INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9432613

Domain decomposition algorithms and parallel computation techniques for the numerical solution of PDE's with applications to the finite element shallow water flow modeling

Cai, Yihong, Ph.D.

The Florida State University, 1994

Copyright ©1994 by Cai, Yihong. All rights reserved.

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

THE FLORIDA STATE UNIVERSITY
COLLEGE OF ARTS AND SCIENCES

DOMAIN DECOMPOSITION ALGORITHMS AND PARALLEL COMPUTATION TECHNIQUES FOR THE NUMERICAL SOLUTION OF PDE'S WITH APPLICATIONS TO THE FINITE ELEMENT SHALLOW WATER FLOW MODELING

By

YIHONG CAI

A Dissertation submitted to the
Department of Mathematics
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Degree Awarded:
Summer Semester, 1994

Copyright © 1994
Yihong Cai
All Rights Reserved

The members of the Committee approve the dissertation of Yihong Cai defended on June 13, 1994.

I. Michael Navon
Professor Directing Dissertation

James J. O'Brien
Outside Committee Member

Louis N. Howard
Committee Member

David A. Kopriva
Committee Member

Eutiquio C. Young
Committee Member

To my lovely fiancée Dr. Xiang Yang Yu, my beloved mother Xiuxia Huang, father Xiaota Cai, sisters Ningning, Xiaoning and brother Yijian.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to Prof. Michael Navon, my dissertation advisor, for his tireless guidance, encouragement, support and trust. I have greatly benefited, for the entire period of my Ph.D. study and research, from his thorough knowledge, innovative ideas and deep insight in both numerical analysis and scientific computing. He has constantly served as a source of novel ideas and has always led me in the right direction. Without his unending advice and dedication to science and education, I can hardly imagine that I could have completed several research projects in these new and exciting areas of domain decomposition and parallel computing. His influence on me will certainly play an important role in my future professional career.

I feel very much obliged to Prof. David Kopriva and Dr. Thomas Oppe for so many helpful professional discussions as well as unforgettable personal conversations and encouragement. I am deeply impressed by their knowledge in numerical analysis and applied sciences.

I am grateful to Prof. David Keyes (ICASE, NASA Langley Research Center) for his invaluable comments on the early draft of Chapter 6 during my visit to ICASE in February, 1994.

This research is not possible without the love, understanding and tolerance of my fiancée Dr. Xiang Yang Yu and my family. They have been making my life so interesting, wonderful and secure. I would also like to express my deepest appreciation to my two aunts, Prof. Siu-Chi Huang and Mrs. Siu-Lien Tsung. They are so special and have been giving me and my family so much love. Ever since I was a college student, Prof. Siu-Chi Huang has been my constant source of encouragement and moral support. She shared with me her experience gained from over forty years of teaching and research in this country. Her personal experience and motherly advice will be very helpful to my future career in science.

It is my pleasure to acknowledge the grant support AFOSR-89-0162 through the Geophysical Fluid Dynamics Institute (GFDI) of the Florida State University.

the support of US DOE contracts # 120054323 and DE-FC05-85ER250000 through the Supercomputer Computations Research Institute (SCRI) of the Florida State University. The challenging atmosphere, excellent facilities and friendly people in SCRI have made my research here really enjoyable. I think I owe SCRI a debt which I can never repay.

I also want to acknowledge CA-DISSPLA graphics software package, version 11.0. All figures in this dissertation were produced by this sophisticated software package. CA-DISSPLA is a registered trademark of Computer Associates International, Inc.

Finally, I wish to thank Prof. James O'Brien, Prof. Louis Howard, Prof. David Kopriva and Prof. Eutiquio Young for their willingness to serve on my doctoral committee.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The parallel numerical solution of partial differential equations (PDE's) has been a very active research area of numerical analysis and scientific computing for the past two decades. However, most of the recently developed parallel algorithms for the numerical solution of PDE's are largely based on, or closely related to, domain decomposition principles.

In this dissertation, we focus on (1) improving the efficiency of some iterative domain decomposition methods, (2) proposing and developing a novel domain decomposition algorithm, (3) applying these algorithms to the efficient and cost effective numerical solution of the finite element discretization of the shallow water equations on a 2-D limited area domain and (4) investigating parallel implementation issues.

We have closely examined the iterative Schur domain decomposition method. The Schur domain decomposition algorithm, described in detail in the present dissertation, may be heuristically viewed as an iterative "divide and feedback" process representing interactions between the subdomains and the interfaces. A modified version of the rowsum preserving interface probing preconditioner is proposed to accelerate this process. The algorithm has been successfully applied to the solution of linear systems of algebraic equations, resulting from the finite element discretization, which couple the discretized geopotential and velocity field variables at each time level. A node renumbering scheme is also proposed to facilitate modification of an existing serial code, especially the one which is based on the finite element discretization, into a non-overlapping domain decomposition code.

In the Schur domain decomposition method, obtaining the numerical solutions on the interfaces usually requires repeated exact subdomain solutions, which are not cheaply available for our problem and many other practical applications. In view of this, the modified interface matrix domain decomposition algorithm is proposed and developed to reduce computational complexity. The algorithm starts with an initial guess on the interfaces and then iterates back and forth between the subdomains and the interfaces. Starting from the second outer iteration, it becomes increasingly less expensive to obtain solutions on the subdomains and interfaces due to the availability of successively improved initial solutions from the previous outer iteration. The numerical results obtained by applying this algorithm to our problem improve upon those obtained by employing the traditional Schur domain decomposition algorithm.

We then investigate parallel block preconditioning techniques in the framework of three frequently used and competitive non-symmetric linear iterative solvers, namely, generalized minimal residual (GMRES), conjugate gradient squared (CGS) and Bi-CGSTAB (a variant of bi-conjugate gradient method) algorithms. Many hybrid methods of non-overlapping domain decomposition result from various combinations of linear iterative solvers and domain decomposed (DD) preconditioners (generally consisting of inexact subdomain solvers and an interface preconditioner). Two types of existing DD preconditioners are employed and a novel one is proposed to accelerate the convergence of GMRES, CGS and Bi-CGSTAB. The newly proposed third type of DD preconditioners turns out to be computationally the least expensive and the most efficient for solving the problem addressed in this dissertation, although the second type of DD preconditioners is quite competitive. Performance sensitivities of these preconditioners to inexact subdomain solvers are also investigated.

Parallel implementation issues of domain decomposition algorithms are then discussed. Moreover, a multicolor numbering scheme is described and applied to the parallel assembly of elemental contributions, aimed at removing critical regions and minimizing the number of synchronization points in the finite element assembly process. Typical parallelization results on the CRAY Y-MP are presented and discussed.

This dissertation also contains a relatively thorough review of two fast growing areas in computational sciences, namely, parallel scientific computing in general and iterative domain decomposition methods in particular. A discussion concerning possible future research directions is provided at the end of the last chapter.

CHAPTER 1

INTRODUCTION

> ...WANTED for Hazardous Journey. Small wages, bitter cold, long months of complete darkness, constant danger, safe return doubtful. Honor and recognition in case of success.
>
> — Ernest Shackleton[1]

The commercial availability of high-speed, large-memory computers which began to emerge over a decade ago has made possible the solution of a rich variety of increasingly complex large-scale scientific and engineering problems. For efficient and cost effective utilization of these high performance computing facilities which offer, as peak performances, several hundred millions of floating point operations per second (Mflops) or even a few Gflops (10^3 Mflops), one has to revisit and adapt many of the extant serial numerical algorithms and research further into novel parallel methods, algorithms, data structures and languages which are well suited for the new generation of supercomputers[2].

There is an ever increasing demand for high performance computers in the areas of computational fluid dynamics, aerodynamics simulations, fusion energy research,

[1] From a newspaper advertisement for an Antarctic Expedition.

[2] Supercomputers are loosely defined as the fastest computing machines at any given time.

military defense, elastodynamics, weather prediction, large-scale structural analysis, petroleum exploration, computer aided design, industrial automation, medical diagnosis, artificial intelligence, expert systems, remote sensing, genetic engineering and even socioeconomics.

The past several decades have witnessed a rapid development of various numerical methods whose algorithmic implementation was designed to fit the architecture of a single processor serial computer. Although the development of new generation of computing, namely, parallel computing, has already taken off the ground, the research in this area is much less mature compared to serial computing. Indeed, the area of parallel computing is in a state of flux and the marketplace for high performance parallel computers is volatile.

Although large-scale scientific and engineering computing is the major driving force for the design and development of multi-processor architectures, the recent explosion of research activities in the area of parallel computation is largely motivated by the commercial availability of various powerful high performance computers. It has been a great challenge for numerical analysts and computational scientists to design efficient numerical algorithms and develop suitable programming languages that can fully exploit and utilize the potential power of such advanced computing architectures.

One of the research focuses in the area of parallel computing has centered on the issue of how to cost-effectively introduce parallelism into very strongly coupled problems, such as the parallel solution of very large linear or non-linear systems of algebraic equations, which arise from the finite difference or finite element discretization of PDE's in solid mechanics, fluid dynamics and many other areas of industrial applications. Numerous approaches (see, for example, [66], [94], [115], [117], [185], [188] and [189]) have already been developed and implemented on different types

of parallel computers. However, most of the parallel numerical algorithms recently proposed for this purpose are largely based on, or closely related to, the principles of domain decomposition.

Many of the so-called iterative domain decomposition methods are just organizing principles proposed to effectively decouple the system of algebraic equations, corresponding to which there is an underlying continuous physical problem abstracted in the form of PDE's. The term "domain decomposition" stems from the fact that these smaller decoupled algebraic systems correspond to the discretization of the original differential operators restricted to the subdomains of the original given domain.

Domain decomposition techniques have been receiving great attention in the areas of numerical analysis and scientific computing mainly due to their potential for parallelization. However, we note that the usefulness of domain decomposition extends well beyond the readily apparent issue of parallel computing. In fact, domain decomposition algorithms are well suited for carrying out locally adaptive mesh refinement and for taking advantage of the fast direct solvers which may only be locally exploitable for problems defined on irregular regions. The flexibility of domain decomposition methods makes it less difficult to incorporate different mesh resolutions or numerical methods on different parts of the original physical domain and to couple different mathematical models defined on different subdomains whenever the physics behind the problem has a variable nature therein. We will discuss some of these issues in detail and provide relevant references in Chapter 3, although the primary motivation of employing domain decomposition methods in this dissertation work is related to parallelization concerns.

For the past eight years, there has been a sizable amount of research on various domain decomposition techniques for second-order self-adjoint linear scalar ellip-

tic PDE's. Great progress has been made in this direction and some optimal or nearly optimal methods have been developed (see [130] for the most recent review of this fast-growing area). The most often used mathematical tools for analyses are the Galerkin finite element formulation, subspaces of functions, projection theories, multilevel and Krylov methods. However, both the theory and numerical experience with non-self-adjoint elliptic PDE's are much less satisfactory. Little work has been carried out in devising domain decomposition methods for the solution of linear or nonlinear systems of algebraic equations arising from the finite difference or finite element approximation of the hyperbolic PDE's.

In this dissertation, we are mostly concerned with the extension of domain decomposition ideas to the finite element solution of a set of coupled hyperbolic PDE's, namely, the shallow water equations and to the practical issue of parallelization. Many successful analysis methods for elliptic problems are not directly applicable here. Some extensions have yet to be made. The work contained in this dissertation represents one of the first attempts in applying domain decomposition principles to the hyperbolic equations, especially to the shallow water equations. Direct numerical experience indicates that some of the domain decomposition algorithms proposed for elliptic problems may also apply successfully to the hyperbolic PDE's. Apparently, a vast amount of theoretical studies and numerical experiments still need to be carried out in this direction.

As part of this dissertation, an overview of two fast growing areas, namely, parallel computing in general and domain decomposition in particular, is absolutely necessary. We will review the past research efforts, report what we have done up to this point and look into future research directions.

No attempt was, however, made to give a comprehensive review due to the huge amount of work already having been done in these areas and the broad sense of par-

allel computing, e.g., parallel image processing, parallel pattern matching, parallel matrix computations, parallel structural analysis, parallel numerical optimization, to mention just a few. Different domain decomposition approaches, many possible combinations of relevant techniques and various subtle implementation details on different parallel environments have already led to a plethora of the so-called iterative domain decomposition algorithms. In view of this, we will concentrate instead on an overview and discussions of some important terms and concepts, some novel and challenging issues in the design of parallel numerical algorithms for scientific computation, programming aspects and the performance evaluation for parallel implementations, as well as on issues closely related to the parallel numerical solution of partial differential equations and some well-known domain decomposition algorithms.

Specifically, in Chapter 2, we will briefly present some historical aspects and developments of parallel computers and parallel scientific computing, explain the motivation behind these developments and analyze architectural features and helpful classifications of some currently commercially available multi-processor systems. Through some examples, we emphasize that parallel computing has brought in many new and challenging issues one need not consider for serial computing. In particular, we point out that the quality of a parallel numerical algorithm can no longer be measured by the classical analysis of computational complexity alone. Equally important, we have to take into account such issues as the degree of parallelism in the algorithm, communication, synchronization and the locality of reference within the code. Performance analysis and measurements for parallelization are also briefly discussed. Many relevant references are furnished for those who want to explore further for subtle details.

Starting from Chapter 3, we will focus exclusively on a relatively new and promising branch of parallel numerical methods — domain decomposition, the main topic of this dissertation. We introduce domain decomposition ideas, in Chapter 3, by considering solving a classical elasticity problem, namely, the famous Saint Venant's torsion of a cylindrical shaft with an irregular cross-sectional shape. We then argue that domain-based decomposition is the best among three possible decomposition strategies for the parallel numerical solution of PDE's. Three specific domain decomposition methods developed for solving elliptic PDE's, namely, multiplicative Schwarz, additive Schwarz and iteration-by-subdomain domain decomposition methods are presented and discussed in some detail. Origins and motivations of domain decomposition are also given in this chapter.

Chapter 4 consists of a detailed study of the Schur domain decomposition method and its application to the finite element numerical simulation of the shallow water flow. Two Schur domain decomposition algorithms are presented. Various preconditioning techniques are described. The efficiency of the Schur domain decomposition method largely depends on the effectiveness of a preconditioner on the interfaces. To accelerate the convergence of the Schur complement linear system on the interfaces, we employ the traditional rowsum preserving interface probing preconditioner and also propose a modified version, which is shown to be better than the traditional one. A node renumbering scheme is also proposed in this chapter to facilitate the modification of an existing serial code, especially the one which uses the finite element discretization, into a non-overlapping domain decomposition code based on the substructuring ideas. Various numerical results of the Schur domain decomposition method as applied to the finite element solution of the shallow water equations are reported and discussed.

As will be explained in Chapter 4, the Schur domain decomposition method may not be cost effective in the absence of fast subdomain solvers and the unavailability of fast subdomain solvers is usual, rather than an exception, for most application problems. In view of this potential disadvantage of the Schur domain decomposition, we propose, in Chapter 5, a novel approach to handle the coupling between the subdomains and the interfaces. We name this new algorithm as the modified interface matrix domain decomposition (MIMDD) method. Different from the Schur domain decomposition method, in which the numerical solutions on the interfaces are determined first, the MIMDD algorithm starts with an initial guess on the interfaces and then iterates back and forth between the subdomains and the interfaces. It turns out that this approach allows successively improved intial solutions to be made both in the subdomains and on the interfaces. The reduced cost in obtaining subdomain solutions due to the improved initial guesses mitigates the aforementioned disadvantage. Both theoretical and algorithmic aspects of the MIMDD method as well as numerical results will be presented and discussed in detail.

Chapter 6 is concerned with the development and application of parallel block preconditioning techniques. Many hybrid methods of non-overlapping domain decomposition result from various combinations of linear iterative solvers and domain decomposed preconditioners (generally consisting of inexact subdomain solvers and interface preconditioners). Two types of existing domain decomposed preconditioners are employed and a novel one is proposed to accelerate the convergence of three currently frequently used and competitive iterative algorithms for the solution of non-symmetric linear systems of algebraic equations, namely, the generalized minimal residual (GMRES) method, conjugate gradient squared (CGS) method and a recently proposed Bi-CGSTAB method, which is a variant of the bi-conjugate gradient method. While all three types of these preconditioners are found to perform

well with GMRES, CGS and Bi-CGSTAB, the newly proposed third type of domain decomposed preconditioners turns out to be computationally the least expensive and the most efficient for solving the problem addressed here, although the second type of domain decomposed preconditioners is quite competitive. Performance sensitivities of these preconditioners to inexact subdomain solvers is also investigated.

Parallel implementation issues are discussed in Chapter 7. We argue in this chapter that, while the domain decomposition method offers an opportunity to carry out subdomain by subdomain calculations, which can be parallelized quite efficiently at the subroutine level, the parallelization of the element by element calculations corresponding to the finite element discretization is also important for achieving a high efficiency of parallelism. A multicolor numbering scheme is described and applied to the parallel assembly of elements, aimed at removing critical regions and minimizing the number of synchronization points in the finite element assembly process. Three parallel processing software packages currently available on the CRAY Y-MP, namely, macrotasking, microtasking and autotasking, are compared. Autotasking utilities are exploited to implement a parallel block preconditioning algorithm. Speed-up results for several mesh resolutions are reported.

Major results and conclusions based on the research work of this dissertation are summarized and a discussion concerning possible future research directions is provided in Chapter 8.

Finally, Appendix A is provided for the purpose of completeness, but it can also serve as a document for those who may not be familiar with the finite element solution of the shallow water equations.

CHAPTER 2

PARALLELISM: TOOLS AND METHODS

> Should we build it if we could? Its potential for solving the problems of a complex world may well justify the expense.
>
> — Willis H. Ware[1]

2.1 Why Parallelism

Computation speed has increased by orders of magnitude over the past four decades of computing. This speed increase was mainly achieved by increasing the switching speed of the logic components in computer architectures. In other words, the time required for a circuit to react to an electronic signal was constantly reduced.

Logic signals travel at the speed of light, or approximately one foot per nanosecond (10^{-9} second) in a vacuum[2]. This signal propagation time could largely be ignored in the past when logic delays were measured in the tens or hundreds of nanoseconds. However, the delay caused by this "extremely fast" signal propagation has become today's fundamental hurdle which inhibits the further increase of the computing speed (see, among others, [66, 188, 211]).

[1] From "The ultimate computer", IEEE Spectrum, 9(3):84-91, 1972.

[2] In practice, however, the speed of electronic pulses through the wiring of a computer ranges from 0.3 to 0.9 foot per nanosecond.

Faced by the limitation of the speed of light, computer designers have explored other architectural designs to achieve further increase in computation speed. One of the simplest of these ideas, yet hard to implement effectively and efficiently, is parallelism, i.e., the ability to compute on more than one part of the same problem by different physical processors at the same time.

2.2 A Brief History

The idea of using parallelism is actually not so new and may be traced back to Babbage's analytical engine in 1840s. A summary of Babbage's early thinking on parallelism may be found in [156]:

> D'ailleurs, lorsque l'on devra faire une longue série de calculs identiques, comme ceux qu'exige la formation de tables numériques, on pourra mettre en jeu la machine de manière à donner plusieurs résultats à la fois, ce qui abrégera de beaucoup l'ensemble des opérations.

Although many of the fundamental ideas were formed more than a hundred years ago, their actual implementation hasn't been made possible until recently.

Limited technology and lack of experience led early computer designers to the simplest computer design model of von Neumann — a single instruction stream/single data stream (SISD) machine in which an instruction is decoded and the calculation is carried out to completion before the next instruction and its operands are handled. As an improvement of this model, parallelism was first brought into a single processor. The parallelism within a single processor was made possible by, for example,

- using multiple functional units, i.e., dividing up the functions of arithmetic and logical unit (ALU) into several independent, but interconnected func-

tional units, say, a logic unit, a floating-point addition unit, a floating-point multiplication unit, etc., which may work concurrently.

- using pipelining, i.e., segmenting a functional unit into different pieces and splitting up a calculation (addition, multiplication, etc.) into correspondingly several stages[3], the sub-calculation within each stage being executed on a piece of the functional unit in parallel with other stages in the pipeline.

The pipelining technique is often heuristically compared to an assembly line in an industrial plant. Successive calculations are carried out in an overlapped fashion and, once the pipeline is filled, a result comes out every clock cycle. Of course, the start-up time, i.e., the time required for the pipeline to become full, incurs an unavoidable overhead penalty. The evolution of the vector processor was considered to be one of the earliest attempts to remove the von Neumann bottleneck [82].

Coupling the pipelining technique with the vector instruction, which results in the processing of all elements of a vector rather than one data pair at a time, leads to the well-known vectorization — parallelism in a single processor. The vector instruction made it possible for the same operation to be performed on many data items and thus multiple fetches of the same instruction are eliminated.

As a first commercially successful vector computer which has had an important impact on scientific and engineering computing, the CRAY-1 was put into service[4]

[3]For instance, a floating point addition may be split up into the following four stages, namely, choosing the larger exponent; normalizing the smaller number to the same exponent; adding the mantissas; renormalizing the mantissa and exponent of the result. A pipelined floating-point adder with four processing stages, for which a simplified description was given above, was illustrated in [124, p. 149]. A simplistic five-stage pipeline for the floating-point multiplication may be found in [66, p. 5].

