptor Systems*) when the system can be expressed as a constant-coefficient constant-state dimension dynamical system constrained at the boundaries. There has been little effort to describe and analyze singular systems with varying state dimension, so one could extend the stochastic theory given in [31] for 1-D boundary value sytems with time-varying state dimension to provide a deterministic framework for boundary value systems with varying state dimension.

Another area of possible research is to better understand the properties of the approximations made in Chapter 3. A first step would be to implement the approximation described in Section 3.2.1 for nested dissection and to compare the accuracy of this algorithm with that of the approximate mesh partitioning algorithm.

A better understanding of the approximations might lead to fast iterative solutions to elliptic P.D.E.'s. For diagonally dominant systems, one might use the solution obtained by an approximate direct factorization to $Ax = b$ to initialize an iterative algorithm. If x_{app} is the solution obtained by the approximate algorithm, then express x_{app} as

$$x_{app} = A_o^{-1} b \tag{4.1}$$

where we let $A_o = A - \Delta$. The matrix Δ represents the perturbation in the system due to the approximation. Noting that

$$(A_o + \Delta)x = b \tag{4.2}$$

the following iterative scheme can be derived

$$x_{k+1} = -(A_o^{-1}\Delta)x_k + A_o^{-1}b \tag{4.3}$$

$$x_0 = x_{app} \tag{4.4}$$

One must then obtain bounds on the spectral radius of the matrix $(A_o^{-1}\Delta)$ to see how fast the errors in the original estimate x_{app} decay to zero.

The approximate factorizations might also suggest an iterative solution to 2-D boundary value systems which are not diagonally dominant. Consider a boundary value system in which A is not diagonally dominant (or "just barely" diagonally dominant), yet can be expressed as $A = (A_{dd} + A_\delta)$, where A_{dd} is strictly diagonally dominant and A_δ is a relatively small perturbation. One has the relation

$$(A_{dd} + A_\delta)x = b \tag{4.5}$$

$$A = A_{dd} + A_\delta \tag{4.6}$$

where the system $A_{dd}x = b$ corresponds to a 2-D filter system in which A_{dd} can be factored efficiently by one of the approximate algorithms of Chapter 3. One can then obtain efficiently

$$x = -A_{dd}^{-1}A_\delta x + A_{dd}^{-1}b \tag{4.7}$$

which suggests the following iteration

$$x_{k+1} = -A_{dd}^{-1}A_\delta x_k + A_{dd}^{-1}b \tag{4.8}$$

These two iterative solutions are only suggestions which might lead to more interesting and efficient iterative solutions to low-order 2-D boundary value problems. These algorithms also might be of interest by themselves to the linear algebra community.

Appendix A

Coefficients for NNM with Radial Mesh Ordering

The form of the coefficient matrices D_ρ, F_ρ, G_ρ, and H_ρ in Equations (2.17)-(2.19) is described in this appendix for both 1-norm and ∞-norm radial orderings. With n_ρ given by (2.12) and (2.16) for scalar $x[i,j]$, the dimensions of the coefficient matrices are as follows:

$$D_\rho \in \Re^{n_\rho \times n_\rho} \tag{A.1}$$

$$G_\rho \in \Re^{n_{\rho+1} \times n_\rho} \tag{A.2}$$

$$F_\rho \in \Re^{n_{\rho-1} \times n_\rho} \tag{A.3}$$

$$H_\rho \in \Re^{n_\rho \times m_\rho} \tag{A.4}$$

$$H_\rho = diag\{B, B, \ldots, B\} \tag{A.5}$$

where m_ρ is the dimension of the input vector u_ρ. For inputs $u[i,j]$ equal in dimension to $x[i,j]$, m_ρ and n_ρ are equal. Also, due to the the time-invariance of the NNM, an operator * will be defined such that

$$\{N^*, S^*, E^*, W^*, I\} = \{S^T, N^T, W^T, E^T, I\} \tag{A.6}$$

For the purposes of this thesis, the effect of the * operation is that for a matrix Q whose elements consist only of N, S, E, W, I, and 0, then Q^* equals Q with N interchanged with