[4]Four years after Seymour Cray started his company, Cray Research, Inc., in 1972.

at Los Alamos National Laboratory in 1976. Since then, tremendous achievements have been made in the area of vector processing for scientific computing. Today various techniques are quite well established to take advantage of this architectural feature efficiently. For a detailed discussion and account of the history of pipelining and vector processing, see [82, 120, 124, 135].

In comparison with the already available hardware and software technology for vectorization, parallel computers and parallel computing techniques are much less mature. A lack of proper definitions, confusion of terms and concepts and the plethora of different parallel computing systems remain. The difficulty of programming for parallel operations has even led some researchers to the conclusion that sequential operations were to be preferred to parallelism (see [188] and references therein).

The attempt to actually build various parallel computing machines can be, however, traced back to the 1950s. A sizable amount of research on parallel scientific and engineering computing was carried out in the 1960s due to the impending advent of parallel computers. An excellent survey which covers most research activities in parallel scientific computing before and up to the 1970s was provided in [162]. In particular, the author reviewed studies of parallelism in such numerical analysis topics as optimization, root finding, differential equations and solutions of linear systems. A complete annotated bibliography up to the time of its publication on vector and parallel numerical methods and applications in meteorology, physics and engineering, etc. can be found in [191]. A review of early results on vector and parallel solutions of linear systems of equations and eigenvalue problems, along with background information concerning the computer models and fundamental techniques was provided in [117]. The early recognition of fundamental differences between parallel and sequential computing was reviewed in [219]. Factors that limit com-

puter capacity, the need to build powerful computing systems and the possible cost ranges were discussed in detail in [234].

During the past twenty years, the literature on parallel computing has been increasing at a very rapid rate. One of the most frequently referenced works is [120], which contains detailed information on the history, parallel architecture hardware as well as parallel languages and algorithms. More about the history and evolution of architectures may be found in [148, 240]. Detailed discussions on both hardware and software for quite a number of currently commercially available parallel computers have been provided in [15]. Several supercomputer architectures and some technologies were reviewed in [142, 143]. [188] contains a thorough review of vector and parallel scientific computing and a rather complete bibliography up to 1985. A more recent contribution, [189] collects over two thousand references on vector and parallel numerical algorithms research up to 1990.

2.3 Taxonomy of Parallel Architectures

2.3.1 Flynn's Taxonomy

The most frequently referenced taxonomy of parallel architectures was provided by Flynn in [89]. He characterized computers to fall into the following four classes, according to whether they possess one or multiple instruction streams and one or multiple data streams[5]:

1. SISD — single instruction stream/single data stream. This is the conventionally serial scalar von Neumann computer, mentioned in Section 2.2. This type of computer performs each instruction of a program to completion before starting the next instruction.

[5] A stream is defined as a sequence of items (instructions or data) as executed or operated on by a processor.

2. SIMD[6] — single instruction stream/multiple data stream. This type of computer allows new instructions to be issued before previous instructions have completed execution or the same instruction to be operated on different data items at the same time. Thus, the simultaneous processing of different data sets within a single processor or a collection of many identical processors (referred to as processing elements) becomes possible. This classification includes all types of vector computers.

3. MISD — multiple instruction stream/single data stream. Although Flynn [90] indicated special cases to support this classification of the architecture, there are, currently, no computers that issue multiple instructions to be operated on a single data stream.

4. MIMD[7] — multiple instruction stream/multiple date stream. Computers in this class usually have arrays of linked physical processors, each processor running under the control of its own instruction stream. Thus, a great flexibility is permitted in the tasks that processors are carrying out at any given time. This classification includes all forms of multi-processor systems.

Typical examples of computing systems which belong to each of the SISD, SIMD and MIMD types were given in [183] (see also [192]). Block diagrams for SISD, SIMD, MISD and MIMD machines may be found in [124, p. 33].

The taxonomy discussed above provides a simplistic and broad characterization of quite different computer architectures. There exist other more complicated classifications. Notable is the one introduced by Hockney [120], which was based on

[6]This type of machines is often referred to as array processors.

[7]This type of machines is commonly known as multi-processor systems.

a structural notation for machine systems. However, this classification scheme was considered too fine to be useful [240].

Flynn's taxonomy of computer architectures, although coarse, is certainly helpful to computational scientists. It should be borne in mind, however, that the current (super) computers are much more complicated and endowed with a hybrid design, i.e., an architecture which falls under more than one category. For example, the CRAY Y-MP is a MIMD machine in general, with each individual processor being of SIMD type[8]. In addition, the complication is even furthered by the memory organization — local, shared or local & shared, and by numerous inter-connection schemes between memories and processors. These intricate factors have led some researchers to suspect that there will never be an absolutely satisfactory taxonomy for parallel computing systems (see, for example, [240]).

A more complete description and discussion of both hardware and software on MISD and MIMD machines (including multiple SIMD (MSIMD) and partitionable SIMD/MIMD machine architectures) and other relevant theoretical issues are given in [19, 124, 210].

2.3.2 More on MIMD Architectures

Most of the current research interest lies in the architectures, programming languages, data structures and algorithms for the MIMD type of machines. MIMD architectures may be further divided into two categories, namely, multi-computer networks and multi-processors. The former category refers to physically dispersed

[8]It is believed that a hybrid MIMD/SIMD machine is ideal for adaptive mesh refinement (AMR) numerical algorithms [198].

and loosely coupled computer networks[9]. The latter category may be further divided into another two classes: shared (or common) memory or tightly coupled and distributed (or local) memory or loosely coupled parallel computers.

For a shared memory parallel machine, for which a schematic model is presented in Figure 2.1, all processors share a common pool of the main memory and every processor can access any byte of memory in the same amount of time.

Date items associated with private variables are located in physically disjoint memory spaces and only locally visible to the processors. Different processors have their own private variables. Communications between different processors are accomplished through explicit declarations for the memory space to be shared between processors. In other words, data items associated with shared variables are made globally visible or accessible to all the processors involved (see [26] for a good presentation of relevant concepts).

A major advantage enjoyed by this type of architectures is the fast communication between processors when the number of processors is relatively small.

An obvious disadvantage is, however, that several processors may try to access the same memory location (or the shared variable from a programmer's point of view) at the same time. Because of the random scheduling of the processes, a synchronization mechanism must be used to ensure that different processors are working in the correct order and with the correct data. This accounts for the so-called contention delay, which obviously aggravates as the number of processors increases[10].

[9]The computing on multi-computer networks is usually referred to as distributed computing. This type of computing was considered not to speed up the execution of individual jobs, but to increase the global throughput of the whole system [183].

[10]It is generally considered to be a practical limit for 16 processors to share a common memory, [82].

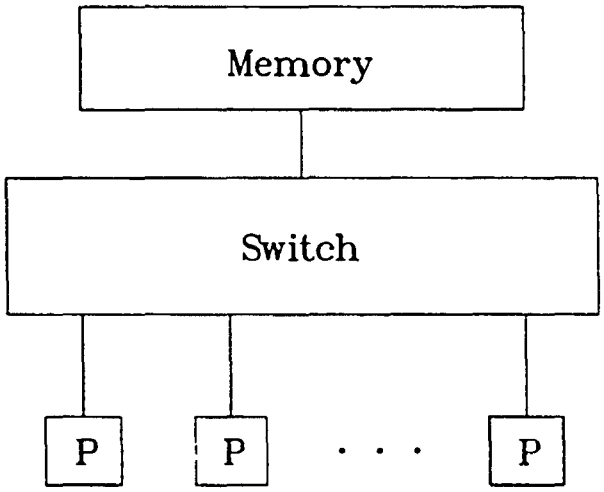


Figure 2.1: The schematic model of a shared memory multi-processor computer.

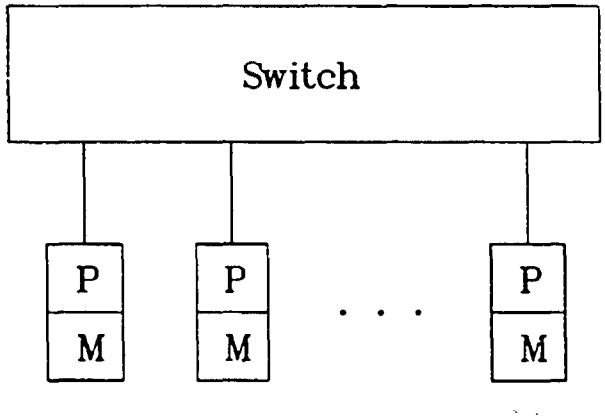


Figure 2.2: The schematic model of a distributed memory multi-processor computer.

For a distributed memory parallel machine, for which a schematic model is presented in Figure 2.2, there is no global memory. Each processor has its own local memory and thus there are no direct interactions with the memory on any other processor.

This type of computer architecture is heuristically termed as a "message passing" multi-processor computer because of the fact that the communications between various processors are made possible by sending and receiving messages through some inter-connection networks. For distributed memory machines, different from shared memory computers, there is no explicit hardware synchronization. Synchronization must be explicitly coded by the programmer. In general, this requires major recoding efforts for porting programs written for serial machines onto distributed parallel computers.

There are numerous inter-connection schemes [66, 124, 188] in which processors are connected. Whatever schemes are used, they all suffer from the same shortcoming, namely, data may need to be passed through several intermediate processors prior to their reaching their final destinations. An important parameter which may be used to measure the seriousness of this disadvantage is the communication diameter or length, which refers to the maximum number of transmissions that must be made in order to communicate between any two processors [185].

Due to the coexistence of these two quite different MIMD types of parallel architectures, there is constantly an ongoing debate as to which one is to be preferred with regard to implementing numerical algorithms. An excellent discussion on this issue is provided in [204].

To conclude this section, we point out that another simple and clear classification strategy was proposed in [183] which classifies multi-processor systems into

the following three categories, namely, fine-grain, medium-grain and coarse-grain machines.

2.4 Some Issues Related to the Design of Parallel Numerical Algorithms

The availability of multi-processor systems has introduced new issues and challenges [182, 204] for numerical analysts and computational scientists. To take advantage of such advanced architectures, one has to partition his problem into separate computational tasks, schedule each task for execution on a processor and perform communication and synchronization among the tasks. This generally goes well beyond some trivial reorganization of an existing sequential code, but requires significant redesigning and restructuring of the basic algorithm. As a consequence, some good sequential algorithms were found unsuitable and, on the other hand, some old and inefficient sequential algorithms have been revisited and resuscitated due to their potential for parallelism (see [219] and references therein).

2.4.1 Complexity and Degree of Parallelism

Traditionally, efforts were made to design such algorithms as to minimize the number of arithmetic operations (computational complexity) involved. Although computational complexity is still a fundamental consideration for parallel scientific computing, there are other more important factors to take into account: the degree of parallelism[11], i.e., the amount of work (measured in percentage) which can be executed in parallel, and the communication & synchronization penalty [95].

[11] which is a key parameter in the Amdahl's law (see Section 2.6). Note that the definition here for the degree of parallelism is different from that given in [185].

The availability of various parallel architectures requires computational scientists (1) to adapt already existent sequential algorithms to the new architecture and (2) to directly design and develop novel parallel algorithms for the efficient task-to-processor mapping. In order to impose a parallel structure on a given problem and/or increase the degree of parallelism, one usually applies (often simultaneously) the following two related ideas [2, 183, 185, 187, 207], namely, (1) renumbering or reordering and (2) divide and conquer or decomposition strategies.

Renumbering or reordering is used for restructuring the computational domain and/or the sequence of operations in order to increase the percentage of the computation that can be carried out in parallel and, sometimes, remove critical regions and minimize the number of synchronization points [35, 36, 85]. The divide and conquer approach itself is concerned generally with breaking up the original problem into several smaller subproblems and computations into a number of stages, then assigning the different subproblems to different physical processors for independent treatment within each stage followed by inter-processor communication and synchronization at the end of the stage. Quite often, renumbering or reordering is applied to decouple the original problem and realize the divide and conquer approach. Domain decomposition based on substructuring ideas, to be detailed later, is one good example.

It should be pointed out, however, that the degree of parallelism is often improved at the cost of introducing extra computational work. In a series of papers which address the problem of solving bi- or tri-diagonal linear systems [48, 76, 116, 140, 218, 233] on vector and parallel computers, the authors developed the following three techniques, namely, recursive doubling, cyclic reduction and divide and conquer. Unfortunately, all these three techniques resulted in a substantial increase in the number of floating-point operations which made these methods less attractive. This

emphasizes the important role of computational complexity analysis even in the study of parallel computing. An efficient parallel algorithm should possess a good degree of parallelism and, at the same time, result in only a small amount of extra computation.

In general, an algorithm with a high degree of parallelism does not necessarily result in an efficient parallel computing method (remember that point Jacobi method as an iterative scheme for solving algebraic linear systems possesses a perfect degree of parallelism, but it is seldom used due to its slow convergence rate). There is a balance to be found between parallelism and the amount of computation necessary to find the solution to a given problem [240]. The ultimate goal of parallel processing is to reduce the wall clock time by a factor close to the number of processors allocated to a job without having to pay significantly more for the increase in CPU time. In other words, parallel processing shortens the production time of computing results but, at the same time, usually introduces an extra cost and is computationally more expensive than its sequential counterpart. It is this extra cost that we try to minimize.

2.4.2 Communication and Synchronization

Another critical issue that has an important impact on the performance of a parallel algorithm involves communication and synchronization, which constitute, if several physical processors are employed to cooperatively solve the same problem simultaneously, an unavoidable overhead which we strive to minimize. During communication and synchronization, processors are not performing any useful computation and some of them, in order to coordinate their steps, may be forced to stay idle. Memory contention delays for a shared memory system may cause serious problems, depending on the amount of computational work within the critical

region of the code and the number of processors involved. Hence, communication and synchronization should be used sparingly in order to achieve high efficiency of parallel processing on a single job.

Sometimes, a clever renumbering (e.g. a multicolor renumbering), which is essentially equivalent to a proper reordering of computational sequences, may remove the critical regions and, at the same time, minimize the number of synchronization points [35, 36, 85]. In general, instead of devising algorithms of small granularity that require relatively frequent communication and synchronization between processors, a non-interactive way of apportioning the work among different processors should be found so that tasks of relatively large granularity are created. However, this may lead to additional difficulty in load balancing[12] — another important issue in parallel computing. An unbalanced load distribution will, in turn, increase synchronization costs. Indeed, parallel computing has brought about much more complicated issues than sequential computing.

2.4.3 Synchronized vs. Asynchronous Parallel Algorithms

In all of the above discussions, we have tacitly assumed that the algorithm under consideration is the so-called synchronized parallel algorithm. This type of algorithms consists of more than one process with the property that there exists a process such that some stage of the process can not be activated until after another process has completed a certain stage of its program. The elapsed time for completing a certain stage of the computing is determined by the slowest process. This constitutes the basic weakness of a synchronized algorithm, which may result in worse than expected speedup results and inefficient processor utilization.

[12] Load balancing is easier for tasks of smaller granularity, e.g., splitting a loop and distributing them across processors [66].

As a remedy, some researchers have been concerned with designing and developing asynchronous parallel algorithms [124], in which, although contention delay is still a potential problem, processes generally do not have to wait for each other and communication is achieved by reading dynamically updated shared variables stored in the shared memory. It should be pointed out, however, that in developing asynchronous parallel iterative numerical algorithms, the iterates generated by the asynchronous iterative algorithm may be different from those produced by the sequential algorithm or synchronized parallel iterative algorithms due to the random scheduling of processes, hence the convergence or convergence rate is hard to predict and a general theory is not yet available.

2.4.4 Memory Access and Data Organization

As a final, but not the least important issue, we address concisely those concepts and techniques which are most relevant to vector and parallel computing, namely, memory access and data organization.

In fact, with the increasing power of functional units, it is important to match, at a reasonable price, the information transfer rate (the so-called memory bandwidth[13]) with the processor speed. In other words, to sustain the fastest possible processing speed offered by the computational units, one has to ensure that the memory is able to deliver instructions & operands fast enough to the computational units and that the computational units can get rid of their output sufficiently fast. The movement of data from and to the main memory can be as costly as arithmetic operations on data. An example was given in [237], which showed, for matrix-matrix multiplication

[13]The bandwidth of a system is generally defined as the number of operations performed per unit time. The memory bandwidth is measured by the number of memory words that can be accessed per unit time.

problems, that the execution of a vector code can be slower than that of a scalar version due to improper memory management.

The data flow between the memory and the computational units is the most important and critical part of a computer design. It is too expensive to build a large memory with such a high speed as to match the fast processing speed. To get around the dilemma of needing rapid access to the memory, on the one hand, and also having a large amount of memory space, on the other hand, computer designers built a hierarchical structure into the memory. A typical memory hierarchy (consisting of a fast but small cache[14] and the slow but large main memory, etc.) was illustrated in [183]. From bottom to top, each level in the hierarchy represents an order-of-magnitude increase in memory access speed, and several orders-of-magnitude decrease in capacity, for the same cost.

In general, efforts should be made to obtain a high ratio of time spent on computations to time spent on memory references in order to efficiently and effectively utilize high speed processors. Typically, processor speed is much greater than the access (either fetch or store) speed to the main memory. The speed gap between the processor and main memory is closed by using a fast, but small, cache memory between them. To prevent the memory from becoming a possible bottleneck, one must exploit and make use of the locality of references in the code development. Specifically, memory references to data should be contained within a small range of addresses and the code exhibits reuse of data [66]. Thus, most memory references will be to data in the cache and the overall memory access rate will be effectively approaching that of the fast cache memory, which is typically $5 \sim 10$ times faster

[14]Cache memories are high speed buffers inserted between the processors and main memory, which are typically five to ten times faster than the main memory.

than the main memory, resulting in a bandwidth balance between processors and the memory.

As a very good example, we mention here that one of the reasons that Basic Linear Algebra Subprograms (BLAS) [63, 141, 65, 64] were gradually brought to higher levels is to increase the ratio of floating-point operations to data movement and make an efficient reuse of data residing in cache or local memory. Typically, for level 3 BLAS, it is possible to obtain $O(n^3)$ floating-point operations while requiring only $O(n^2)$ data movement, where n specifies the size of matrices involved.

2.5 Programming Aspects

Many different types of parallel computers have been provided by various companies. Each one of these types has its own unique architecture and characteristics equipped with extensions, for parallel programming, to an existing programming language such as FORTRAN, or a new language specially designed for a particular machine (see, for instance, [15, 26, 83] and [67] along with references therein). Imposing a standard model of parallel computing language is still too early and is impeded by the current level of understanding about parallelism. In fact, there is no agreed-upon point of view about what a parallel programming language should be or, at least, in which direction the extension should be made to currently available serial high-level languages. Researchers interested in implementing their parallel algorithms on different types of parallel computers are, therefore, faced with formidable tasks.

The purpose of this section is, of course, not to review different parallel programming tools supported by their respective hardware and operating systems. Appropriate computer manuals should be consulted for this purpose. However,

knowing where to impose a parallel structure in the code is essentially a data problem, not a coding problem. Indeed, the major consideration that limits parallelism within a program is the scope of data items. Whichever machines and programming languages are chosen, a parallel code consists of some logical and machine-understandable expressions of control flows which enable the computer to process those sections containing data capable of being operated on simultaneously without adversely affecting other data. In view of this, we discuss, in this section, topics related to data dependencies, control and execution flow.

2.5.1 Control Flow Graph

A control flow or data dependency graph is an invaluable aid to parallel programming which was presented and recommended in [67, 68]. Although it was originally employed to illustrate and facilitate the use of a software package called SCHEDULE for portable FORTRAN parallel programming, the technique actually has a much wider applicability in the development of codes for both fine-grained and coarse-grained parallelism.

A typical control flow graph consists of two basic elements, namely, nodes and directed edges, which stand for processes or subroutines and execution dependencies, respectively. A process (represented by a node) can not be initiated unless all the other processes with edges directed to it have completed their execution (i.e., the incoming edges have been removed from that node). Processes without incoming edges (as have been removed) have no data dependencies[15] and hence may be executed in parallel.

[15]For shared memory systems, this means that there is no contention for "write" access, but "read" access to a shared variable is allowed.

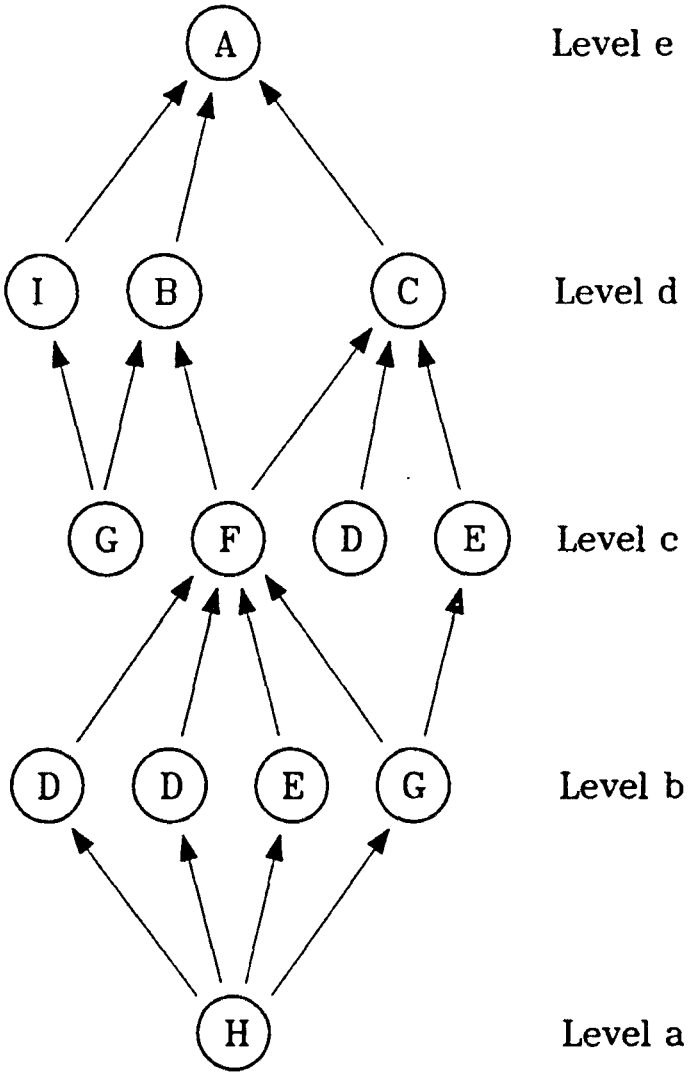


Figure 2.3: A sample control flow or data dependency graph.