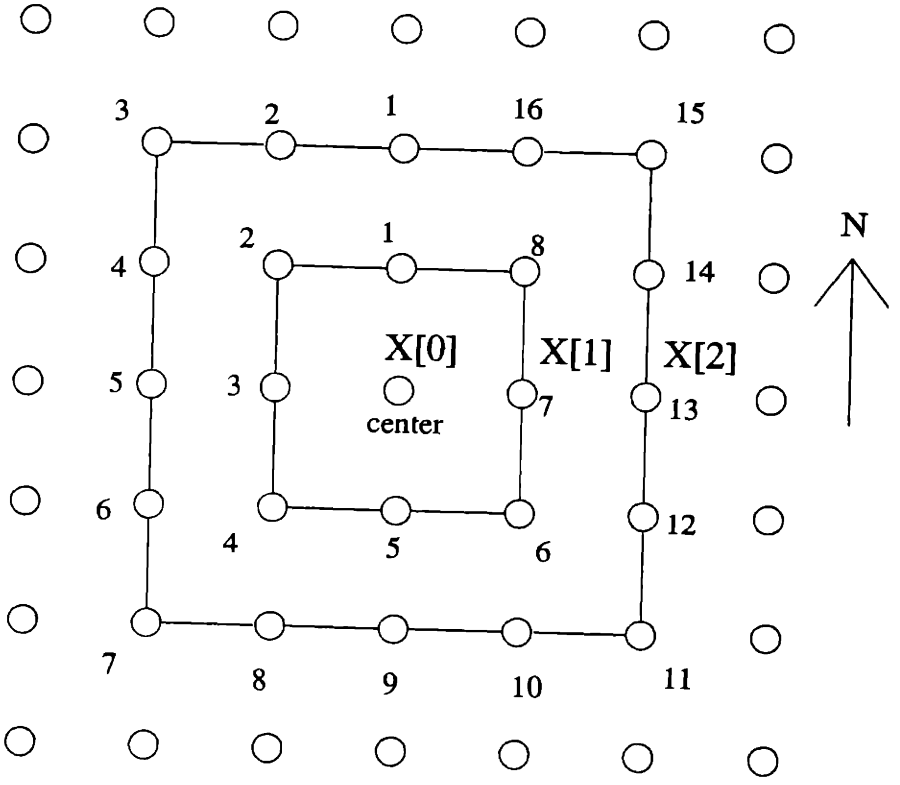

Figure A-1: A possible ordering of states x_0, x_1, and x_2 for an ∞-norm radial ordering of the mesh.

S, W interchanged with E, and all of the elements transposed. Equation (A.6) gives the relationship

$$F_{\rho+1} = \{G_\rho^T\}^* \tag{A.7}$$

which means that it is sufficient to specify the matrices F_ρ without explicitly describing G_ρ.

For ∞-norm orderings, in order to give a more detailed description of D_ρ and F_ρ, one must first specify the ordering of the mesh elements within x_ρ. The ordering assumed for the rest of this appendix is illustrated for x_1 and x_2 in figure A-1. Generalizing to larger states, the first element of x_ρ is the mesh variable at a radial distance ρ from the center which lies directly north of the center. The rest of the elements follow by stepping counter-clockwise around the radial square.

In the following matrices, without any loss of generality, only the scalar form of (2.2) is shown, and the capital NNM coefficients from Equation 2.2 become the scalars n, s, e, and

w. An ordering like that in figure A-1 gives the following matrices for $\rho = 0$, 1, and 2:

$$D_0 = 1 \tag{A.8}$$

$$D_1 = \begin{bmatrix} 1 & -w & 0 & 0 & 0 & 0 & 0 & -e \\ -e & 1 & -s & 0 & 0 & 0 & 0 & 0 \\ 0 & -n & 1 & -s & 0 & 0 & 0 & 0 \\ 0 & 0 & -n & 1 & -e & 0 & 0 & 0 \\ 0 & 0 & 0 & -w & 1 & -e & 0 & 0 \\ 0 & 0 & 0 & 0 & -w & 1 & -n & 0 \\ 0 & 0 & 0 & 0 & 0 & -s & 1 & -n \\ -w & 0 & 0 & 0 & 0 & 0 & -s & 1 \end{bmatrix} \tag{A.9}$$

$$D_2 = \begin{bmatrix} 1 & -w & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -e \\ -e & 1 & -w & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -e & 1 & -s & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -n & 1 & -s & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -n & 1 & -s & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -n & 1 & -s & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -n & 1 & -e & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -w & 1 & -e & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -w & 1 & -e & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -w & 1 & -e & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -w & 1 & -n & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -s & 1 & -n & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -s & 1 & -n & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -s & 1 & -n & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -s & 1 & -w \\ -w & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -e & 1 \end{bmatrix}$$

$$F_1 = \begin{bmatrix} n & 0 & w & 0 & s & 0 & e & 0 \end{bmatrix} \tag{A.10}$$

$$F_2 = \begin{bmatrix} n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & n & 0 & w & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & w & 0 & s & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & s & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & s & 0 & e & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & e & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & e & 0 & n \end{bmatrix} \tag{A.11}$$

$$G_0 = \begin{bmatrix} s \\ 0 \\ e \\ 0 \\ n \\ 0 \\ w \\ 0 \end{bmatrix} \tag{A.12}$$

$$G_1 = \begin{bmatrix} s & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & s & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & e & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & e & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & e & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & n & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & n & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & n & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & w & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & w & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & w \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & s \end{bmatrix} \tag{A.13}$$

Matrices corresponding to greater radial distances from the origin can be easily induced from those above, or one can refer to [37] for a algorithmic description of the matrices for arbitrary ρ.

With 1-norm radial orderings, $D_\rho = I$. F_ρ follows a similar a pattern similar to the F_ρ created by ∞-norms, and will be left to the reader to formulate. (Again, an algorithmic description of the matrices in $\tilde{A}$ for 1-norms is given in [37]). Note also that one can easily modify the coefficient matrices given in this appendix to represent an NNM difference equation like that in (2.2), but with time-varying coefficients.

Appendix B

NNM Parameters Causing Singularities

As is mentioned in Chapter 2 the block LU factorization exists if and only if the leading principal sub-matrices of A in Equation (2.25)

$$A_\rho = \begin{bmatrix} D_0 & -F_1 & 0 & 0 & \cdots & 0 \\ -G_0 & D_1 & -F_2 & 0 & \cdots & 0 \\ 0 & -G_1 & D_2 & -F_3 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -G_{\rho-2} & D_{\rho-1} & -F_\rho \\ 0 & \cdots & 0 & 0 & -G_{\rho-1} & D_\rho \end{bmatrix} \tag{B.1}$$

are non-singular for all $\rho = 0, \ldots, R=I/2$. It can be shown that if the leading principal sub-matrix A_ρ in nonsingular, then $\tilde{D}_\rho$ is also nonsingular. Namely, if the Block LU algorithm has progressed for $(\rho-1)$ steps without having encountered a singularity, then $\tilde{D}_\rho$ will be singular if and only if A_ρ is singular. This claim is proven as follows: first, express A_ρ in block form as

$$A_\rho = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ 0 & U_{22} \end{bmatrix}$$

where

$$A_{11} = A_{\rho-1} \tag{B.2}$$

$$A_{21} = [0, 0, \ldots, -G_{\rho-1}] \tag{B.3}$$

$$A_{12} = \left[0, 0, \ldots, -F_{\rho}^{T}\right]^{T} \tag{B.4}$$

$$A_{22} = D_{\rho} \tag{B.5}$$

$$U_{22} = A_{22} - A_{21} A_{11}^{-1} A_{12} \tag{B.6}$$

U_{22} is known as the Schur Complement [12] of this block factorization. Since A_{11} is non-singular, then the matrix U_{22} is singular if and only if A_{ρ} in singular. The matrix U_{22}, however, is equal to $\tilde{D}_{\rho}$, as can be shown by making the proper substitutions into (B.6) and comparing it with (2.28), (which concludes the proof).