We present a sample data dependency graph in Figure 2.3, where level a $\sim$ level e have no real meaning but are labels used for an easy and clear description of the graph. The computations begin with H, G [level c] and D [level c] working in parallel. As soon as H is completed, D [level b], E [level b] and G [level b] can proceed simultaneously and, possibly, in parallel with the already existing processes G and D mentioned above, if they have not finished yet. Two copies of D at level b may be understood as two identical subroutines operating on two different, independent data sets (the same convention applies to other identical copies of the nodes). The execution of I can commence and continue simultaneously with other existing processes provided that G [level c] has completed its calculation. However, processes B and C can not be initiated even if G [level c] and D [level c] have finished their jobs. E [level c] may start immediately after the completion of G [level b]. However, F can not be started unless all four processes at level b have completed their calculations. B (C) can be executed as long as G [level c] and F (D [level c], E [level c] and F) have been completed. Finally, as soon as B, C and I finish, one can execute A, which terminates the whole computation process upon its completion.

It is obvious that the control flow graph to a given problem is not unique. Even using the same parallel algorithm, the graph can be made at different levels of detail and granularity. In principle, given a target machine, the control flow graph may readily be translated into a parallel program. A parallel algorithm and the corresponding control flow graph for the solution of a triangular linear system of equations $Tx = b$ was provided in [66]. A control flow graph for evaluating x^{2^k}, where k is a positive integer, was given in [237]. Some more examples [67, 68] are available in the context of developing a user interface to the SCHEDULE software

package. Simplified control flow graphs for the iterative Schur and modified interface matrix domain decomposition algorithms are provided later (see pages 81 and 122).

A control flow graph respects data dependency relations and identifies the next schedulable process(es). Following the logical flow of such a graph, the computation can be expected to proceed in the correct order leading to correct computing results. However, the graph does not take performance efficiency into consideration. For load balancing and other implementation details, an execution graph may be constructed to properly partition the computation and/or data as well as to force certain processes to complete before others. Such a graph must not, of course, violate the execution dependencies specified by the control flow graph.

2.6 Performance Analysis

2.6.1 Performance Analysis for Vectorization

The computational speed of today's high performance architectures is usually measured in terms of Mflops (millions of floating-point operations in one second) or even Gflops (1 Gflop = 1000 Mflops). Extensive research is yet to be carried out for achieving a sustainable teraflop (1000 Gflops) peak performance in the future.

By definition, the rate of computing may be expressed the following way

$$r = \frac{N}{t} \text{ Mflops} \tag{2.1}$$

where N is the number of floating-point operations (flops) carried out in t microseconds (10^{-6} seconds).

Vectorization is capable of sustaining a high speed of computing by performing exactly the same operation (no multiple fetches of the same instruction from the memory) on many data items and by using pipelining techniques. However, due to

the delay before obtaining the desirable results for timely loading of operands into the pipeline, there exists a problem of data dependency and recurrence which will prevent a given loop from vectorization, resulting in a low speed of calculation. In fact, the computational speed is often far from uniform for any realistic numerical simulation of a process. Therefore, the classical way of evaluating the efficiency of an algorithm by simply counting the number of flops is obviously no longer valid for vector processors. New ways of analyzing and modeling the computational performance had to be developed.

A formula capable of predicting the improved performance as a result of vectorization was introduced in [6] while vector machines were still in the design stage. The formula is the well-known Amdahl's law for vector processing. If we assume that there are only two modes of operation, namely, one with a high speed and the other with a low speed, then the formula predicting the overall performance may be expressed by

$$r = (\frac{\alpha}{v} + \frac{1-\alpha}{s})^{-1} \text{ Mflops} \tag{2.2}$$

where

- r — overall or average speed of computation;
- v — high computing speed or vector processing speed;
- s — low computing speed or scalar processing speed;
- α — degree of vectorization, i.e., fraction of the total amount of work carried out with a computing speed of v Mflops.

In order to quantitatively appreciate (2.2), we plot, in Figure 2.4, the speedup r/s due to vectorization as a function of α and the ratio of vector processing speed to scalar processing speed v/s. It is clear that the overall performance is unfortunately

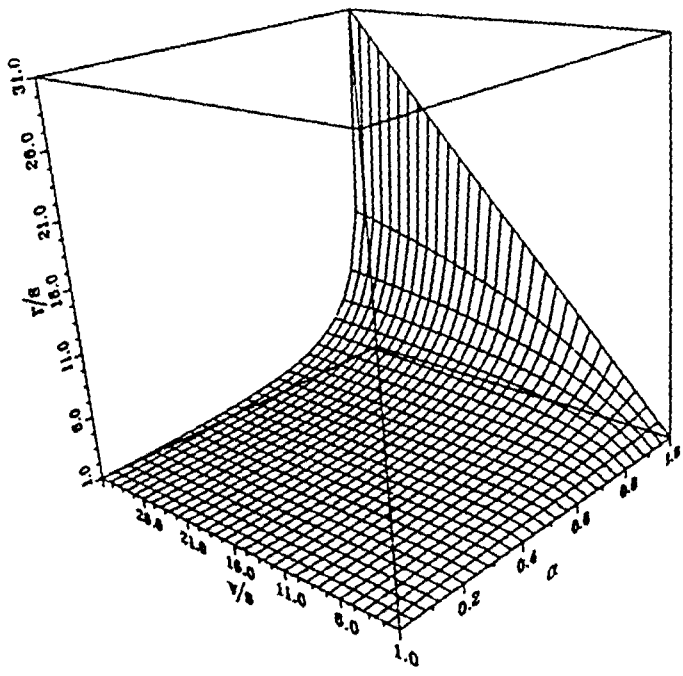


Figure 2.4: Overall performance associated with two modes (high speed and low speed) of operation. $r/s = (\frac{\alpha}{v/s} + 1 - \alpha)^{-1}$.

dominated by the low-speed mode. In other words, the overall computing speed can be drastically reduced for even a small portion of calculations ($1 - \alpha$ being small) which are carried out with the rate s. Increasing the vector processing speed v will only marginally improve the overall performance if the bottleneck due to the low-speed mode of operation is not removed or made less serious. All this suggests that it may not be cost effective for computer manufacturers to invest a lot of human effort and material resources to improve vectorization without investing into the enhancement of the scalar processing speed (see also [61]).

The formula corresponding to n modes of operation can easily be generalized as follows

$$r = \frac{N}{(N_1/r_1 + N_2/r_2 + \cdots + N_n/r_n)} \text{ Mflops} \tag{2.3}$$

where, for $i = 1, 2, \ldots, n$,

- r_i — computing speed corresponding to i-th mode of operation;
- N_i — number of flops carried out with a computing speed of r_i Mflops

and $N = N_1 + N_2 + \cdots + N_n$ is the total number of flops required of a given task.

2.6.2 Performance Analysis for Parallelization

For parallel processing, the goal is to reduce the wall clock time (elapsed time for execution), while the total CPU time involved in parallel computing is usually larger than the CPU time consumed by the execution of a sequential code for the same problem. Ideally, the wall clock time would be reduced to $1/n$ of the wall clock time required for sequential calculation if the work can be divided into n equal-size parts which are executed by n equally powerful processors. However, this is not possible due to the existence of non-parallel segments contained in a parallel code and various parallel processing overheads involved.

The performance of a parallel code is usually measured by the speedup, namely,

$$S_n = \frac{W_s}{W_n} \tag{2.4}$$

where

- S_n — speedup for a system of n processors;
- W_s — serial processing wall clock time;
- W_n — parallel processing wall clock time by n processors.

The corresponding efficiency for using a computing system with n processors is defined by

$$E_n = \frac{S_n}{n} \tag{2.5}$$

It should be pointed out that W_s in (2.4) was proposed to mean the wall clock time for using the best sequential algorithm on a particular architecture [188]. However, determining the fastest sequential algorithm for a specific application problem on any particular computing system may be more difficult than developing a parallel algorithm itself. Consequently, the speedup as defined by (2.4) often measures how a given algorithm compares with itself on one and n processors. This measurement properly incorporates any communication and synchronization overhead and, in a sense, expresses how busy the processors are kept computing (rather than communicating and waiting). However, a potential pitfall is that an algorithm with a perfect speedup may not run much faster or may even run slower than a serial algorithm designed for solving the same problem.

Amdahl's law (2.2) is easily extended to the parallel case. The assumption is that the whole computational work can be divided into only two parts, i.e., a strictly sequential part and a part which can be carried out simultaneously on n processors.

Then, in the absence of communication and synchronization overhead, the speedup is

$$S_n = \frac{n}{\alpha + (1-\alpha)n} \tag{2.6}$$

where

- S_n — speedup obtained on a system with n processors;
- α — degree of parallelism, i.e., percentage of the total work (measured in CPU time) which can be carried out in parallel using n processors.

The similarity between Amdahl's laws for vector and parallel processing is notable. Specifically, if we make the following substitutions

$$S_n \rightarrow \frac{r}{s}, \quad n \rightarrow \frac{v}{s} \tag{2.7}$$

and understand α in the context of parallel processing, then the speedup S_n obtainable on a system with n processors is graphically shown in Figure 2.4 as a function of α and n. Similarly, we conclude that the overall performance of using a multiprocessor system is dominated by the sequential part. A small amount of serial work ($1-\alpha$ being small) will result in a large reduction in the speedup, especially for large n. Moreover, no matter how close α is to 1, the speedup will soon fall behind n. In Figure 2.5, we show that the efficiency (obtained by (2.5)) is a decreasing function of the number of processors involved, while keeping α fixed.

For a vector-parallel computing system, i.e., a multi-processor system in which each of its processors has vector capabilities, the most efficient utilization of the resources requires that the product $r/s \cdot S_n$ of speedup r/s due to vectorization and the speedup S_n due to parallel processing be maximized. It was illustratively explained in [83] that both the degrees of vectorization and parallelism must be high

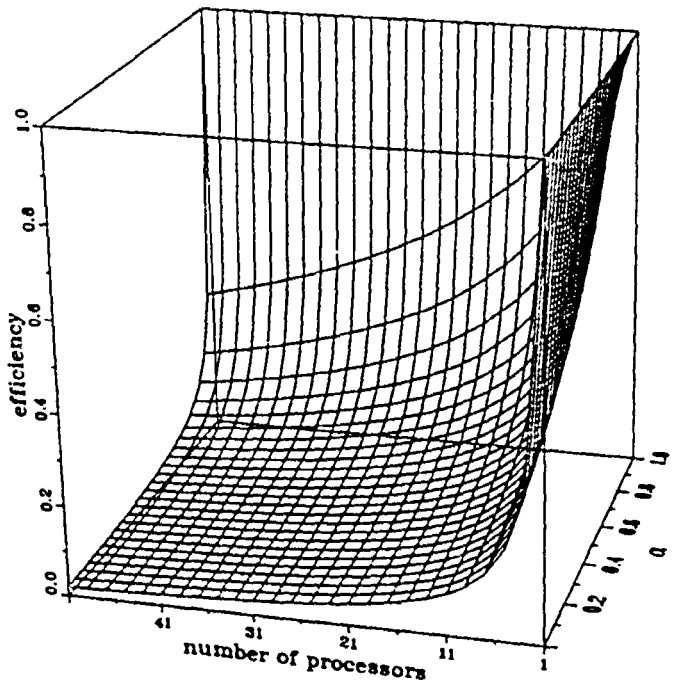


Figure 2.5: Efficiency for using a multi-processor computing system.

enough in order to ensure substantial improvement over serial processing and the overall performance will be deteriorate as the number of processors increases.

Amdahl's law for parallel processing casts a pessimistic shadow on the possible benefits one can obtain from massively parallel computing. The implicit assumption behind Amdahl's law is that the size (measured in W_s, see (2.4)) of the problem under consideration is fixed. The formula given in (2.6) was obtained by assuming that part of this fixed-size problem may be carried out in parallel and the rest be done sequentially. By this formula, the possible maximum achievable speedup is only 100 even if 99% of work is carried out in parallel on a multi-processor system with an infinite number of processors. Is massively processing really meaningful and beneficial or is Amdahl's law inappropriate in this context?

An alternative formulation was put forth in [106] in an attempt to explain some unprecedentedly excellent speedup results [107] on a 1024-processor hypercube in Sandia National Laboratories. The fundamental observation is that, in practice, the problem size is not fixed, but scales with the number of processors involved in the actual computation. The key assumption in this new formulation is that W_n (measured in a dedicated mode for multi-programming operating systems) being held fixed. If a fraction α (measured as a percentage) of W_n is spent on parallel computing and the rest time on serial processing, then the same work would take $W_s = (1-\alpha)W_n + \alpha n W_n$ to run on a single processor. Therefore, the speedup is

$$S_n = \frac{W_s}{W_n} = 1 - \alpha + \alpha n \tag{2.8}$$

In contrast to Amdahl's law, (2.8) indicates a predicted speedup of $S_n = 0.01 + 0.99n$ on a multi-processor system with n processors if 99% of the work (measured by W_n) may be carried out in parallel.

For comparison with the efficiency predicted by Amdahl's law (see Figure 2.5, we produce a similar 3-D figure indicating how the modified efficiency, as calculated by using (2.8), depends on the number of processors and α (see Figure 2.6).

The above discussion did not, however, take into consideration the communication overhead, which may dominate the overall performance. The overhead issue was incorporated into the formulation in [7] for analyzing the expected performance due to massive parallelism. The result is, unfortunately, discouraging. Due to this result, it was claimed in [61] that, in the future, there may be a convergence of ideas and techniques around architectures comprising only a few hundred processors.

2.7 Conclusions

- Parallelism was introduced to be one of the novel architectural features of today's computers for further increasing processing speed. Although the ideas of parallelism are old and simple, the efficient and cost effective implementation of parallel numerical algorithms is not an easy task. More difficulties originate in a plethora of different parallel computing systems.

- Flynn's taxonomy provides one of the simplest characterizations of essentially different parallel computer architectures. There are other more refined classifications. However, the intricate nature of parallel computing systems may make an absolutely satisfactory taxonomy impossible.

- In the MIMD category, we have

$$\text{MIMD} \begin{cases} \text{multi-computer networks} \\ \text{multi-processors} \begin{cases} \text{shared memory parallel computers} \\ \text{local memory parallel computers} \end{cases} \end{cases}$$

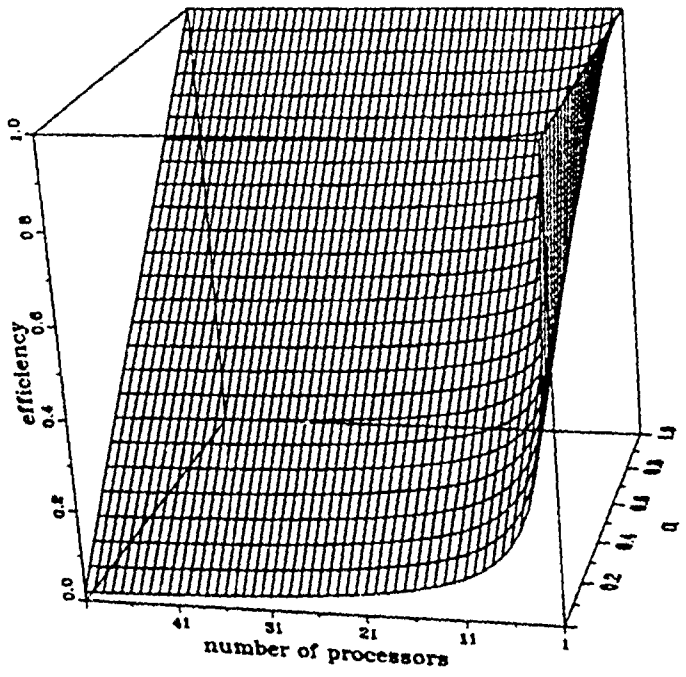


Figure 2.6: Modified efficiency (in comparison with that shown in Figure 2.5) for using a multi-processor computing system.

It is generally considered that the shared memory (tightly coupled) parallel computers represent conventional architectures; while the local memory (loosely coupled) parallel computers stand for novel or modern computer architectures.

- In the context of parallelism, the quality of a numerical algorithm can not be judged by the analysis of computational complexity alone, equally important factors are the degree of parallelism (i.e., the percentage of the total work measured in CPU time which may be done in parallel) in the algorithm, communication & synchronization issues and the locality of reference within a code.

- A control flow or data dependency graph is an invaluable aid to parallel programming.

- The performance analysis for vectorization shows that the overall computing speed is dominated by the scalar processing rate. Rather similarly, the performance analysis for parallelization reveals that the efficiency of parallelism is very sensitive to the existence of even a small amount of serial computational work.

CHAPTER 3

DOMAIN DECOMPOSITION METHODS

3.1 Origins

Over the past thirty years, a tremendous variety of parallel numerical algorithms for scientific and engineering computing have been proposed [189]. Most of the recently proposed parallel computational strategies for solving partial differential equations (PDE's) were based on domain decomposition ideas.

Domain decomposition ideas are actually not new, but are rather old ones which have been forgotten. It is widely acknowledged that Schwarz [209] was the first to employ domain decomposition ideas for establishing the existence of harmonic functions on regions with nonsmooth boundaries, by constructing the region under consideration as a repeated union of other regions. A class of domain decomposition techniques based on the early work of Schwarz is now known as the Schwarz alternating method (see, among others, [16, 98, 144, 145, 146, 160, 161, 215]). The Schwarz alternating procedure is now, however, generally called multiplicative Schwarz method in contrast with the more recently proposed additive Schwarz algorithm, which may be regarded as a method for constructing parallelizable domain decomposition preconditioners (relevant references and some details will be furnished later in Section 3.5.2).

Another class of domain decomposition methods, namely, iterative substructuring methods[1], may be traced back to the work of structural engineers in the sixties

[1]The methods are closely related to some mathematical theories developed by Poincaré and Steklov in the 19th century (see [197] and references therein).

(see [93, 193, 202] and, in addition, we mention, among numerous other papers, [3, 37, 84, 131, 236]). Substructuring techniques were developed in the sixties primarily for the following two reasons:

- the substructuring treatment of aerospace structures provides a way to save a significant amount of computer storage and thus made possible the finite element modeling of very complex structures at that time;
- the substructures themselves may be viewed as complex (super) elements whose stiffness matrices can be stored for later use in an overall different problem but with the same or rather similar structural components to avoid repetitive work.

We refer to [235] for some further, however, general remarks on the aforementioned two classes of domain decomposition methods.

The term "domain decomposition", in a rather general sense, refers to a class of numerical techniques for the replacement of PDE's defined over a given domain with a series of problems defined over a number of subdomains which collectively span the original domain. The solution to the original problem is obtained by solving a subproblem (which is probably much easier to solve than the original) defined on each of these subdomains (different grid resolutions and numerical techniques may be used in different subdomains) and by patching together the subdomain solutions. The computational work on each subdomain is usually associated with a task or software process which will be scheduled onto and handled by a different processor of a parallel computer.

3.2 Saint Venant's Torsion of a Cylindrical Shaft with an Irregular Cross-Sectional Shape

As an example, let us consider the torsion of a cylindrical shaft — an important problem in engineering. The cross section (in the x-y plane which is not shown in the figure) of the bar is of the shape given in Figure 3.1 (a). The problem is to determine the stress distribution and deformation of the shaft under the action of an external torque. The solution to the problem can be obtained by Saint Venant's theory of torsion [92]. The corresponding mathematical problem may be formulated in either of the following two ways:

1.

$$\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} = 0 \quad \text{in } \Omega \tag{3.1}$$

subject to Neumann's boundary condition

$$\frac{\partial \varphi}{\partial n} = y\cos(x,n) - x\cos(y,n) \quad \text{on } \partial\Omega \tag{3.2}$$

where $\varphi(x,y)$ is the warping function and n is a unit outward normal vector to the lateral surface of the shaft:

2.

$$\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = -2G\alpha \quad \text{in } \Omega \tag{3.3}$$

subject to Dirichlet's boundary condition

$$v = 0 \quad \text{on } \partial\Omega \tag{3.4}$$

where $v(x,y)$ is Prandtl's stress function, G is the shear modulus of the shaft material and α is the angle of twist per unit length.

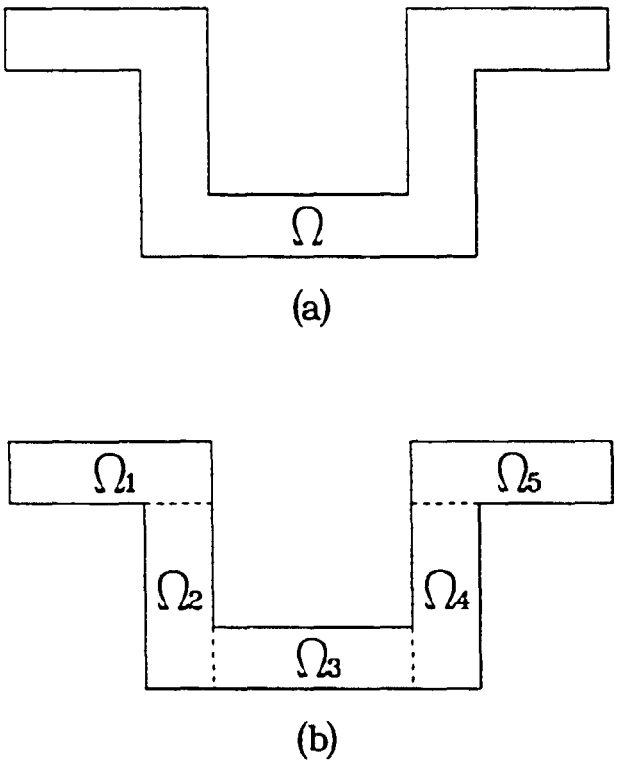


Figure 3.1: Saint Venant's torsion of a cylindrical shaft with the cross section shown in (a). The original cross-sectional domain Ω is artificially divided into five nonoverlapping subdomains $\Omega_1, \Omega_2, \ldots, \Omega_5$, as shown in (b).

Domain decomposition techniques can be applied to solve the above Laplace's or Possion's equation by artificially dividing the original domain Ω into five nonoverlapping subdomains $\Omega_1, \Omega_2, \ldots, \Omega_5$ (see Figure 3.1 (b)). In principle, as long as the numerical values on the interfaces (represented by the dash lines in Figure 3.1 (b)) are known, the subdomain problems are well defined and may be solved independently and, due to the regularity of the subdomain shape, by using fast solvers [221, 222]. The key issue is how to determine the interfacial degrees of freedom efficiently. An iterative procedure with appropriate preconditioners [38, 46, 131] is usually carried out to find interfacial values with desired accuracy. Because fast solvers are locally exploitable, the application of domain decomposition methods to this particular problem can be beneficial even for serial computing, provided that the computational cost involved in enforcing proper conditions on the interfaces is not greater than the computational work already saved. The possibility of mapping each of the subdomain calculations onto a different processor for parallel computing would result in a further reduction in execution time.

3.3 Three Decomposition Strategies for the Parallel Solution of PDE's

In general, given a boundary value problem in Ω of the form

$$Lu = f \tag{3.5}$$

there are three different ways of devising parallel numerical algorithms for the solution of (3.5). These are operator decomposition, function-space decomposition and domain decomposition. The main algorithmic features of these three approaches are explained below:

- Operator decomposition: The main idea is to decompose the original differential operator, namely,

$$L = L_1 + L_2 + \ldots + L_n \tag{3.6}$$

The parallelism is usually exploited by separately inverting L_i, $i = 1, 2, \ldots, n$, or when n is small, by taking advantage of the independent tasks within each phase embedded in an outer iteration which encompasses all terms of L_i, $i = 1, 2, \ldots, n$;

- Function-space decomposition: For this approach, one decomposes the solution $u \in U$ into n component solutions u_i, $i = 1, 2, \ldots, n$, which belong to appropriate subspaces U_i, $i = 1, 2, \ldots, n$, of the space U, namely,

$$u = u_1 + u_2 + \ldots + u_n \tag{3.7}$$

Numerical algorithms are designed so that u_i, $i = 1, 2, \ldots, n$, may be obtained in parallel;

- Domain decomposition: In this algorithmic paradigm, one decomposes the original physical domain Ω into n subdomains, namely,

$$\Omega \supseteq \Omega_1 \cup \Omega_2 \cup \cdots \cup \Omega_n \tag{3.8}$$

The form of the original differential operator is preserved in each subdomain and the smaller problems defined in the subdomains are coupled only at their common boundaries (interfaces).

Three examples are furnished in [130] corresponding to these three types of decomposition just described, which are the classical alternating direction implicit (ADI) scheme, the spectral method and the additive Schwarz overlapping domain decomposition method. Generally speaking, by employing domain decomposition

algorithms, each processor operates on a subset of the data and the amount of data flow for sustaining the parallel computation among processors is rather small. However, for the other two forms of decomposition, each processor performs a subset of the calculations operated on all of the data. Hence, the expensive global data flow or exchange costs proportional to the discrete problem size are expected.

According to the classification proposed in [237], there are broadly two types of strategies for partitioning a given task for distribution across the available processors, namely, (1) partitioning of the computation, and (2) partitioning of the data. Although these two types of partitioning are not mutually exclusive, domain-based parallelism may be roughly classified into the latter while the other two decomposition approaches usually fall into the former type.

3.4 Motivations for Domain Decomposition

During the past decade, there has been a significant increase of research activities in the area of domain decomposition. The primary motivation for developing and extending this "old" technique is, no doubt, to exploit potentially high level parallelism it can offer and to take advantage of the commercially available high-performance parallel computers.

In fact, from the discussion above, domain decomposition algorithms are generally superior to either function-space or operator decomposition from the perspective of parallel computing[2]. Thus, for the purpose of parallel computing, the original physical domain is often divided into a number of regular subdomains in such a

[2]see [130, p. 9] and references therein for a more detailed exposition in terms of the interprocessor data flow analysis. In particular, it can be argued that, among aforementioned three decomposition strategies, only domain-based decomposition does not require the global movement of a significant amount of data between processors.

way that computational work may be approximately equally distributed across all subdomains (processors).

However, even for sequential computing, there are good motivations for this revival of interest in domain decomposition. Some of these, among others, are listed below:

1. Domain decomposition techniques offer a possible way to solve PDE's defined on irregular domains with a finite difference or spectral method [136, 201]. To this end, one divides the original problem domain into subdomains of simpler structure on which standard solvers (possibly, fast subdomain solvers based on fast Fourier transform or cyclic reduction techniques) are effective.

2. Domain decomposition techniques allow us to use different numerical schemes, orders of approximation and resolutions for different subdomains (see, for example, [137]). Thus they offer opportunities to combine the advantages of finite element, spectral and multigrid methods for devising more efficient and accurate algorithms applicable to multi-processor architectures (see [224] and references therein).

3. The technique also provides us with possible means for isolating regions which require special treatment. Thus the computational effort can be focused upon regions where large gradients, boundary layers, shocks or even singularities occur [96, 138], by, for example, carrying out local adaptive mesh refinement.

4. The technique may be applied to some physical problems which require different mathematical models for different subdomains, for instance, in fluid dynamics, using a viscous model near the boundary and an inviscid model in

the far field [62, 97]. Interested readers are also referred to [196] and references therein.

3.5 Some Domain Decomposition Algorithms

Domain decomposition consists of two major classes of techniques which are characterized by the way subdomains are partitioned, namely, with and without overlapping of the subdomains. For both overlapping and nonoverlapping domain decomposition methods, the original physical domain can be constructed as a union of strips or boxes, while the corresponding subdomain solvers may be exact or inexact. Box-wise domain decomposition methods are more likely to be used on parallel computers with a very large number of loosely coupled processors, while partitioning of the original domain into strips is better suited for parallel computers with a small number of powerful vector processors, like the CRAY Y-MP (see [157] for more information).

It should be pointed out that the overlapping and nonoverlapping approaches are not fundamentally different. In some cases and under some conditions, for a given Schwarz overlapping algorithm there corresponds a Schur complement nonoverlapping domain decomposition algorithm, with a particular preconditioner, which produces identical iterates on the interfaces [21, 42].

3.5.1 The Multiplicative Schwarz Overlapping Domain Decomposition Algorithm

A typical example of overlapping methods is the Schwarz alternating procedure which is now discussed in detail. Consider solving an elliptic PDE $Lu = f$ defined in domain $\Omega = \Omega_1 \cup \Omega_2$ shown in Figure 3.2, subject to some specified boundary condition on $\partial\Omega$. The restricted problems of the original problem to subdomains

Ω_1 and Ω_2 can be independently solved as long as the boundary conditions are correctly specified on the artificially introduced interfaces γ_1 and γ_2. These boundary conditions are iteratively updated until a desired accuracy is obtained by repeatedly solving subdomain problems as follows:

1. Start with an initial guess on one of the interfaces, say, γ_1;
2. Solve the problem $Lu = f$ on Ω_1 using the boundary condition on γ_1 together with the original boundary condition on the rest of the boundary;
3. Based on the solution from step (2), the boundary condition on γ_2 is updated;
4. Solve the problem $Lu = f$ on Ω_2 using the boundary condition on γ_2 together with the original boundary condition on the rest of the boundary;
5. Based on the solution from step (4), the boundary condition on γ_1 is updated, go back to step (2).

The convergence of the above procedure is guaranteed [144] using an analysis in terms of projections in Hilbert spaces. The rate of convergence depends on the extent of overlapping. The larger the region $\Omega_{12} = \Omega_1 \cap \Omega_2$, the faster the convergence. Some numerical results for the Possion's equation defined on a unit square can be found in [157]. The procedure may obviously be extended to the case of more than two subdomains, although there is a decrease in convergence rate as the number of subdomains increases. Moreover, a classical red/black numbering of subdomains may parallelize the procedure.

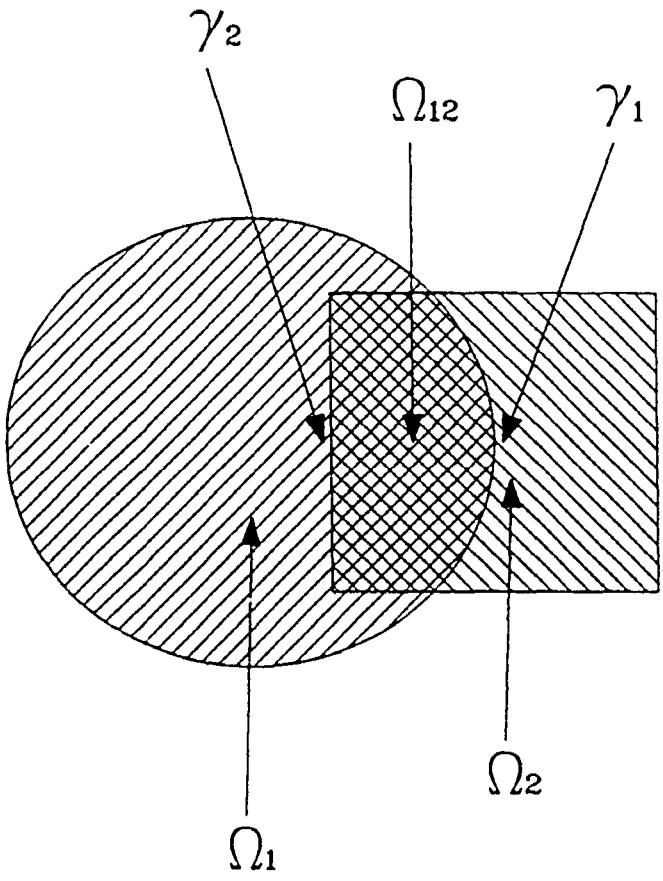


Figure 3.2: The union of two overlapped regions $\Omega = \Omega_1 \cup \Omega_2$ in which the solution of a PDE is sought.

3.5.2 The Additive Schwarz Overlapping Domain Decomposition Algorithm

The additive Schwarz algorithm [32, 33, 71, 72, 73, 74, 75, 130, 151], proposed for the solution of elliptic PDE's with the Galerkin finite element method, is an approach for constructing domain decomposed preconditioners by deriving an alternative linear system which has the same solution as the original problem. As the simplest example, the additive Schwarz method without overlapping corresponds to transforming the original linear system into a block-Jacobi preconditioned linear system.

Krylov or conjugate gradient-like algorithms are usually employed for the solution of this transformed linear system of algebraic equations. At each iteration, one solves a global coarse grid finite element problem and a number of local problems defined in the overlapping subdomains, which collectively span the original physical domain. The local subdomain problems may be solved either exactly or approximately and in parallel. This approach is very competitive for certain classes of elliptic problems due to its use of overlapping subdomains and the incorporation of a global coarse mesh. The introduction of additive Schwarz algorithms was motivated by the error propagation operator (a polynomial of projections) associated with the multiplicative Schwarz method.

To fix these ideas, let us consider solving a steady state heat conduction problem specified by the following equations

$$\frac{\partial}{\partial x_i}(k_{ij}\frac{\partial u}{\partial x_j}) + f = 0 \quad \text{in } \Omega \tag{3.9}$$

$$u = \bar{u} \quad \text{on } \Gamma_u \tag{3.10}$$

$$k_{ij}\frac{\partial u}{\partial x_j}n_i = \bar{q} \quad \text{on } \Gamma_q \tag{3.11}$$

where k_{ij}'s, $i,j = 1,2$ are conductivities and u is the temperature. Note that Einstein's summation convention has been used here.

If we define a bilinear form

$$a(w,u) = \int\int_{\Omega} k_{ij}\frac{\partial w}{\partial x_i}\frac{\partial u}{\partial x_j}d\Omega \tag{3.12}$$

and two inner products

$$(w,f) = \int\int_{\Omega} wf\,d\Omega \tag{3.13}$$

$$(w,q)|_{\Gamma_q} = \int_{\Gamma_q} w\bar{q}\,d\Gamma \tag{3.14}$$

our original boundary value problem (3.9), (3.10) and (3.11) (known as the strong form) can be shown to be equivalent to the following so-called weak form

$$a(w,u) = (w,f) + (w,\bar{q})|_{\Gamma_q} \tag{3.15}$$

Here u and w belong to properly defined Sobolev spaces V and W of trial solutions and test functions, respectively (see Appendix A).

To solve numerically the heat conduction problem by the Galerkin finite element method, we need to construct a finite dimensional space W^h to approximate the function space W defined as

$$W = \{w \in H^1(\Omega) \,|\, w = 0 \text{ on } \Gamma_u\} \tag{3.16}$$

$$H^1(\Omega) = \{w \,|\, w \in L^2(\Omega);\ \nabla w \in L^2(\Omega)\} \tag{3.17}$$

Assume that the function $g^h = \bar{u}$ on Γ_u and let $u^h = v^h + g^h$ for $v^h \in W^h$. Then the Galerkin finite element formulation may be stated as follows:

- Find $u^h = v^h + g^h$, $v^h \in W^h$, such that for all $w^h \in W^h$,

$$a(w^h,v^h) = (w^h,f) + (w^h,\bar{q})|_{\Gamma_q} - a(w^h,g^h). \tag{3.18}$$

From (3.18), we deduce that $a(w^h, v) = a(w^h, v^h)$. As a result, the finite element solution v^h (from which, we obtain $u^h = v^h + g^h$ for the solution of the original problem) is the projection (with respect to the inner product definition in (3.12)) of the exact solution v onto the finite element space W^h.

To be specific, let us think of W^h as a piecewise linear triangular element function space (see Appendix A). Upon specifying a set of global shape functions N_i's in the space W^h, (3.18) reduces to a linear system of algebraic equations

$$Av = g. \tag{3.19}$$

We now decompose the original physical domain Ω into N nonoverlapping subdomains Ω_i, $i = 1, 2, \ldots, N$ (see Figure 3.3). In addition to the h-level finite element space W^h, we define another piecewise linear H-level function space W^H, whose typical element w^H is continuous in Ω, linear in the subdomain Ω_i and vanishes on Γ_u. To obtain some overlapping, we extend each subdomain to a larger region Ω_i' which does not cut through any h-level elements. Subdomain extensions lying outside the original physical domain are cut off (see Figure 3.3). Associated with each extended subdomain Ω_i' is a finite element space $W_i^h = H_0^1(\Omega_i') \cap W^h$, which is inherited from the already defined h-level function space W^h.

The finite element function space W^h may be represented as the sum of the following $N + 1$ subspaces

$$W^h = W_0^h + W_1^h + \cdots + W_N^h \tag{3.20}$$

where W^H is replaced by W_0^h. The projection operator

$$P_i : \quad W^h \to W_i^h \tag{3.21}$$

for each $i = 0, 1, 2, \ldots, N$, may be defined as follows

$$a(w^h, P_i v^h) = a(w^h, v^h), \quad \forall w^h \in W_i^h. \tag{3.22}$$

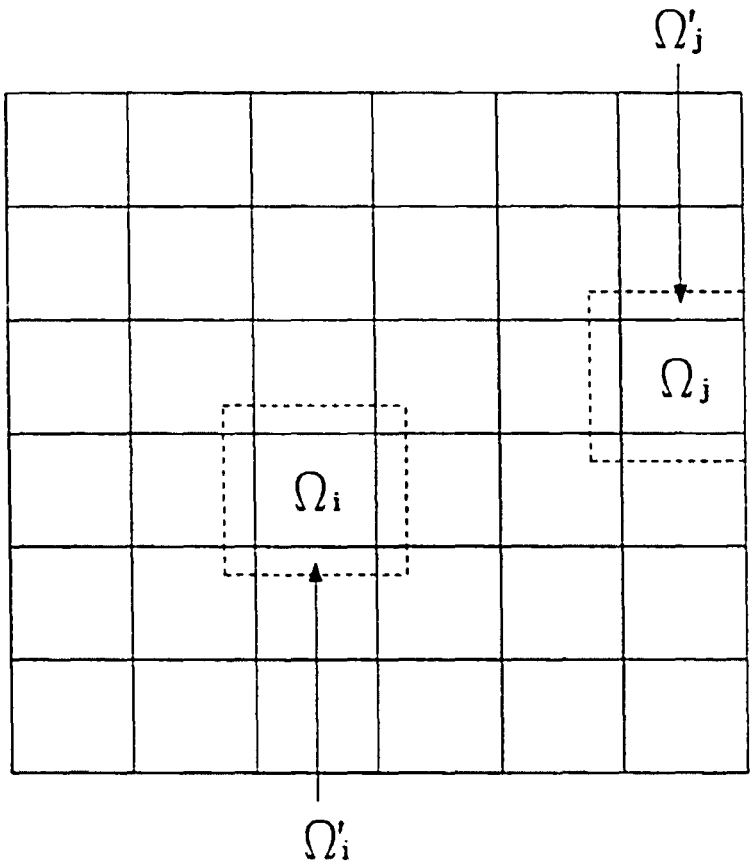


Figure 3.3: The original physical domain is decomposed into N nonoverlapping subdomains Ω_i, $i = 1, 2, \ldots, N$. To obtain some overlapping, each subdomain Ω_i is extended to a larger one Ω'_i.

Instead of directly solving the linear system (3.19) resulting from (3.18), in the additive Schwarz algorithm, one works on a transformed equation of the following form

$$Pv^h = b \tag{3.23}$$

where P is the sum of $N+1$ projection operators defined above in (3.22), namely, $P = \sum_{i=0}^{N} P_i$. The right hand side $b = \sum_{i=0}^{N} P_i v^h$, where $P_i v^h$, for each of $i = 0, 1, \ldots, N$, is obtained by solving

$$a(w^h, P_i v^h) = (w^h, f) + (w^h, \bar{q})|_{\Gamma_q} - a(w^h, g^h), \quad \forall w^h \in W_i^h. \tag{3.24}$$

Algebraically, the equation (3.23) is nothing but a transformed linear system of (3.19)

$$B^{-1}Av = B^{-1}g \tag{3.25}$$

where

$$B^{-1} = \sum_{k=0}^{N} R_k^T (A_k)^{-1} R_k \tag{3.26}$$

A typical entry of the subdomain matrix A_k is defined by $(A_k)_{ij} = a(N_i, N_j)$, where N_i's are piecewise linear global shape functions which span the space W_k^h. Each matrix R_k, $k = 1, 2, \ldots, N$, plays the role of restricting the global solution vector to the interior of the extended subdomain Ω_k'. Finally, the matrix R_0 serves as a fine-to-coarse grid restriction operator.

3.5.3 The Iteration-by-Subdomain Nonoverlapping Domain Decomposition Algorithm

As one of many possible nonoverlapping domain decomposition methods, we describe the so-called iteration-by-subdomain method [91, 150, 194, 195]. The method allows for the reduction of the original problem into a number of independent subproblems of reduced computational complexity at the differential equation level.

rather than at the algebraic level, by enforcing proper transmission of information (according to some applicable physical laws) between adjacent subregions. Because the method is based on solid physical ground, the resulting domain decomposition algorithm is rather robust and may be applied to a wide range of physical problems of practical interest.