Taking the determinant of A_{ρ} in (B.1) and setting it equal to zero allows one to solve for the model parameters at which the block LU algorithm will fail due to a singularity in $\tilde{D}_{\rho}$. In this section, the analysis is limited to ∞-norm radial orderings, although one could painstakingly derive similar results for 1-norm orderings. Using Appendix A, one can see that the determinant will be solely in terms of the NNM parameters, n, s, e, and e, and is independent of the inputs, boundary conditions, and input parameter b. As would be expected from the symmetric structure of the NNM, the determinant of A_{ρ} is symmetric about the values of n and s, and about e and w, such that

$$det\{A_{\rho}\} = f_{\rho}(n, s, e, w) = f_{\rho}(s, n, e, w) = f_{\rho}(n, s, w, e)$$

Furthermore, setting the polyvariate determinant $f_{\rho}(n, s, e, w)$ equal to zero yields roots in the form

$$ns = g_{\rho}((ew)^{1/2}) \tag{B.7}$$

$$ew = g_{\rho}((ns)^{1/2}) \tag{B.8}$$

where $g_{\rho}(x)$ is a second-order polynomial in x. Specifying the roots of the determinant of A_{ρ} is, therefore, identical to specifying the all of the functions $g_{\rho}(x)$. Let S_{ρ} be the set of all functions $g_{\rho}(x)$. From the set S_{ρ} one can then determine all of the NNM parameters which cause the matrix $\tilde{D}_{\rho}$ to be singular. With some symbolic manipulation (using the application MAPLE), the sets S_{ρ} for $\rho = 0, \ldots, 4$ are shown in Equations (B.9)-(B.13), where S_0 is obviously the null set since $D_0 = 1$. The roots then follow from (B.7) and (B.8).

$$S_0 = \phi \tag{B.9}$$

$$S_1 = \left(\frac{1}{2}, \quad \frac{2x^2 \pm 2\sqrt{2}x + 1}{2} \right) \tag{B.10}$$

$$S_2 = \left(\frac{1}{3}, \quad 1, \quad 3x^2 \pm 2\sqrt{3}x + 1, \quad x^2 \pm 2x + 1, \quad \frac{3x^2 \pm 2\sqrt{3}x + 1}{3}, \quad \frac{x^2 \pm 2x + 1}{3} \right) \tag{B.11}$$

$$S_3 = \left(\begin{array}{cc} \frac{1}{2}, & \frac{2 \pm \sqrt{2}}{2}, \\ \frac{2x^2 \pm 2\sqrt{2}x + 1}{2}, & \\ \frac{(6+4\sqrt{2})x^2 \pm 2\sqrt{2}\sqrt{10+7\sqrt{2}}x + (2+\sqrt{2})}{2}, & \frac{(6-4\sqrt{2})x^2 \pm 2\sqrt{2}\sqrt{10-7\sqrt{2}}x + (2-\sqrt{2})}{2}, \\ \frac{(2+\sqrt{2})x^2 \pm 2\sqrt{2+\sqrt{2}}x + 1}{2}, & \frac{(2-\sqrt{2})x^2 \pm 2\sqrt{2-\sqrt{2}}x + 1}{2}, \\ \frac{2x^2 \pm 2\sqrt{2}\sqrt{2+\sqrt{2}}x + 2 + \sqrt{2}}{2}, & \frac{2x^2 \pm 2\sqrt{2}\sqrt{2-\sqrt{2}}x + 2 - \sqrt{2}}{2}, \\ \frac{(4+2\sqrt{2})x^2 \pm (4+4\sqrt{2})x + 2 + \sqrt{2}}{2}, & \frac{(4-2\sqrt{2})x^2 \pm (4-4\sqrt{2})x + 2 - \sqrt{2}}{2}, \end{array} \right) \tag{B.12}$$