To illustrate these ideas, we consider the physical domain Ω shown in Figure 3.4. The original domain Ω is divided into two nonoverlapping subdomains of smaller size Ω_1 and Ω_2 with an artificially introduced interface γ. The model problem under consideration is $Lu = f$ with some specified boundary condition, say $u = g$, on $\partial\Omega$, where $L = -\sum_{i,j=1}^{2} \frac{\partial}{\partial x_j}(a_{ij}\frac{\partial}{\partial x_i}) + a_0$. Denoting by u_i the restriction of u to Ω_i, for $i = 1, 2$, the original problem is equivalent to

$$Lu_1 = f \quad \text{in } \Omega_1 \tag{3.27}$$

with the original boundary condition on $\partial\Omega_1 - \gamma$, i.e.,

$$u_1 = g \quad \text{on } \partial\Omega_1 - \gamma \tag{3.28}$$

and

$$Lu_2 = f \quad \text{in } \Omega_2 \tag{3.29}$$

with the original boundary condition on $\partial\Omega_2 - \gamma$, i.e.,

$$u_2 = g \quad \text{on } \partial\Omega_2 - \gamma \tag{3.30}$$

provided that the following transmission conditions are imposed on the interface γ

$$\Phi(u_1) = \Phi(u_2) \quad \text{on } \gamma \tag{3.31}$$

and

$$\Psi(u_1) = \Psi(u_2) \quad \text{on } \gamma \tag{3.32}$$

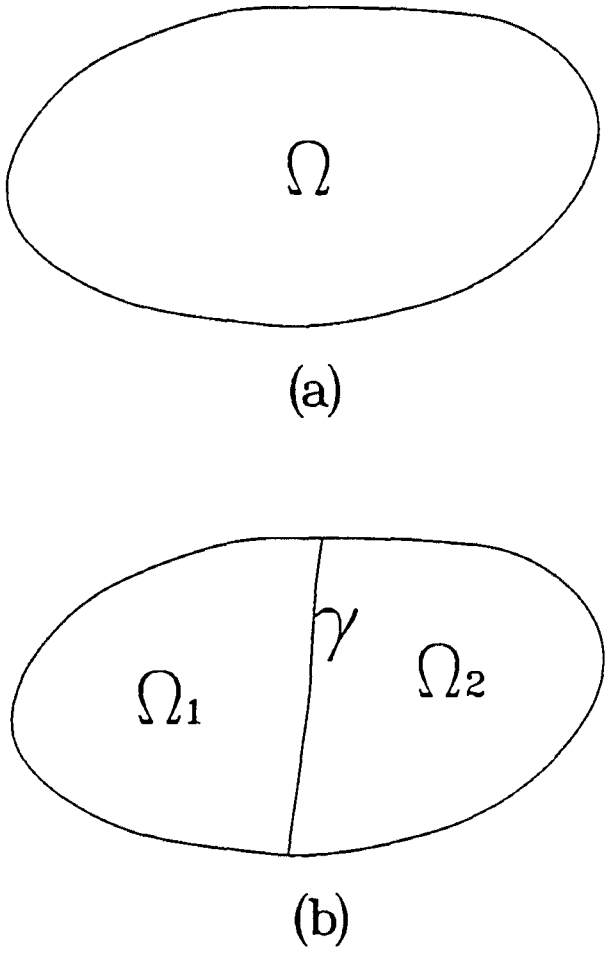


Figure 3.4: The original physical domain Ω in (a) is divided into two nonoverlapping subdomains Ω_1 and Ω_2 in (b).

For the Possion's equation, $\Phi(u) = u$ and $\Psi(u) = \frac{\partial u}{\partial n}$, where n is the unit normal vector on γ directed from Ω_1 to Ω_2. In the case of linear elasticity, by using Einstein's summation convention, we have the following well-posed problems in the domain Ω (see [139] for the meaning of notations and additional details)

- Equations of motion

$$\sigma_{ij,j} + \rho f_i = \rho \ddot{u}_i \tag{3.33}$$

- Stress-strain relationships

$$\varepsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}) \tag{3.34}$$

- The constitutive law

$$\sigma_{ij} = \frac{\partial W}{\partial \varepsilon_{ij}} = c_{ijkl}\varepsilon_{kl} \tag{3.35}$$

or for isotropic materials

$$\sigma_{ij} = \lambda u_{k,k}\delta_{ij} + 2\mu\varepsilon_{ij} \tag{3.36}$$

- The boundary condition of specified displacements

$$u_i = \bar{u}_i \quad \text{on } \partial\Omega_u \tag{3.37}$$

- The boundary condition of specified stresses

$$t_i = \bar{\sigma}_{ij}n_j \quad \text{on } \partial\Omega_\sigma \tag{3.38}$$

Appropriate initial conditions are also required for this initial boundary value problem. The interfacial transmission conditions consist physically of the continuity of displacements and stresses across the common interface, i.e.,

$$u_i^1 = u_i^2 \tag{3.39}$$

$$\sigma_{ij}^1 n_j = \sigma_{ij}^2 n_j \tag{3.40}$$

where n_i is a unit normal vector to the interface γ shown in Figure 3.2. Hence, in this case, $\Phi(u_i) = u_i$ and $\Psi(u_i) = \sigma_{ij} n_j$.

The iteration-by-subdomain algorithm assumes the following form, for $k = 0, 1, \ldots$ until convergence,

$$\begin{cases} Lu_1^{k+1} = f & \text{in } \Omega_1 \\ u_1^{k+1} = g & \text{on } \partial\Omega_1 - \gamma \\ \Phi(u_1^{k+1}) = \theta\Phi(u_2^k) + (1-\theta)\Phi(u_1^k) & \text{on } \gamma \end{cases} \tag{3.41}$$

$$\begin{cases} Lu_2^{k+1} = f & \text{in } \Omega_2 \\ u_2^{k+1} = g & \text{on } \partial\Omega_2 - \gamma \\ \Psi(u_2^{k+1}) = \Psi(u_1^{k+1}) & \text{on } \gamma \end{cases} \tag{3.42}$$

where $\theta > 0$ is a relaxation or acceleration parameter and u_1^0 and u_2^0 are given initially.

The algorithm just presented for the case of two subdomains is sequential, i.e., calculations in Ω_1 and Ω_2 can not be carried out simultaneously. However, this does not really matter, since parallel computing involving only two subdomains is of little interest. In the case of n subdomains (n being much larger than 2), the subdomains may be renumbered as red/black (see Figure 3.5) such that calculations in different subdomains with the same color can be carried out in parallel.

The iteration-by-subdomain algorithm is also known as the Dirichlet-Neumann method. It is interesting to note that, at the discrete level, the method can be related by the Poincaré-Steklov's operator [4, 197] to the Schur complement (to be introduced in Chapter 4) of a linear system of algebraic equations. Precisely, an iteration-by-subdomain iterative procedure can be shown to be equivalent to a preconditioned Richarson's iterative method for the solution of a linear system with the coefficient matrix being the Schur complement matrix C.

red
black
red
black
red
black
red
black

black	red	black	red
red	black	red	black
black	red	black	red
red	black	red	black

Figure 3.5: The red/black subdomain numbering for strip-wise and box-wise domain decomposition.

3.6 Conclusions

- Although a tremendous variety of parallel numerical methods have been proposed for solving PDE's over the past thirty years, the most recently invented parallel computational strategies for the numerical solution of PDE's are largely based on or closely related to domain decomposition principles.

- Among many other desirable properties, the domain decomposition method extends the usefulness of some special numerical techniques (for example, fast direct solvers). The local applicability of these special numerical techniques makes the domain decomposition method attractive even for serial computing. The possibility of mapping subdomain calculations onto different processors for parallel processing puts a further premium on the application of the method.

- Decomposition by domains is the best among three possible decomposition strategies for the parallel solution of PDE's, namely, operator decomposition, function-space decomposition and domain decomposition. The nature of domain decomposition techniques guarantees that only a small volume of data (relative to the scale of the discretization) needs to be exchanged between processors and the global remapping of the data onto processors is avoided.

- The iteration-by-subdomain, multiplicative and additive Schwarz domain decomposition methods introduced in this chapter as well as the Schur and the modified interface matrix (advocated here and proposed before by the author) domain decomposition methods along with parallel block preconditioning techniques to be introduced in later chapters are some general approaches of domain-based decomposition methods for solving elliptic PDE's or time evolution problems (of parabolic or hyberbolic type) discretized with implicit

temporal schemes. These general approaches, with a modification of one or more of their ingredients, provide almost infinitely many variants of the so-called iterative domain decomposition algorithms. Another dimension will be added to this variety if implementation details are taken into account.

CHAPTER 4

THE SCHUR DOMAIN DECOMPOSITION METHOD AND ITS APPLICATIONS TO THE FINITE ELEMENT NUMERICAL SIMULATION OF THE SHALLOW WATER FLOW

4.1 Introduction

We have pointed out and illustrated two approaches of domain decomposition methods, namely overlapping and nonoverlapping. Generally speaking, the multiplicative Schwarz overlapping approach is rather robust. It can be applied to various difficult physical problems of practical interest. With this approach, different numerical schemes, different mesh resolutions or even different mathematical models in different subdomains may easily be used or incorporated into the formulation. However, it is usually less efficient than a nonoverlapping domain decomposition approach designed for a specific application which sets up an iterative procedure accelerated by appropriate preconditioners.

To reduce the serial complexity of the nonoverlapping domain decomposition algorithms, most of the research, up to now, has almost exclusively focused on the interface(s), or more specifically, on finding good preconditioners for the conjugate gradient (CG) algorithm [52] or for any competitive iterative method (GMRES [206], for example) for symmetric or non-symmetric linear systems of algebraic equations, arising from finite difference or finite element discretizations of elliptic partial differential equations in two or three dimensional regions. To recommend just a few

papers, see [20, 34, 38, 43, 44, 46, 69, 70, 99, 131, 133, 149, 157, 158, 159, 173, 174] for more details.

The primary reason for this focus is that the iterative solution of the interface Schur complement matrix system involves repeated solutions of all the subdomain problems (see Figure 4.5 for a highly simplified data dependency graph) and the interface solver itself is a potential bottleneck for the coarse-grained parallelism. A good interface preconditioner can drastically reduce the number of iterations on the interfaces, thus allows significant saving of computational work in the subdomains and in the whole solution process.

Due to the paramount importance of the shallow water equations in meteorology and oceanography, where they serve as test models for the development of new algorithms, the efficient finite element solution of the shallow water equations has attracted the interest of many researchers. A tremendous amount of work has been carried out in this direction, see, for example, [58, 166, 167, 168, 169, 170, 172, 175, 177, 179, 181, 217], to cite but a few references. Unfortunately, these algorithms were not designed to run efficiently on various multi-processor architectures.

In this chapter we extend applicability of nonoverlapping domain decomposition methods to a set of coupled nonlinear hyperbolic shallow water partial differential equations defined on a 2-D limited-area domain, using finite element discretization in space and an implicit integration scheme in time (see Appendix A).

Specifically, we report on our work on the Schur domain decomposition method applied to a finite element model (Appendix A) of the nonlinear shallow water equations over a limited-area domain. We begin with the idea of substructuring and the Schur complement matrix. Then we gradually introduce tools and algorithms designed to solve efficiently the capacitance linear systems associated with interface nodal variables. Numerical results and discussions concerning the finite element

domain decomposition solution of a set of nonlinear shallow water equations are given in Section 4.7. Considerations related to parallelization will be postponed to a later chapter for a unified treatment.

4.2 Substructuring and the Schur Complement

It is well known, in the finite element method, that internal degrees of freedom can be condensed out at the element level prior to the assembly process (see, for example, [246]). When this idea is applied to a group of elements, i.e., a substructure or a subdomain, it leads to what is known among engineers as the substructuring techniques.

The idea is that the whole structure or domain is considered to be an assembly of substructures or subdomains (see, for example, [93, 193, 202]). Each substructure or subdomain, in turn, is idealized as an assembly of finite elements, and all internal degrees of freedom are statically condensed out (see also [241]).

To fix ideas, two classes of variables are usually identified, namely the internal variables relevant to nodes within subdomains, and the interface variables relevant to nodes belonging to two or more subdomains. The internal variables may be numbered either before or after the interface ones.

We only consider solving time dependent PDE's with implicit time discretization. Explicit methods are algebraically equivalent to a matrix-vector product problem and thus readily parallelizable. However, implicit time schemes involve matrix inversions and, consequently, are highly sequential. As a standard practice in parallel computing (see Chapter 2), a renumbering or reordering strategy may be used for restructuring the sequence of calculations in such a way as to reveal the parallel structure of the problem.

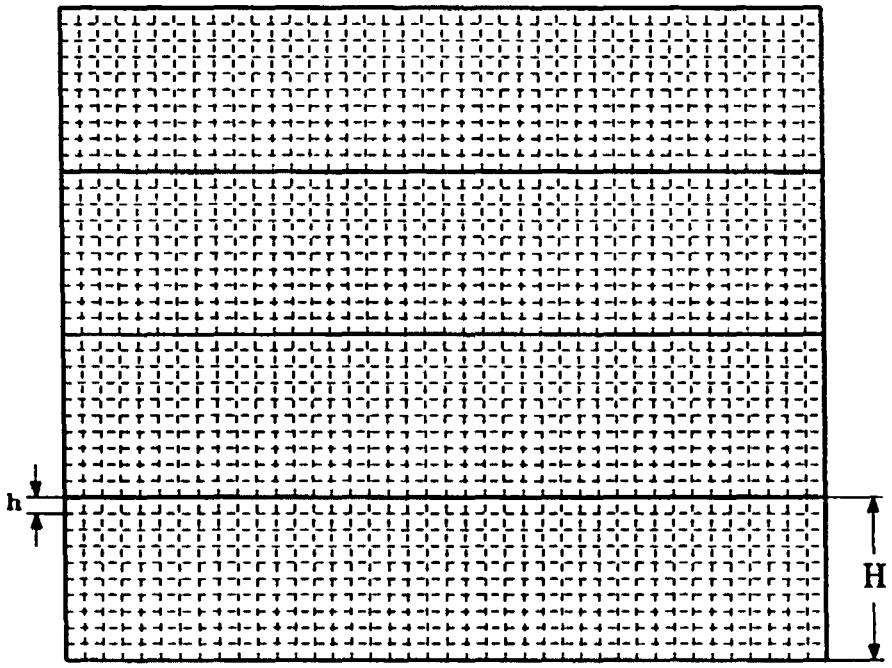


Figure 4.1: The original domain Ω is decomposed into four subdomains of equal or nearly equal sizes with a quasi-uniform subdomain width H and quasi-uniform grid size h.

Here we are particularly interested in the computational speedup resulting from applying the domain decomposition technique. Our target machine for the implementation of various algorithms is the one with a number of powerful vector processors sharing a common memory, such as the CRAY computer family. For this type of computer architecture, it is more suitable to divide the physical domain into strips instead of small boxes in order to yield longer vectors [157]. Thus, we will consider subdividing the domain Ω under consideration into n nonoverlapping horizontal strips $\Omega_i, i = 1, \ldots, n$ of equal or nearly equal sizes (see Figure 4.1). The $n-1$ interfaces $\Gamma_i, i = 1, \ldots, n-1$ separating these n subdomains from each other are collectively denoted by Γ. We have the following relations

$$\Omega = \Omega_1 \cup \Omega_2 \cup \cdots \cup \Omega_n \cup \Gamma \tag{4.1}$$

$$\Omega_i \cap \Omega_j = \phi \quad \text{for } i \neq j \tag{4.2}$$

$$\Gamma = \Gamma_1 \cup \Gamma_2 \cup \cdots \cup \Gamma_{n-1} \tag{4.3}$$

Clearly, cross points are eliminated from our consideration.

The differential operator governing the problem on Ω can be split up into operators acting on the interfaces Γ and the n subdomains $\Omega_i, i = 1, \ldots, n$ at each time step, as can be realized by identifying two types of variables and renumbering. To fix ideas, let us consider solving PDE's involving only first and/or second order partial derivatives, which are typical of computational fluid dynamics (CFD) problems, by employing numerical schemes having a five point finite difference or a seven point linear triangular finite element stencil[1] as shown in Figure 4.2. If we denote the matrix representations of these reduced operators as $A_{ii}, i = 1, \ldots, n$ on each of the subdomains and A_{ss} on the interfaces Γ, we obtain systems of algebraic equations

[1] A nine point linear rectangular finite element stencil may similarly be taken into account.

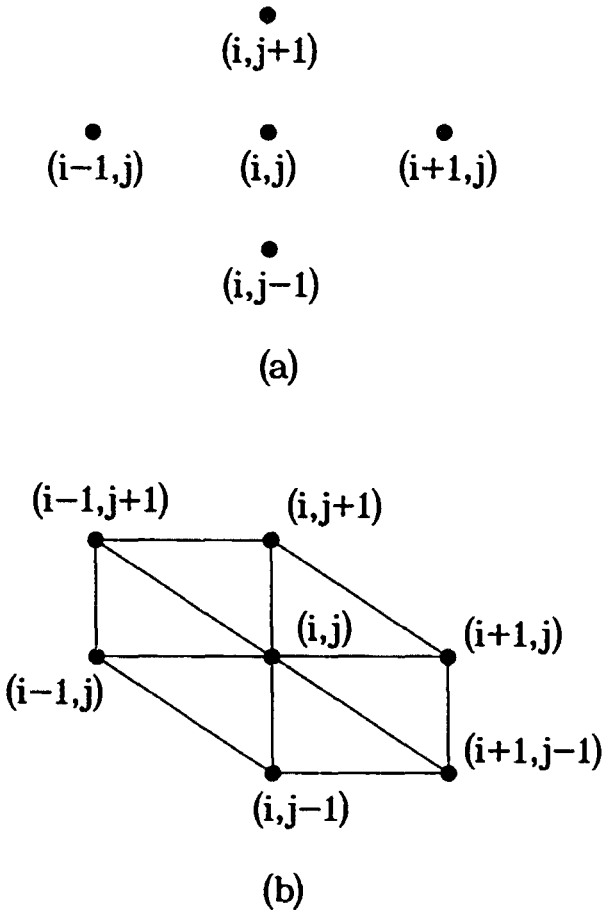


Figure 4.2: (a) A five point finite difference stencil; (b) A seven point linear triangular finite element stencil.

at each time step of the following form

$$Ax = f \tag{4.4}$$

where the matrix A assumes the following block-bordered structure:

$$A = \begin{bmatrix} A_{dd} & A_{ds} \\ A_{sd} & A_{ss} \end{bmatrix} \tag{4.5}$$

In (4.5), A_{dd} is a block diagonal matrix

$$A_{dd} = \text{diag}[A_{11}, A_{22}, \ldots, A_{nn}] \tag{4.6}$$

with each block A_{ii}, for $i = 1, 2, \ldots, n$, being the discrete analog of the restriction of the original differential operator on each subdomain.

A_{ds} and A_{sd} represent connections between subdomains to interfaces. They assume the following block bi-diagonal forms

$$A_{ds} = \begin{bmatrix} E_1 & & & & \\ F_2 & E_2 & & & \\ & F_3 & \ddots & & \\ & & \ddots & E_{n-1} & \\ & & & F_n & \end{bmatrix} \tag{4.7}$$

and

$$A_{sd} = \begin{bmatrix} G_1 & H_2 & & & \\ & G_2 & H_3 & & \\ & & \ddots & \ddots & \\ & & & G_{n-1} & H_n \end{bmatrix} \tag{4.8}$$

where

$$E_i = (\underbrace{0, 0, \ldots, 0}_{m_i - 1 \text{ blocks}}, E_{i,m_i})^T \tag{4.9}$$

$$F_i = (F_{i,1}, \underbrace{0,0,\ldots,0}_{m_i-1 \text{ blocks}})^T \tag{4.10}$$

$$G_i = (\underbrace{0,0,\ldots,0}_{m_i-1 \text{ blocks}}, G_{i,m_i}) \tag{4.11}$$

$$H_i = (H_{i,1}, \underbrace{0,\ldots,0,0}_{m_i-1 \text{ blocks}}) \tag{4.12}$$

m_i being the number of horizontal grid lines in the i'th subdomain. The blocks E_{i,m_i}, $F_{i,1}$, G_{i,m_i} and $H_{i,1}$ in matrices E_i, F_i, G_i and H_i, respectively, are either diagonal or bi-diagonal point matrices, depending on whether a five point finite difference scheme or a seven point stencil resulting from a linear triangular finite element method being used.

Note the matrix A consists essentially of the assembly of the subdomain stiffness matrices, see also [20]. If we let n_i, $i = 1,2,\ldots,n$, be the number of unknowns in each of the subdomains and n_s be the number of unknowns on the interfaces, then each of the matrices A_{ii}, A_{is} and A_{si} is of size $n_i \times n_i$, $n_i \times n_s$ and $n_s \times n_i$, respectively, for $i = 1,2,\ldots,n$. Likewise A_{ss} is of the size $n_s \times n_s$.

A_{ss} corresponds to the discretization of the original differential operator restricted to the interfaces. The interfaces include $n-1$ internal boundaries Γ_i, $i = 1,2,\ldots,n-1$, and no mesh points from different internal boundaries will appear in the same stencil. This accounts for the special structure of the matrix A_{ss}. For instance, A_{ss} is diagonal for a three point stencil 1-D problem and the following block diagonal form for a five point finite difference or a seven point triangular finite element stencil in the 2-D case

$$A_{ss} = \text{diag}[T_{11}, T_{22}, \ldots, T_{n-1,n-1}] \tag{4.13}$$

where each block is associated with one of the interfaces.

The non-zero entries of T_{ii}, $i = 1, 2, \ldots, n-1$, amount to the following cyclic tridiagonal structure induced by the presence of periodic boundary conditions imposed on our physical problem (to be detailed in Section 4.6)

$$T_{ii} = \begin{bmatrix} b_1^{(i)} & c_1^{(i)} & & & a_1^{(i)} \\ a_2^{(i)} & b_2^{(i)} & c_2^{(i)} & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1}^{(i)} & b_{n-1}^{(i)} & c_{n-1}^{(i)} \\ c_n^{(i)} & & & a_n^{(i)} & b_n^{(i)} \end{bmatrix} \tag{4.14}$$

By plotting the surface formed by the entries of a matrix as a function of its indices, the typical block-bordered structure of the matrix A given by (4.5) corresponding to a substructure numbering of the nodes is generated for a four subdomain case (see Figure 4.3), where the following modifications are made to the entries of the matrix A:

$$a_{ij} \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } a_{ij} \neq 0 \\ 0 & \text{if } a_{ij} = 0 \end{cases} . \tag{4.15}$$

The numerical solution of $Ax = f$ is equivalent to solving the following

$$Cx_s = g \qquad \text{on } \Gamma \tag{4.16}$$

$$A_{ii}x_i = f_i - A_{is}x_s \qquad \text{in } \Omega_i \tag{4.17}$$

where Γ and Ω_i, $i = 1, 2, \ldots, n$, stand for the interfaces and subdomains, respectively and

$$C \stackrel{\text{def}}{=} A_{ss} - A_{sd}A_{dd}^{-1}A_{ds} \tag{4.18}$$

$$= A_{ss} - \sum_{i=1}^{n} A_{si}A_{ii}^{-1}A_{is} \tag{4.19}$$

and

$$g \stackrel{\text{def}}{=} f_s - A_{sd}A_{dd}^{-1}f_d \tag{4.20}$$

$$= f_s - \sum_{i=1}^{n} A_{si}A_{ii}^{-1}f_i \tag{4.21}$$

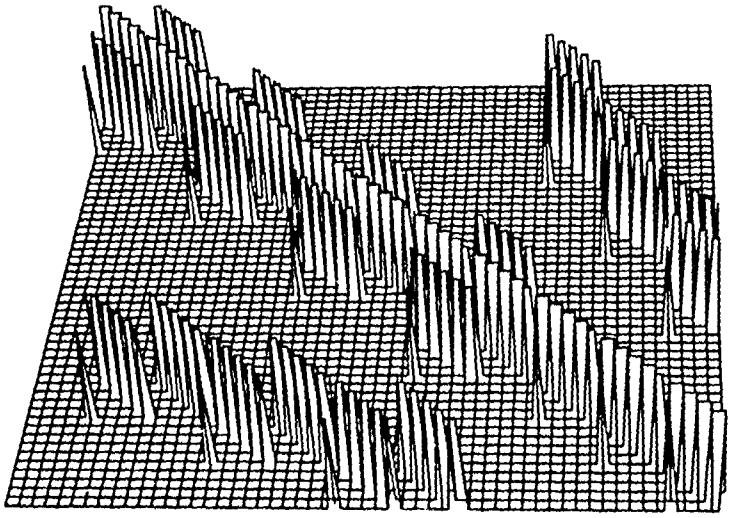


Figure 4.3: The typical block-bordered matrix structure corresponding to a substructure numbering of the nodes for a four-subdomain domain decomposition. Since the relative magnitudes of entries in the matrix A are of no interest here, all non-zero elements ($a_{ij} \neq 0$) have been set to 1.

where C is the well-known Schur complement matrix, capacitance matrix or Gauss transform in different contexts (see [53, 119, 185]).

It is clear that the subdomain problems (4.17) are trivially decoupled (independent of each other) and the subdomain solutions may be sought in a highly parallel way once the internal boundary conditions x_s on the interfaces Γ between artificially divided subdomains are obtained by solving (4.16).

It should be pointed out that the Schur complement matrix C is generally dense, although each of the matrices A_{ii}, $i = 1, 2, \ldots, n$ and A_{ss} corresponds to a lower dimensional local differential operator and hence are sparse. Consequently, the solution of (4.16) is expensive to obtain using a direct solver, especially as the degrees of freedom on the interfaces increase as higher mesh resolutions are introduced. We illustrate the denseness of the matrix C associated with the three interfaces of a four-subdomain strip-wise domain decomposition (see Figure 4.1) in Figure 4.4, where each $\times$ represents a non-zero entry of the matrix C, assuming that there are 15 nodes on each interface. The fact that the matrix C assumes a block tridiagonal structure, as clearly shown in Figure 4.4, may readily be explained in the context of the finite element method by observing that the Schur complement matrix C consists essentially of the assembly of the substructure (or sometimes called a "superelement") stiffness matrices.

It is also important to note that sizes of A_{dd} and A_{ss} in (4.5) are very different. Specifically, the discrete dimensions of A_{dd} and A_{ss} are, respectively, $O(h^{-2})$ and $O(H^{-1}h^{-1})$, where H characterizes the subdomain length scale and h is the mesh size (see Figure 4.1). Thus, in general, we have a relatively smaller problem to solve on the interfaces than that in each of the subdomains. We point out that, in the case of box-wise domain decomposition, the cross points of interfaces (or equivalently, subdomain vertices) constitute a coarse grid discretization in the original physical

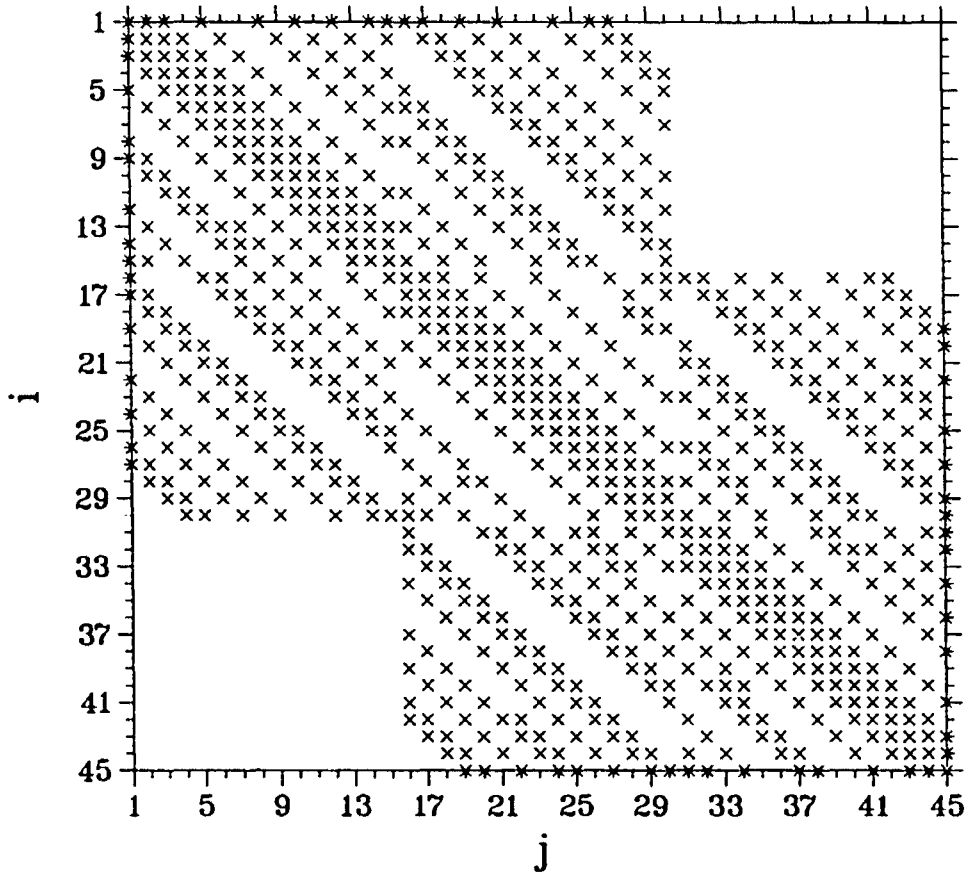


Figure 4.4: The Schur complement matrix structure associated with the three interfaces of a four-subdomain strip-wise domain decomposition shown in Figure 4.1, assuming there are 15 nodes on each interface. Here, each × represents a non-zero entry in the matrix.

domain. The discrete dimension of the matrix corresponding to the discretization of the original differential operator restricted to this coarse grid is only $O(H^{-2})$.

Finally, we mention that, for box-wise domain decomposition approach, two levels of discretization are usually considered for the original physical domain, namely, a global coarse grid constituted by subdomain vertices and a local fine grid defined within each subdomain. Therefore, two length scales, i.e., the H-scale and the h-scale, respectively, have been taken advantage of in the construction of domain decomposition preconditioners. We refer interested readers to [22, 30, 31, 101, 105] and references therein for details on two-level Krylov domain decomposition methods.

4.3 A Node Renumbering Scheme

In a computer program designed for solving partial differential equations, we may introduce a modification of the code in order to accommodate various mesh resolutions aimed at obtaining higher orders of accuracy and testing the convergence of the method.

Let us now denote the original nodal numbers by the old numbering, while the nodal numbers after renumbering (i.e. the substructured numbering) will be denoted by the new numbering.

If we discretize the partial differential equation based on the new numbering scheme, the relations between the interface nodal numbers and those nodal numbers of the subdomains adjacent to the interfaces become difficult to predict for arbitrary high mesh resolutions.

This becomes evident for the case of the finite element discretization in which the relationship amongst global nodes, local nodes and element numbering turns out to be very different if we try to formulate the problem using the new numbering systems for various mesh resolutions.

In view of this, other efficient ways for transforming the original system matrix into the block-bordered matrix like that in (4.5) need to be devised. For example, the following node renumbering scheme may be used for this purpose:

Step 1 Set up the relationship between the old and the new numbering system by defining an array number(n), where n is the total number of nodes and the number(i) is the nodal number under the new numbering scheme corresponding to the old nodal number i;

Step 2 Renumber the nodes by putting the j'th column of the original matrix into the j_t'th column of the transformed matrix, where j_t = number(j);

Step 3 Change the positions of nodal actions by putting the i'th row of the matrix obtained at step (2) into i_t'th row of the transformed matrix, where i_t = number(i).

After these three steps have been implemented we obtain the block-bordered matrix which has the same structure to that of (4.5) corresponding exactly to the new nodal numbering defined by the array number(n). This idea also provides an easy way for implementing multicoloring techniques.

A segment of computer code for calculating number(i), $i = 1, 2, \ldots, n$ is provided below for illustration purposes:

```
*
*     Renumber the nodes in the subdomains
*
      do 10 i=1,kd
        ib=(i-1)*ns
        do 20 j=(i-1)*ni+1,i*ni
```

```
          num(j+ib)=j
20       continue
10     continue
*
*      Renumber the nodes on the interfaces
*
       do 30 i=1,ks
         k=kd*ni+(i-1)*ns
         ib=i*ni+(i-1)*ns
         do 40 j=ib+1,ib+ns
           k=k+1
           num(j)=k
40       continue
30     continue
```

where ns is the number of nodes on one interface, ni is the number of nodes in the i'th subdomain, kd is the number of subdomains and ks is the number of interfaces.

4.4 Schur Domain Decomposition Algorithms

Historically, it was common practice to apply the LU or Cholesky factorization in each subdomain and form the Schur complement matrix C and its right hand side g. Thus the solution to (1.16) can be obtained, followed by that to each of the subsystems (4.17). However, explicitly obtaining the Schur complement matrix C is an expensive process involving nn_s subdomain solves, where n is the number of subdomains and n_s is the number of interfacial degrees of freedom.

The coupling between subdomains is now handled more efficiently using preconditioned iterative solvers without constructing the Schur complement matrix explicitly. However, as has been pointed out in [131], the approach based on explicitly forming the Schur complement matrix still remains a useful procedure for some cases. As an example, the computational cost of constructing the Schur complement matrix C may be amortized over multiple right-hand sides (g in (1.16)) when a discrete linear system with an identical coefficient matrix appears at each time step for a time dependent problem (see [190]).

As formulated in [185], it requires $n(n_s + 1)$ forward and back substitutions to obtain C and g. A minor improvement can be introduced here to reduce this number to nn_s. The algorithm assumes the following form:

Algorithm 4.1

Step 1 Carry out the LU decomposition for each of the subdomain matrices $A_{ii} = L_i U_i$, $i = 1, \ldots, n$. This part is highly parallel.

Step 2 Solve $Y_{si} U_i = A_{si}$ and $X_{si} L_i = Y_{si}$ row by row for $i = 1, \ldots, n$. Form $C = A_{ss} - \sum_{i=1}^{n} X_{si} A_{is}$ and $g = f_s - \sum_{i=1}^{n} X_{si} f_i$. This part can also be calculated in parallel, where X_{si} and Y_{si} are two $n_s \times n_i$ matrices.

Step 3 Solve (4.16). This is a bottleneck for parallelism.

Step 4 Solve (4.17) in parallel by using the LU decomposition of $A_{ii}, i = 1, \ldots, n$.

The aforementioned algorithm requires the formation of the Schur complement matrix explicitly, a computationally expensive procedure for most problems. Like the Schur complement matrix approach formulated by the CG algorithm, the following algorithm may be formulated for any non-symmetric iterative solver in which

only matrix-vector multiplications are required, although we specifically refer here to the conjugate gradient squared (CGS) algorithm ([216], see also Section 4.5.2).

Algorithm 4.2

Step 1 Solve

$$A_{ii}b_i = f_i \tag{4.22}$$

and form $A_{si}b_i$, $i = 1, 2, \ldots, n$ in parallel; Form the right hand side of the Schur complement matrix system

$$g = f_s - \sum_{i=1}^{n} A_{si}b_i \tag{4.23}$$

Step 2 Use the CGS algorithm to solve the Schur complement system (4.16). This is an iterative process carried out until a prescribed convergence criterion on the interfaces is met. The product of the Schur complement matrix with a vector w_s, Cw_s, may be evaluated as follows: Solve each subdomain problem

$$A_{ii}v_i = -A_{is}w_s \tag{4.24}$$

once and form the product $A_{si}v_i$ for $i = 1, 2, \ldots, n$ in parallel. Then form the product

$$Cw_s = A_{ss}w_s + \sum_{i=1}^{n} A_{si}v_i \tag{4.25}$$

Step 3 Once the interface nodal values x_s are obtained, we may solve in parallel (4.17) for each subdomain.

The above algorithm may be heuristically described as a "divide and feedback" or "divide, conquer and combine" process. Krylov or conjugate gradient-like iterative algorithms (CGS algorithm in this case) are employed for the solution of

the interfacial Schur complement linear system. The matrix-vector multiplication Cw_s is obtained by concurrently solving n smaller problems in the subdomains Ω_i, $i = 1, 2, \ldots, n$. The information gathered from the solution of each of these subdomain problems is then fed back to the interface iterative solver. If the convergence criterion is not met on the interfaces, the domain is decomposed again and the subdomain problems are solved. This "divide and feedback" process continues until the convergence criterion on the interfaces is attained.

A highly simplified control flow (data dependency) graph for the iterative Schur domain decomposition algorithm (Algorithm 4.2) is illustrated in Figure 4.5.

After convergence is attained on the interfaces $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \cdots \cup \Gamma_{n-1}$, the subdomain problems are trivially decoupled with specified boundary conditions of desired accuracy on the artificially introduced interfaces between subdomains and may be solved in parallel. Finally, the solutions in the subdomains and on the interfaces are patched together to obtain the sought-after solution in the original physical domain.

This "divide and feedback" process described above immediately indicates that the efficiency of this approach relies on the number of iterations required for obtaining convergence on the interfaces as well as on the computational cost for each subdomain solver. Preconditioning techniques must be used to reduce the number of iterations on the interfaces. In the subdomains, for the case where fast solvers do not exist, either direct solvers based on LU factorizations or iterative methods may be applied.

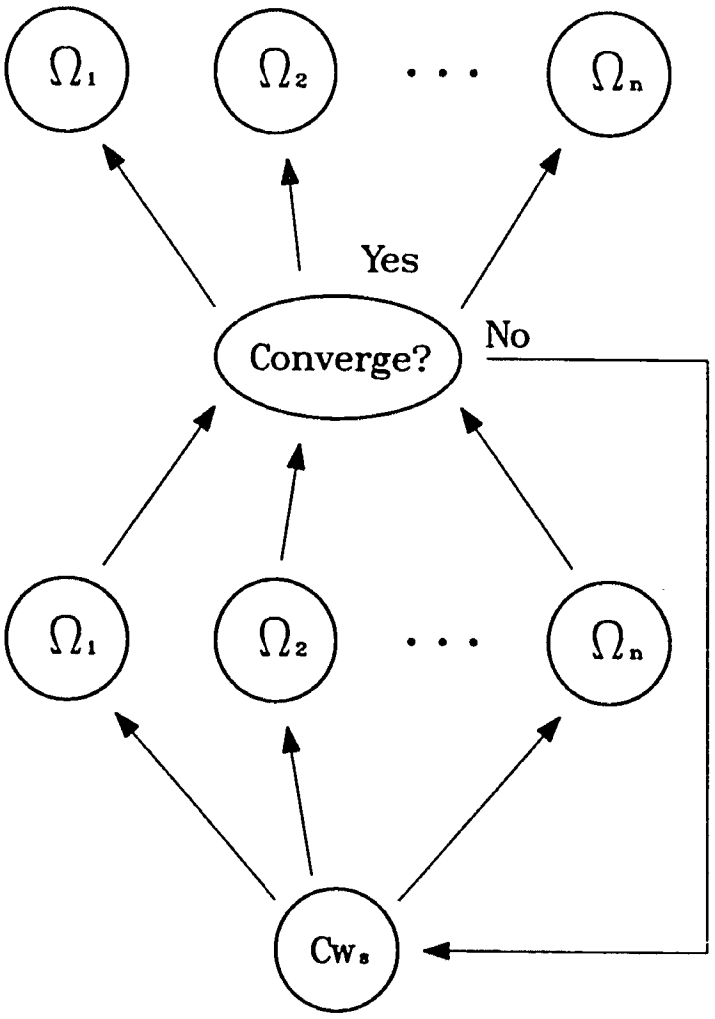


Figure 4.5: A highly simplified control flow (data dependency) graph for the iterative Schur domain decomposition algorithm.

4.5 Iterative Linear Solvers and Preconditioning Techniques for the Subdomain and Interface Problems

4.5.1 Direct Methods vs. Iterative Methods

For reasons indicated in Section 4.2, instead of using a direct solver, a Krylov or conjugate gradient-like iterative linear solver is generally employed for solving the Schur complement linear system on the interfaces. However, whether to use a direct or an iterative scheme in each of the subdomains is quite case-dependent.

Direct solving techniques for linear systems are often based on the Gaussian elimination or some variant thereof and are able to generate the exact solution (within machine accuracy) in a finite number of arithmetic operations. However, there are some fundamental drawbacks. The direct elimination will cause the phenomenon of so-called "fill-in", which results in an increase in both the computational complexity and the storage requirement, although some techniques have been developed to minimize the fill-ins (see [10], [77], [78], [147], [153] and referenced cited therein). For large problems (especially for numerical solutions of PDE's in three dimensions), peripheral storage devices may be required to store the matrix and I/O costs will dominate. In addition, the build-up of round-off errors or errors in the initial data may be rather severe for direct methods (see [9] and [10]).

In contrast, iterative methods do not suffer from the fill-in phenomenon and have the advantage of minimal storage [127], typically requiring only the nonzero entries of the matrix and a few vectors of corresponding length. For Krylov or conjugate gradient-like algorithms, all that is needed is the matrix-vector product Ax and the storage of the matrix is not required. As pointed out in [9], when these algorithms are effectively preconditioned, one may derive algorithms with almost optimal computational complexities and by a careful evaluation of the residuals, the

influence of round-off errors may be reduced by orders of magnitude as compared to direct methods. In addition, iterative methods can benefit from a good initial approximation in the design of iterative solvers for boundary value problems or time dependent problems with implicit time discretization (see [10, p. 385] and Chapter 5) and are much easier to program than direct methods. To appreciate how complicated a code may be for performing the sparse Gaussian elimination, see [77] for a segment of the Fortran code, which is directly extracted from the Harwell Subroutine Library MA28. The most recent systematic review of iterative solution techniques for large sparse linear systems can be found in [112].

As a result, we generally prefer to let iterative methods serve as subdomain solvers. In [174], we compared the relative computational costs of two domain decomposition algorithms using the same interface preconditioning, but with preconditioned CGS (PCGS) iterative and LU direct solvers, respectively, in the subdomains. The results indicate that the algorithm with LU direct subdomain solvers is less expensive than the PCGS method when the mesh resolution is relatively coarse and the ratio between the size of each subdomain problem and that of the interface problem is less than two. However, as the mesh resolutions increase and the degrees of freedom on each subdomain get larger compared to those on the interfaces, the algorithm with PCGS subdomain solvers turns out to be computationally more efficient. The results obtained there agree in some sense with those reported in [200], in which the Gauss direct solver was compared with the PCGS method. Similar results were obtained in [194] where the direct solver based on a complete LU factorization was compared with the preconditioned conjugate gradient (PCG) method.

We point out that, in most practical applications, the subdomain problems are much larger than the one defined on the interfaces. As a result, the preconditioned

iterative solution in each of the subdomains is to be preferred (at least for our case), especially for large-scale problems.

However, for certain problems with regular subdomain geometries and nice operators (see Section 3.2 in Chapter 3), fast direct solvers, based on either the fast Fourier transform or cyclic reduction, are available ([221] and [222], see also [39]). In these cases, fast direct solvers are definitely to be preferred in the subdomains. In fact, domain decomposition techniques are often applied in order to extend the usefulness of some special and powerful numerical techniques (fast solvers in this case), which are only locally exploitable in irregular regions.