$$S_4 = \left(\begin{array}{cc} \frac{3 \pm \sqrt{5}}{2}, & \frac{5 \pm \sqrt{5}}{10} \\ \frac{10x^2 \pm 2\sqrt{2}\sqrt{5+\sqrt{5}}\sqrt{5}x + 5 + \sqrt{5}}{10}, & \frac{10x^2 \pm 2\sqrt{2}\sqrt{5-\sqrt{5}}\sqrt{5}x + 5 - \sqrt{5}}{10} \\ \frac{(10+4\sqrt{5})x^2 \pm 2\sqrt{2}\sqrt{25+11\sqrt{5}}x + 3 + \sqrt{5}}{2}, & \frac{(10-4\sqrt{5})x^2 \pm 2\sqrt{2}\sqrt{25-11\sqrt{5}}x + 3 - \sqrt{5}}{2} \\ \frac{(5+\sqrt{5})x^2 \pm 4\sqrt{5+2\sqrt{5}}x + 3 + \sqrt{5}}{2}, & \frac{(5-\sqrt{5})x^2 \pm 4\sqrt{5-2\sqrt{5}}x + 3 - \sqrt{5}}{2} \\ \frac{(15+5\sqrt{5})x^2 \pm 4\sqrt{5+2\sqrt{5}}\sqrt{5}x + 5 + \sqrt{5}}{10}, & \frac{(15-5\sqrt{5})x^2 \pm 4\sqrt{5-2\sqrt{5}}\sqrt{5}x + 5 - \sqrt{5}}{10} \\ \frac{2x^2 \pm 2\sqrt{2}\sqrt{3+\sqrt{5}}x + 3 + \sqrt{5}}{2}, & \frac{2x^2 \pm 2\sqrt{2}\sqrt{3-\sqrt{5}}x + 3 - \sqrt{5}}{2} \\ \frac{(5+\sqrt{5})x^2 \pm 4\sqrt{5}x + 5 - \sqrt{5}}{10}, & \frac{(5-\sqrt{5})x^2 \pm 4\sqrt{5}x + 5 + \sqrt{5}}{10} \\ \frac{(10+4\sqrt{5})x^2 \pm (10+6\sqrt{5})x + 5 + \sqrt{5}}{10}, & \frac{(10-4\sqrt{5})x^2 \pm (10-6\sqrt{5})x + 5 - \sqrt{5}}{10} \\ \frac{(7+3\sqrt{5})x^2 \pm (8+4\sqrt{5})x + 3 + \sqrt{5}}{2}, & \frac{(7-3\sqrt{5})x^2 \pm (8-4\sqrt{5})x + 3 - \sqrt{5}}{2} \end{array} \right) \tag{B.13}$$

The ideal results from the symbolic manipulation would be that the collection of sets S_ρ for all ρ is finite, or that a discernible pattern emerges where S_ρ could be described in closed form as a function of ρ. For the moment, however, perhaps due to a lack of insight and motivation, specific comments can only be made with respect to the sets above. A pattern does appear to be emerging, and finding it might be the subject of future endeavors. However, one can use 1-norm radial orderings or alternative algorithms for those NNM parameters in which ∞-norm orderings lead the Block LU algorithm to failure. Also, the four sets S_1-S_4 have proven empirically to be an excellent and surprisingly comprehensive guideline for weeding out those NNM filters ill-suited for the block LU algorithm. For these two reasons, this issue is not pursued any further.

Bibliography

[1] R. E. Bank. *Sparse Matrix Computations.* Academic Press, New York, 1976. pp. 293-307.

[2] R. E. Bank. Marching algorithms for elliptic boundary value problems. *SIAM Journal on Numerical Analysis*, 14:792–829, Oct. 1977.

[3] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[4] A. C. Bovik, M. Clark, and W. S. Geisler. Multichannel texture analysis using localized spatial filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1990.

[5] B. L. Buzbee, G. H. Golub, and C. W. Nielson. On direct methods for solving Poisson's equations. *SIAM Journal on Numerical Analysis*, 1970.

[6] M. Clark and A.C. Bovik. Texture segmentation using gabor modulation demodulation. *Pattern Recognition Letters*, 1987.

[7] M. Y. Dabbagh and W. E. Alexander. Multiprocessor implementation of 2-d denominator separable digital filters for real-time processing. *IEEE Transactions on Acoustics and Signal Processing*, June 1989.

[8] H. Derin and P. A. Kelly. Discrete-index Markov-type random processes. *IEEE Proceedings*, 1989.

[9] Fred W. Dorr. The direct solution of the discrete Poisson equation on a rectangle. *SIAM Review*, 1970.

[10] Dan E. Dudgeon and Russell M. Mersereau. *Multidimensional digital signal processing.* Prentice Hall, Englewood Cliffs, NJ, 1984.

[11] David G. Feingold and Richard S. Varga. Block diagonally dominant matrices and generalizations of the gerschgorin circle theorem. *Pacific Journal of Math*, 1962.