4.5.2 Iterative Algorithms for Linear Systems of Algebraic Equations

The basic preconditioned iterative algorithm for solving a linear system

$$Ax = b \tag{4.26}$$

begins by defining another matrix B, called preconditioning matrix, which must be relatively inexpensive to invert. The iterative scheme has the following defect-correction type, namely, for $k = 0, 1, \ldots$

$$B\Delta x^{(k+1)} = -r^{(k)}, \quad x^{(k+1)} = x^{(k)} + \Delta x^{(k+1)} \tag{4.27}$$

where $r^{(k)} = Ax^{(k)} - b$ is the defect or residual and $\Delta x^{(k)}$ is the correction at stage k.

If we split the matrix A into $A = D - L - U$, where D is (block) diagonal part and L and U are the lower and upper (block) triangular parts of A, then we obtain the following four basic preconditioners or iterative schemes ([229] and [239])

- the (block) Jacobi preconditioner or iterative method if $B = D$;
- the (block) Gauss-Seidel preconditioner or iterative method if $B = D - L$;

- the (block) successive overrelaxation (SOR) preconditioner or iterative method if $B = \omega^{-1}D - L$;

- the (block) symmetric successive overrelaxation (SSOR) preconditioner or iterative method if $B = (2-\omega)^{-1}(\omega^{-1}D - L)(\omega^{-1}D)^{-1}(\omega^{-1}D - U)$.

To accelerate the basic scheme (4.27), one introduces acceleration parameters τ_k and considers the following more general iterative scheme

$$x^{(k+1)} = x^{(k)} - \tau_k B^{-1} r^{(k)} \tag{4.28}$$

If we introduce an iteration error $e^{(k)} = x - x^{(k)}$, then, by (4.28), $e^{(k+1)} = (I - \tau_k B^{-1}A)e^{(k)}$. Upon defining a polynomial $P_m(\lambda) = \prod_{k=0}^{m}(1-\tau_k\lambda)$, the error $e^{(m)}$ may be expressed as $e^{(m)} = P_m(B^{-1}A)e^{(0)}$. Requiring that $\|e^{(m)}\|/\|e^{(0)}\|$ be minimized leads to the well-known Chebyshev polynomial acceleration of the basic iterative schemes ([113], see also [9] and references therein).

As is well known, the most efficient acceleration method for symmetric and positive definite linear systems is the conjugate gradient (CG) algorithm [52], [60], [118] (see also [10], [100], [212] for recent expositions of this method). Today this method with suitable preconditioners is considered one of the best methods available for the iterative solution of large sparse symmetric and positive definite linear systems. We also note that vectorized conjugate gradient methods have been applied to large-scale minimization problems in meteorology [176].

The CG method is self-adaptive in the sense that the optimal parameters are calculated within the algorithm so that the error in the energy norm $\|e^{(m)}\|_A = [(e^{(m)})^T A e^{(m)}]^{1/2}$ is automatically minimized. It may be shown ([10]) that, when applying the conjugate gradient method with a preconditioner $B = EE^T$ to the

solution of (4.26), the number of iterations will not exceed

$$m = \text{int}[\frac{1}{2}\kappa^{\frac{1}{2}}\ln(\frac{2}{\epsilon}) + 1] \tag{4.29}$$

in order to achieve the result $\|x - x^{(m)}\|_A \leq \epsilon\|x - x^{(0)}\|_A$, where κ is the spectral condition number of the matrix $E^{-1}AE^{-T}$ (which is equal to the spectral condition number of $B^{-1}A$) and ϵ is a small positive real number (convergence accuracy). Moreover, when the eigen-spectrum of $E^{-1}AE^{-T}$ is clustered, the upper bound given by (4.29) may even be reduced (see [9] and references therein).

Unfortunately, this algorithm is not applicable to the solution of non-symmetric linear systems such as arise from the discretizations of non-self-adjoint elliptic partial differential equations, and of the hyperbolic system of shallow water equations we are considering here. As such, some other algorithms applicable to non-symmetric linear systems have to be considered.

A large number of generalizations were proposed in the literature (see [66, 203, 205] and references therein) for solving large linear systems with non-symmetric coefficient matrices. However, none of them may claim to be a clear winner.

In this application, we choose to apply conjugate gradient squared (CGS) algorithm for the iterative solution of systems of linear algebraic equations both in the subdomains and on the interfaces. The algorithm arranged in a form ready for computer implementation (by introducing only seven one dimensional arrays of the size of the problem) is provided below for solving $Ax = b$. Detailed theoretical and numerical comparisons between various competitive non-symmetric iterative solvers, such as Bi-CGSTAB [227], CGS [216] and generalized minimal residual (GMRES) [206] methods, applied to our problem will be presented in Chapter 6 in connection with domain decomposed preconditioning techniques.

The CGS Algorithm:

$$r = b, \quad x = x_0, \quad r = r - Ax$$

Choose $\tilde{r}$ such that $\delta 1 = (\tilde{r}, r) \neq 0$

$$p = r, \quad u = r$$

$$p1 = Ap \tag{4.30}$$

$$\delta 0 = (\tilde{r}, p1), \quad \alpha = \delta 1/\delta 0$$

$$q = u - \alpha p1, \quad u = u + q, \quad x_s = x_s + \alpha u$$

If the convergence criterion is met then stop, otherwise continue

$$p1 = Au, \quad r = r - \alpha p1, \quad \delta 2 = (\tilde{r}, r)$$

$$\beta = \delta 2/\delta 1, \quad \delta 1 = \delta 2$$

$$u = r + \beta q, \quad p = u + \beta(q + \beta p)$$

goto (4.30)

where the right hand side b and the initial guess x_0 are input vectors and (,) denotes the usual Euclidean inner product.

It is the preconditioning which makes CGS and other iterative methods highly competitive. While we adopt an untransformed version of PCGS method in [174], here we use the transformed PCGS algorithms. Instead of solving $Ax = b$, if a preconditioner is applied on the left, we solve a transformed linear system $\hat{A}x = \hat{b}$, where $\hat{A} = B^{-1}A$ and $\hat{b} = B^{-1}b$; if a preconditioner is chosen to be applied on the right, we solve $\hat{A}\hat{x} = b$, where $\hat{A} = AB^{-1}$ and $\hat{x} = Bx$. For the left preconditioning, the transformed residual at k'th iteration is related to the original one by $\hat{r}^{(k)} = B^{-1}r^{(k)}$. For the right preconditioning, the residual remains unchanged but the final solution needs to be recovered by $x = B^{-1}\hat{x}$.

Typically, the preconditioning linear system (say $Bp = q$) is solved by the direct method. Hence, the so-called preconditioning iterative scheme may be viewed as a hybrid of iterative and direct methods. In the extreme, it becomes a "purely" iterative method ($B = I$) or, basically, a direct solver ($B = A$). In between, the matrix B is usually of a certain special structure or form which allows it to be cheaply inverted by the direct method.

4.5.3 Preconditioning in the Subdomains

In addition to the four basic types of preconditioners B mentioned in Section 4.5.2, there are other preconditioning techniques available which generally behave better, such as those based on LU or Cholesky incomplete factorizations and truncated series approximations to A^{-1} (polynomial preconditioning), where A is the matrix to be inverted. For details, we refer readers to [9, 10, 81, 134] and references cited therein. Reference [184] is especially recommended for a clear and concise overview of various acceleration methods and preconditioning techniques.

Incomplete LU (ILU) or modified incomplete LU (MILU) factorizations are among the most popular preconditioning methods which have been incorporated into the domain decomposition algorithms. The methods are based on Gaussian elimination techniques with restricted fill-ins and can be traced back to [228]. The methods are further developed in [12, 13, 41, 128, 154, 155].

Before the ILU can be started, one chooses a set of ordered integer pairs $J \subset \mathcal{J} = \{(i, j) \mid i, j = 1, 2, \ldots, n\}$ which defines the allowed fill-ins, where n is the size of the matrix under consideration. Most often, the set J is chosen to be

$$J = \{(i, j) \mid \text{if } a_{ij} \neq 0\} \tag{4.31}$$

where $A = (a_{ij})$ is the matrix to be inverted. In all our numerical experiments, we employ the set (4.31) of the index pairs, which indicates that no fill-in beyond the

non-zero pattern of the original matrix is to be allowed in the factorization. The ILU or MILU method corresponding to this choice of J is often denoted by ILU(0) or MILU(0), respectively.

Briefly speaking, upon specifying the index-pair set J and at the k'th stage of the factorization, we use the pivot row $a_{kj}^{(k)}$, $j = k, k+1, \ldots, n$ to eliminate the nonzero entries $a_{ik}^{(k)}$, $i = k+1, k+2, \ldots, n$ below the pivot $a_{kk}^{(k)}$. If $a_{ij}^{(k)} \in J$, we compute $a_{ij}^{(k+1)} = a_{ij}^{(k)} - a_{ik}^{(k)} a_{kj}^{(k)} / a_{kk}^{(k)}$, for $i, j = k+1, k+2, \ldots, n$. Otherwise, $a_{ij}^{(k)}$ is unchanged, or in the case of MILU ([10, 109, 110]), $a_{ij}^{(k)}$ is unchanged and the modification $-a_{ik}^{(k)} a_{kj}^{(k)} / a_{kk}^{(k)}$ is added back to the pivot $a_{kk}^{(k)}$. In the latter case, it may be shown that rowsum of the original matrix A is preserved, namely, $\sum_{j=1}^{n} a_{ij} = \sum_{j=1}^{n} \sum_{k=1}^{n} l_{ik} u_{kj}$.

Preconditioning methods based on the MILU factorization were proposed so that the spectral condition number of the matrix $B^{-1}A$ satisfies the following asymptotic relation (a necessary condition of the methods introduced in [109]):

$$\kappa(B^{-1}A) \sim O(h^{-1}), \quad \text{as } h \to 0 \tag{4.32}$$

where h is the mesh size, instead of $O(h^{-2})$ as for ILU. Here $B = LU = A + D + R$, R is the so-called defect or error matrix whose rowsum is zero for each row of R. D is a diagonal matrix containing some preconditioning parameters[2].

Specifically, for a class of M-matrices or a class of weakly diagonally dominant symmetric L-matrices, it was shown that the MILU factorization could be con-

[2]The analysis required to determine optimal preconditioning parameters contained in the diagonal matrix D is far from trivial even for model problems (see [110]). However, many numerical experiments [108] suggest that number of iterations is rather insensitive to the choice of these parameters. As pointed out in [110], for many problems, quite satisfactory results are obtained by choosing $D = \xi h^2 \text{diag}(A)$, for $\xi \geq 0$, say $\xi = 5$ or even $\xi = 0$ (namely $D = 0$). $D = 0$ is what we employ here.

structed such that (4.32) holds. For some other more general coefficient matrices, as pointed out in [109], the same rate of convergence was observed.

In this application of Schur domain decomposition algorithms to the finite element solution of the shallow water equations, we use MILU factorizations as our subdomain preconditioners. It is readily observed that, at each time step, the incomplete factorization has to be carried out only once for each subdomain during the entire divide-and-feedback process. The work required for the preconditioning of each subdomain then consists mainly of the forward and back substitutions which may be computed in parallel.

Another class of factorizations is the so-called relaxed incomplete LU factorization (RILU), whose algorithm will be presented in detail in Chapter 6. For this method, the rowsum criterion is only partly satisfied. However, the convergence rate is often better than that of either ILU or MILU (see [8, 11, 231]).

4.5.4 Interface Probing Preconditioners

The subdomain preconditioning is not so different from the preconditioning of a linear system corresponding to the discretization of the original PDE's on the whole domain in the sense that each of the subdomain matrices A_{ii}, $i = 1, 2, \ldots, n$ is known in advance. Nevertheless, the interface preconditioning poses a new problem. On the interfaces, a preconditioner is to be constructed for the Schur complement linear system $Cx_s = g$, where, however, the matrix C is not explicitly known.

Several interface preconditioners have been proposed for linear elliptic PDE's (see, for example, [20, 22, 23, 43, 45, 99, 214] and references therein). A critical review of some of these interface preconditioners may be found in [46]. Unfortunately, most of the existing preconditioners are constructed based on at least two of following assumptions about the differential operator, namely, elliptic, linear,

constant-coefficient, second order or self-adjoint, etc. It is thus difficult to extend the usefulness of these preconditioners to more complicated differential operators like the ones of nonlinear shallow water equations. As a result, we are interested in some general methods for deriving the interface preconditioners which are less critically dependent on the special form of the differential operator.

The interface probing ideas proposed simultaneously in [43] and [79] (see also [131, 132] and references cited therein) are a class of general and robust techniques for constructing interface preconditioners. We refer to [44] for the most recent and comprehensive review of this type of preconditioners. The method only takes advantage of the property that Schur complement matrix operator is predominantly local, reflecting the strong local coupling of the neighboring nodes and very weak global nodal coupling on the interfaces. Hence, the Schur complement dense matrix may be reasonably approximated, as an interface preconditioner, by a very low-bandwidth sparse matrix, which is obtained by evaluating the multiplications of the Schur complement matrix with a few carefully selected probing vectors ([59]). Thus, the interface probe preconditioning techniques are purely algebraic and may easily be extended for use to complicated nonlinear differential operators or to higher order (say, fourth order) linear elliptic operators. For the latter case, we refer interested readers to [40].

The observation that the entries of the Schur complement matrix C are "large" on or very close to the main diagonal of C and decay very rapidly away from the diagonal was first made in [99]. In this reference, the following estimate has been made regarding $C = (c_{ij})$ for the model problems and geometries and for the five point finite difference stencil

$$|c_{ij}| = O\left(\frac{1}{|i-j|^2}\right) \quad \text{for } i,j \text{ away from the diagonal} \tag{4.33}$$

For more complicated first and/or second order operators and geometries using, say, five point difference or seven point finite element stencil shown in Figure 4.2, one may observe empirically that C is close to being a tridiagonal matrix. This is the motivation for efficiently constructing appropriate tridiagonal or other low-bandwidth approximations G for the Schur complement matrix C, such that the spectral condition number $\kappa(G^{-1}C)$ is small.

A class of interface preconditioners $G(k)$ may be constructed by requiring that $G(k)$ be a banded matrix with upper and lower bandwidth k and have the same action as the Schur complement matrix C on a set of (probing) vectors v_j defined on the interfaces, namely,

$$G(k)v_j = Cv_j, \quad \text{for } j = 1, 2, \ldots, 2k+1 \tag{4.34}$$

If we split $G(k)$ into $G(k) = A_{ss} - P(k)$, by (4.18), the above equation (4.34) may be written as

$$P(k)v_j = A_{sd}A_{dd}^{-1}A_{ds}v_j, \quad \text{for } j = 1, 2, \ldots, 2k+1 \tag{4.35}$$

The least expensive preconditioner is obtained by taking $k = 0$. The probing vector v_1 for this case is chosen to be

$$v_1 = [1, 1, 1, 1, 1, 1, \ldots]^T$$

Three probing vectors v_1, v_2 and v_3 given below are required for the construction of the preconditioner $G(1)$

$$v_1 = [1, 0, 0, 1, 0, 0, \ldots]^T$$

$$v_2 = [0, 1, 0, 0, 1, 0, \ldots]^T$$

$$v_3 = [0, 0, 1, 0, 0, 1, \ldots]^T$$

For $k = 2$, there are five probing vectors which assume the following special forms

$$v_1 = [1,0,0,0,0,1,\ldots]^T$$

$$v_2 = [0,1,0,0,0,0,\ldots]^T$$

$$v_3 = [0,0,1,0,0,0,\ldots]^T$$

$$v_4 = [0,0,0,1,0,0,\ldots]^T$$

$$v_5 = [0,0,0,0,0,1,\ldots]^T$$

The above procedure may be extended in a straightforward manner to cases where $2 < k < n_s$. A MATLAB code for constructing the interface probing preconditioners $G(k)$ is provided in [44].

It can be readily verified that all non-zero entries in the banded matrix $P(k)$ are compressed in the $2k+1$ vectors $P(k)v_j$, $j = 1,2,\ldots,2k+1$, and can be directly read off from corresponding entries in $A_{sd}A_{dd}^{-1}A_{ds}v_j$, $j = 1,2,\ldots,2k+1$. The possible inconsistency of overdetermination ([132]) in (4.35) may reasonably be ignored if the Schur complement matrix operator C is truly predominantly local. It is also clear that $2k+1$ solves are required in each of the subdomains to construct the interface preconditioner $G(k)$. For this reason, in many practical applications, k is usually taken to be 0 or 1. In the limit, when $k = n_s - 1$, the exact Schur complement matrix C is constructed. On the other hand, when $k = 0$, the rowsum of the matrix C is preserved.

It was reported in [43] that, relative to its extra cost of formation, $G(1)$ does not yield much improvement over $G(0)$ in the context of solving a model diffusion-convection equations using a five point finite difference stencil. For solving the shallow water equations, we found, in [174], that $G(1)$ does not behave as well as $G(0)$. Hence, the rowsum preserving preconditioner $G(0)$ is to be preferred for

our problem. However, as indicated in [34], we will make some modifications to $G(0)$ and construct our interface preconditioner by retaining the cyclic tridiagonal structure of each T_{ii} in A_{ss} and then replace each entry in the main diagonal of A_{ss} by the corresponding rowsum of the Schur complement matrix C. Numerical experience indicates that the preconditioner modified in this way is better than $G(0)$ (see [34, 35] and Section 4.7).

Specifically, the preconditioner G used for accelerating CGS algorithm is the following block diagonal matrix

$$G = \text{diag}[G_{11}, G_{22}, \ldots, G_{n-1,n-1}] \tag{4.36}$$

This block diagonal matrix G is the same as A_{ss} (see (4.13) except that the main diagonal of G is modified according to the following

$$\text{diag}(G) = Cv \tag{4.37}$$

where $v = [1, 1, \ldots, 1]^T$. Obviously, only one subdomain solve is required in each subdomain to evaluate (4.37).

The preconditioning linear system $Gw_s = p$ in the PCGS algorithm may be split up into the following $n-1$ smaller systems and solved in parallel

$$G_{ii} w_s^{(i)} = p_i \tag{4.38}$$

for $i = 1, 2, \ldots, n-1$, where the definitions of $w_s^{(i)}$ and p_i are obvious.

Since each G_{ii} in (4.38) is cyclic tridiagonal (see also (4.14), we can use the so-called (see [225]) Ahlberg-Nielson-Walsh algorithm [5], which is an extension of the well-known double-sweep algorithm of Thomas [229]. For an efficient numerical algorithm for solving cyclic pentadiagonal linear systems, we recommend [171].

4.6 The Shallow Water Equations

The shallow water equations are a set of first order nonlinear symmetrizable hyperbolic partial differential equations having many important applications in meteorology and oceanography. These equations may be used in studies of tides and surface water run-off. They may also be used to study large-scale waves in the atmosphere and ocean if terms representing the effects of the earth's rotation (Coriolis terms) are included.

Here we are concerned mainly with the domain decomposition solution of the 2-D shallow water equations on a limited-area domain discretized by finite element approximations (see Appendix A). If we let $U = (\varphi, u, v)^T$ and

$$A = \begin{bmatrix} u & \varphi & 0 \\ 1 & u & 0 \\ 0 & 0 & u \end{bmatrix} \tag{4.39}$$

$$B = \begin{bmatrix} v & 0 & \varphi \\ 0 & v & 0 \\ 1 & 0 & v \end{bmatrix} \tag{4.40}$$

$$C = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -f \\ 0 & f & 0 \end{bmatrix} \tag{4.41}$$

then our shallow water equations (continuity and momentum equations) model under our consideration, in its primitive variables, may be presented compactly in the following form

$$\frac{\partial U}{\partial t} + A\frac{\partial U}{\partial x} + B\frac{\partial U}{\partial y} + CU = 0 \tag{4.42}$$

$$0 \le x \le L, \quad 0 \le y \le D, \quad t > 0$$

where the Coriolis parameter is given by the β-plane approximation

$$f = \hat{f} + \beta(y - D/2) \tag{4.43}$$

and

L and D — dimensions of a rectangular domain of area A = LD;

u and v — velocity components in the x and y directions, respectively;

h — height of the free surface of the fluid;

$\varphi = gh$ — the geopotential;

g — the acceleration of gravity;

f — the Coriolis parameter;

$\hat{f}$ and β — two constants.

The model assumes that the fluid is homogeneous, inviscid, barotropic and incompressible. These pure convection equations are defined on a limited-area rectangular domain which corresponds to a channel on the rotating earth. The southern and northern boundaries are assumed to be rigid, i.e., $v = 0$ and the flow is assumed periodic in the west-east direction. Specifically, we have

$$U(x, y, t) = U(x + L, y, t) \tag{4.44}$$

and

$$v(x, 0, t) = v(x, D, t) = 0 \tag{4.45}$$

Initial conditions also need to be specified for the φ, u and v fields, namely

$$w(x, y, 0) = w_0(x, y) \tag{4.46}$$

Under these boundary and initial conditions, the total energy

$$E = \frac{1}{2}\int_0^L \int_0^D (u^2 + v^2 + \varphi)\frac{\varphi}{g} dx dy \tag{4.47}$$

is independent of time, i.e., it is an integral invariant.