[12] F. R. Gantmacher. *Matrix Theory*, volume 1. Chelsea Publishing, NY, NY, 1959.

[13] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 1973.

[14] Alan George and Joseph W. Liu. *Computer Solution of Large and Sparse Positive Definite Systems*. Prentice Hall, Englewood Cliffs, NJ, 1981.

[15] Alan George and Joseph W. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 1989.

[16] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. John Hopkins, Baltimore, 1990.

[17] Don Heller. Some aspects of the cyclic block reduction algorithm for block tridiagonal linear systems. *SIAM Journal on Numerical Analysis*, 1976.

[18] Berthold K.P. Horn. *Robot Vision*. MIT Press, Cambridge, MA, 1987.

[19] Mathworks Inc. *Matlab User's Guide*. Natick, MA, 4.0 edition, August 1992.

[20] Anil K. Jain. Partial differential equations and finite-difference methods in image processing, part 1: image representation. *Journal of Optimization Theory and Applications*, 1977.

[21] Anil K. Jain. Partial differential equations and finite-difference methods in image processing, part II: image restoration. *IEEE Transactions on Automatic Control*, 1978.

[22] Anil K. Jain. Advances in mathematical models for image processing. *Proceedings of the IEEE*, 1981.

[23] A. J. Krener. Acausal realization theory, part i: Linear deterministic systems. *J. Control Optimiz.*, 1987.

[24] Bernard C. Levy, Milton B. Adams, and Alan S. Willsky. Solution and linear estimation of 2-d nearest-neighbor models. *Proceedings of the IEEE*, 1990.

[25] Jae S. Lim. *Two-dimensional signal and image processing.* Prentice Hall, Englewood Cliffs, NJ, 1990.

[26] Xiaojian Liu and Alfred Fettweis. Multidimensional digital filtering by using parallel algorithms based on diagonal processing. *Mulitidimensional systems and signal processing*, 1990.

[27] David G. Luenberger. Dynamic equations in descriptor form. *IEEE Transactions on Automatic Control*, 1977.

[28] David G. Luenberger. Time-invariant Descriptor Systems. *Automatica*, 1978.

[29] Keith Nabors, Songmin Kim, and Jacob White. Fast capacitance extraction of general 3-d structures. *IEEE Transactions on Microwave Theory and Techniques*, 40(7), July 1992.

[30] Keith Nabors, T. Korsmeyer, and Jacob White. Fast capacitance extraction of general 3-d structures. *Submitted to SISSC*, 1992.

[31] R. Nikoukhah, A. S. Willsky, and B. C. Levy. Kalman filtering and riccati equations for descriptor systems. *IEE Transactions on Automatic Control*, 1992.

[32] Ramine Nikoukhah. Boundary-value descriptor systems: well-posedness, reachability and observability. *International Journal of Control*, 1987.

[33] Ramine Nikoukhah. A deterministic and stochastic theory for Two-Point Boundary-Value Descriptor Systems. *LIDS-TH-1820*, 1988.

[34] Ramine Nikoukhah, Bernard C. Levy, and Alan S. Willsky. Stability, stochastic stationarity, and generalized Lyapunov equations for Two-Point Boundary-Value Descriptor Systems. *IEEE Transactions on Automatic Control*, 1989.

[35] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-time signal processing.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[36] Jens Schroter and Carl Wunsch. Solution of nonlinear finite difference ocean models by optimization methods with sensitivity and observational strategy analysis. *Journal of Physical Oceanography*, 16(11), November 1986.

[37] Darrin Taylor. Parallel estimation on one and two dimensional systems. *LIDS-TH-2092*, 1992.

[38] A. H. Tewfik, B. C. Levy, and A. S. Willsky. Parallel smoothing. *Systems and control letters*, 1990.

[39] J. M. Varah. On the solution of block-tridiagonal systems arising from certain finite-difference equations. *Mathematics of Computation*, 1972.

[40] Richard S. Varga. *Matrix Iterative Analysis.* Prentice Hall, Englewood Cliffs, NJ, 1972.

[41] J. W. Woods. Two-dimensional discrete Markovian fields. *IEEE Transactions on Information Theory*, 1972.