This is a standard shallow water equations model which was used to test various finite difference schemes in [102][3], in which the initial conditions recommended are determined from the following initial height field

$$\begin{aligned} h(x,y) = H_0 &+ H_1 \tanh\left(\frac{9(D/2-y)}{2D}\right) \\ &+ H_2 \mathrm{sech}^2\left(\frac{9(D/2-y)}{2D}\right)\sin\left(\frac{2\pi x}{L}\right) \end{aligned} \tag{4.48}$$

The initial geopotential φ and velocity fields u and v are derived from the initial height field using the following geostrophic relationship

$$\varphi = gh, \qquad u = -(g/f)\frac{\partial h}{\partial y}, \qquad v = (g/f)\frac{\partial h}{\partial x} \tag{4.49}$$

To scale the problem, we need to have a good control of the magnitude of the geopotential field φ, which is dimensionally the dominant variable. For this reason, we introduce a pre-selected reference geopotential φ_0 to scale the geopotential. We use the following set of dimensionless variables, which are different from those introduced in [102], to non-dimensionalize the equations and auxiliary conditions (e.g., initial conditions, geostrophic relationship, etc.)

$$x' = x/L, \qquad y' = y/L, \tag{4.50}$$

$$t' = t\sqrt{\varphi_0}/L, \qquad \varphi' = \varphi/\varphi_0, \tag{4.51}$$

$$u' = u/\sqrt{\varphi_0}, \qquad v' = v/\sqrt{\varphi_0}, \tag{4.52}$$

[3]The model is essentially the same as the one earlier used by Houghton et al. [121].

$$h' = h/L, \qquad H_0' = H_0/L, \tag{4.53}$$

$$H_1' = H_1/L, \qquad H_2' = H_2/L, \tag{4.54}$$

$$g' = gL/\varphi_0, \qquad f' = fL/\sqrt{\varphi_0}. \tag{4.55}$$

where H_0, H_1 and H_2 are three constants related to the Grammeltvedt's initial height field.

By using this set of dimensionless variables, the governing equations assume the same form after dropping the primes. However, the Coriolis parameter and the geostrophic relationship, amongst others, require minor changes (see [174]). Specifically, by using (4.50) — (4.55) and dropping the primes our problem can be shown to be governed by (4.42) — (4.46), (4.48) and (4.49) with L being 1, D replaced by D/L, $\hat{f}$ by $L\hat{f}/\sqrt{\varphi_0}$ and β by $L^2\beta/\sqrt{\varphi_0}$.

Experience has shown that the inviscid, incompressible shallow water equations model is able to describe many important aspects of atmospheric and oceanic motions. Indeed, it has become customary when developing new numerical methods for numerical weather prediction or oceanography, to study first the simpler nonlinear shallow water equations system, which possesses the same mixture of the slow-moving Rossby waves and fast-moving gravity waves as the more complex baroclinic 3-D primitive equations of motion. It should be pointed out, however, that the model may not be applied to situations in which the density-stratification effect can not be ignored.

We now briefly describe the finite element discretization of the shallow water equations given in (4.42). Interested readers are advised to read Appendix A for details.

Upon using the Galerkin finite element discretization procedure with triangular piecewise linear elements, the continuous shallow water equations (4.42) are trans-

formed into the following three sets of matrix equations

$$M\dot{\varphi} - K_1\varphi = 0 \tag{4.56}$$

$$M\dot{u} + K_2 u + K_3\varphi - K_4 v = 0 \tag{4.57}$$

$$M\dot{v} + K_2 v + K_5\varphi + K_4 u = 0 \tag{4.58}$$

where φ, u and v are nodal unknown vectors and M, K_1, ..., K_5 are matrices defined by integrals of the element shape functions or their spatial derivatives and the nodal unknowns u and v.

An extrapolated Crank-Nicolson scheme is employed for the time discretization, in which the nonlinear advective terms are quasilinearized by the following second order approximation in time

$$u^{n+1/2} = \frac{3}{2}u^n - \frac{1}{2}u^{n-1} \tag{4.59}$$

$$v^{n+1/2} = \frac{3}{2}v^n - \frac{1}{2}v^{n-1}. \tag{4.60}$$

After collecting and rearranging terms, we obtain the following three linear systems which need to be inverted at each time step

$$A^n\Delta\varphi^n = f_\varphi^n, \quad B^n\Delta u^n = f_u^n, \quad C^n\Delta v^n = f_v^n \tag{4.61}$$

where A^n, B^n and C^n are nonsymmetric matrices due to the presence of advective terms and $\Delta\varphi^n = \varphi^{n+1} - \varphi^n$, $\Delta u^n = u^{n+1} - u^n$, $\Delta v^n = v^{n+1} - v^n$. Three right hand side terms in (4.61) are functions of nodal unknowns at the previous steps.

It is worth pointing out that the above discretization procedure transforms the originally coupled PDE's into a set of decoupled discrete algebraic equations at the $(n+1)$-th level, which corresponds to reduced storage requirements and improved computational efficiency compared to methods which generate coupled algebraic

equations. The explicit appearance of $\Delta\varphi$, Δu and Δv instead of φ, u and v minimizes the build-up of round-off errors in the computation (see [87]) and offers direct indications as to the choice of initial guesses for iterative methods.

4.7 Numerical Results and Discussions

We carried out numerical experiments on the CRAY Y-MP/432 which has a machine accuracy around 0.8×10^{-14} for a single precision arithmetic. For our numerical experiments, we used the following constants whose numerical values are summarized below (see Section 4.6).

$$L = 6000\ km, \qquad D = 4400\ km$$

$$g = 10\ m/s, \qquad H_0 = 2000\ m$$

$$H_1 = 220\ m, \qquad H_2 = 133\ m$$

$$\hat{f} = 10^{-4}\ s^{-1}, \qquad \beta = 1.5 \times 10^{-11}\ s^{-1}m^{-1}$$

The geopotential field distribution in the present problem has the order of magnitude $10^1\ m^2/s^2$. The pre-selected reference geopotential has been chosen to be $\varphi_0 = 10^2\ m^2/s^2$ so that the initial residual norm of the Schur complement linear system corresponding to the non-dimensionalized geopotential φ' as defined by (4.51) has the order of magnitude of $O(1)$. Under this choice of reference geopotential the dimensionless constants become

$$L' = 1 \qquad D' = 0.733333$$

$$g' = 6000 \qquad H_0' = 0.333333 \times 10^{-3}$$

$$H_1' = 0.366666 \times 10^{-4} \qquad H_2' = 0.221666 \times 10^{-4}$$

$$\hat{f}' = 6 \qquad \beta' = 5.4$$

The initial non-dimensionalized geopotential distribution for the 2-D shallow water equations under consideration is shown in Figure 4.6.

In order to allow for further flexibility of the code, a modification has been introduced which allows, by changing just two parameters corresponding to the number of grid points in the x and y directions respectively, the introduction of arbitrary mesh resolutions of the model. The automatic transformation of the global finite element matrices to the block-bordered (4.5) forms has been rendered possible by the node renumbering scheme presented in Section 4.3.

In the actual implementation, the matrices are seldom stored in full due to their sparsity property. Hence the general node renumbering scheme in Section 4.3 has been adapted for transforming the information corresponding to compact matrices (see Section A.3 in Appendix A). Here we essentially transform the arrays which record the nonzero elements of the sparse matrices. The solutions of shallow water equations using the Schur domain decomposition algorithm (Algorithm 4.2) are the same as those presented in Section A.8 of Appendix A.

In the following, unless otherwise stated, we present numerical results corresponding to a four-subdomain domain decomposition. Computations are carried out corresponding to various spatial mesh resolutions with a time-step size $\Delta t = 1800$ s.

As pointed out in Section 4.5.4, empirically, the matrix C is often close to being a tridiagonal matrix, reflecting a strong coupling between neighboring nodes and weak global dependencies on the interfaces. To justify our choosing to apply the interface probing preconditioners to the iterative solution of the Schur complement linear systems defined on the interfaces, we present in Figure 4.7 the surface generated by the entries in the non-dimensionalized geopotential Schur complement matrix as a

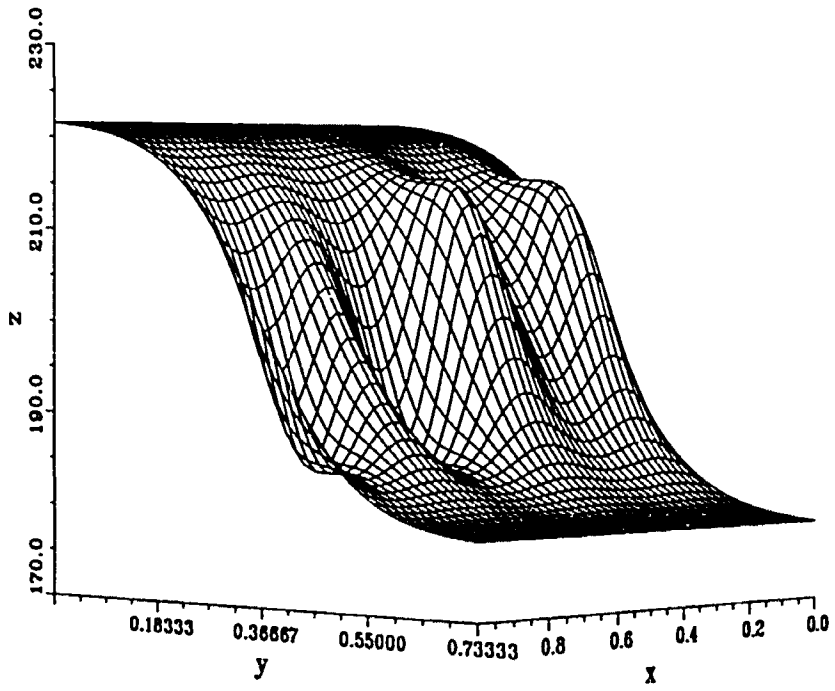


Figure 4.6: A 3-D view of Grammeltvedt's initial non-dimensionalized geopotential field.

function of its two indices at the end of one hour with a half an hour time step. The mesh resolution used for this calculation is 15×15. Hence, the number of nodes on the interfaces is $n_s = 45$ and the size of the Schur complement matrix C is 45×45.

We observe that the matrix structure of C resembles that of A_{ss} and may be viewed, ignoring small entries, as "truly" block cyclic tridiagonal. As a matter of fact, a plot of the surface formed by the entries of the matrix A_{ss} can not be distinguished from that formed by the entries of the matrix C on the scale shown in Figure 4.7.

By (4.18), the contributions to the matrix C come from two parts: A_{ss}, i.e., the discrete counterpart of the original differential operator restricted to the interfaces and those due to the coupling between the interfaces and the subdomains through bi-diagonal matrices A_{ds} and A_{sd} (see (4.7) and (4.8)). As can be expected, the former constitutes the major contribution. The probing interface preconditioning is essentially concerned with how to appropriately and efficiently incorporate the latter contribution into the matrix A_{ss}, so that the spectral condition number $\kappa(G^{-1}C)$ can be rendered as small as possible, without investing much CPU time.

To appreciate how the matrix A_{ss} dominates, we plot, in Figure 4.8, the surface formed by the entries of the matrix $A_{ss} - C = A_{sd}A_{dd}^{-1}A_{ds}$. We notice that, in general, entries of the matrix A_{ss} overestimate those of the Schur complement matrix C. To facilitate the comparison, Figure 4.8 is drawn on the same coordinate systems as Figure 4.7.

We plot, in Figure 4.9 and 4.10, the entries of the sixteenth row (associated with a boundary node) and the twenty fourth row (associated with an interior grid point) of the following four matrices, namely, A_{ss}, C, the rowsum preserving preconditioner $G(0)$ and its modification G (see (4.36)). Since node 16 is on the computational boundary (see Figure A.2), the periodic boundary condition (4.44) determines that

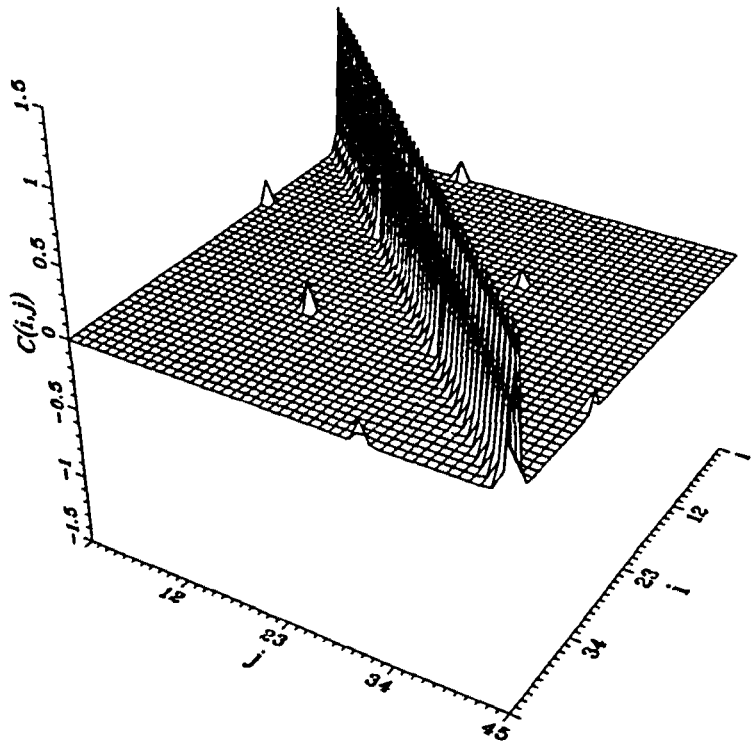


Figure 4.7: The surface generated by the non-dimensionalized geopotential Schur complement matrix C at the end of one hour. The mesh resolution is 15×15 in the original physical domain and the number of nodes on the interfaces is $n_s = 45$.

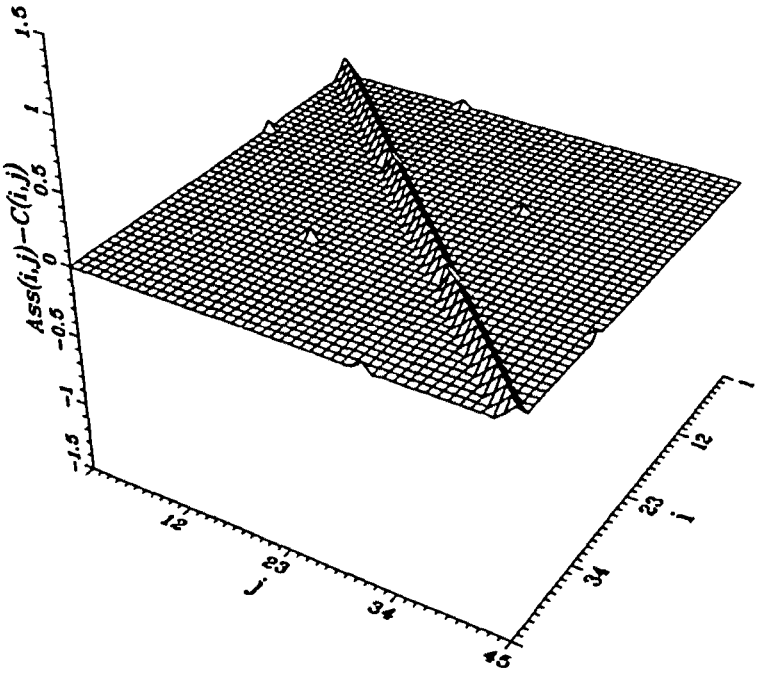


Figure 4.8: The surface generated by $A_{ss} - C' = A_{sd}A_{dd}^{-1}A_{ds}$ corresponding the geopotential at the end of one hour of model integration. The mesh resolution is 15×15 in the original physical domain and the number of nodes on the interfaces is $n_s = 45$.

node 16 will interact not only with its neighboring node (node 17), but also with node 30. This is clearly seen in Figure 4.9, where the entry in column 30 of the matrix does not vanish. In contrast, $A_{ss}(24,30) = G(0)(24,30) = G(24,30) = 0$ and $C(24,30)$ is negligibly small, or precisely, of order $O(10^{-6})$ which is, of course, indistinguishable on the scale of Figure 4.10.

Based on the information provided by Figures 4.7, 4.9(b) and 4.10(b), we conclude that there are only three entries on each row of the Schur complement matrix C which are relatively "large" and all other remaining entries are comparatively very small. However, due to the periodic boundary condition imposed on our problem, C may not be viewed as a tridiagonal matrix. We also notice that, as an approximation of the matrix C, the rowsum preserving preconditioner $G(0)$ underestimates C on the main diagonal but overestimates C on those off-diagonal "large" entries. On the other hand, as a modification of $G(0)$, the preconditioner G consistently overestimates C on all "large" entries.

For higher mesh resolutions, the conclusions just made above remain true. For instance, the Schur complement matrix C inherits the block cyclic tridiagonal structure of the matrix A_{ss}. In Figure 4.11, we show that the matrix structure of C corresponding to a mesh resolution of 55×51. In this case, there are 165 nodes on the interfaces ($n_s = 165$) and hence the size of the matrix C is 165×165. It was also found that structures of the Schur complement matrices governing the velocity distributions on the interfaces are all similar to those displayed in Figures 4.7 and 4.11, respectively.

To show that, in general, the modified preconditioner G behaves better than the rowsum preserving preconditioner $G(0)$ in our application, we employ an IMSL library routine ([125]) to estimate the condition numbers of the following three preconditioned Schur complement matrices, namely, $\kappa(A_{ss}^{-1}C) = \|A_{ss}^{-1}C\|_1\|C^{-1}A_{ss}\|_1$,

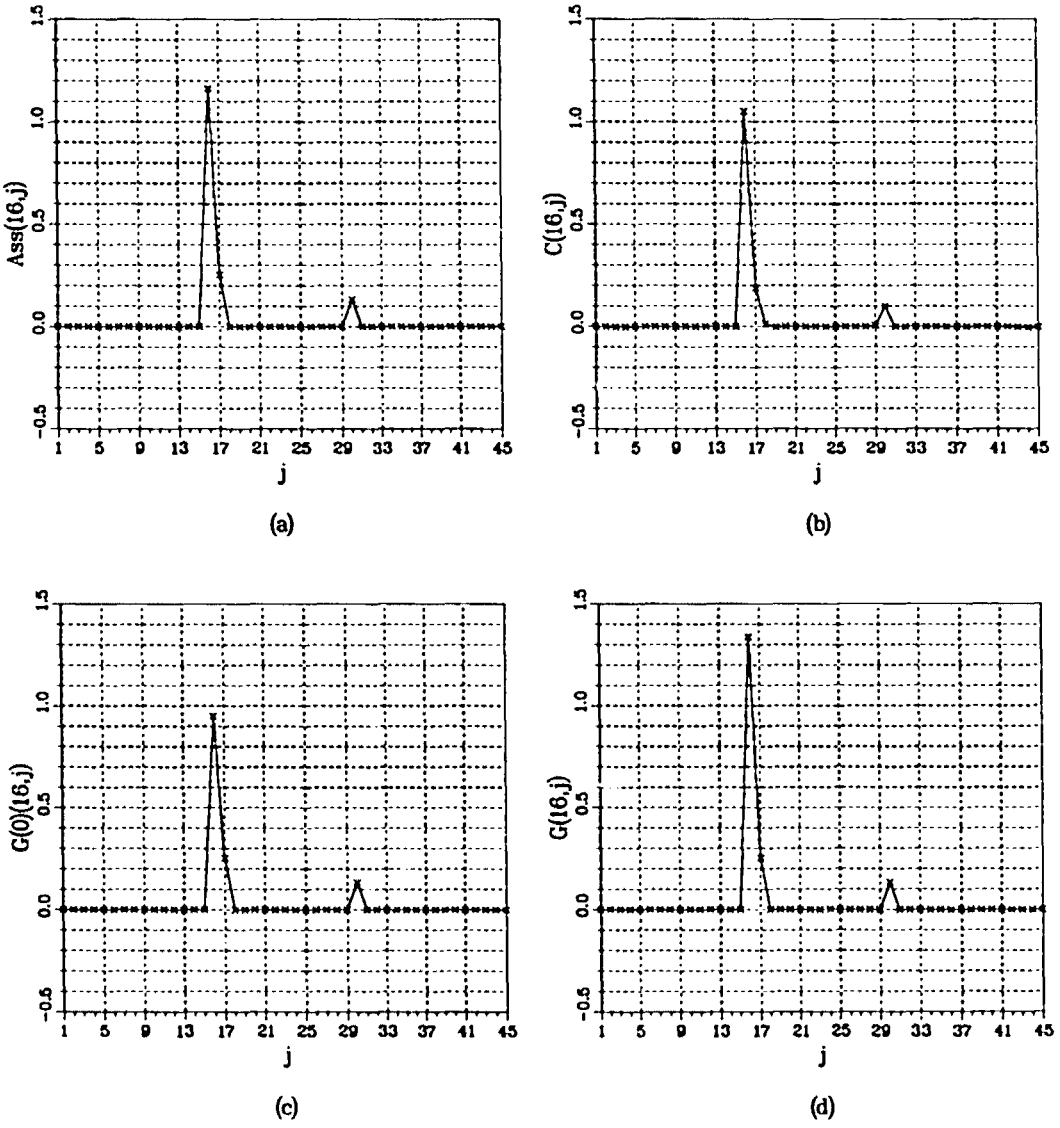


Figure 4.9: Entries of the sixteenth row of the matrices (a) A_{ss}; (b) C; (c) G(0) and (d) G. The number of nodes on the interfaces is $n_s = 45$.

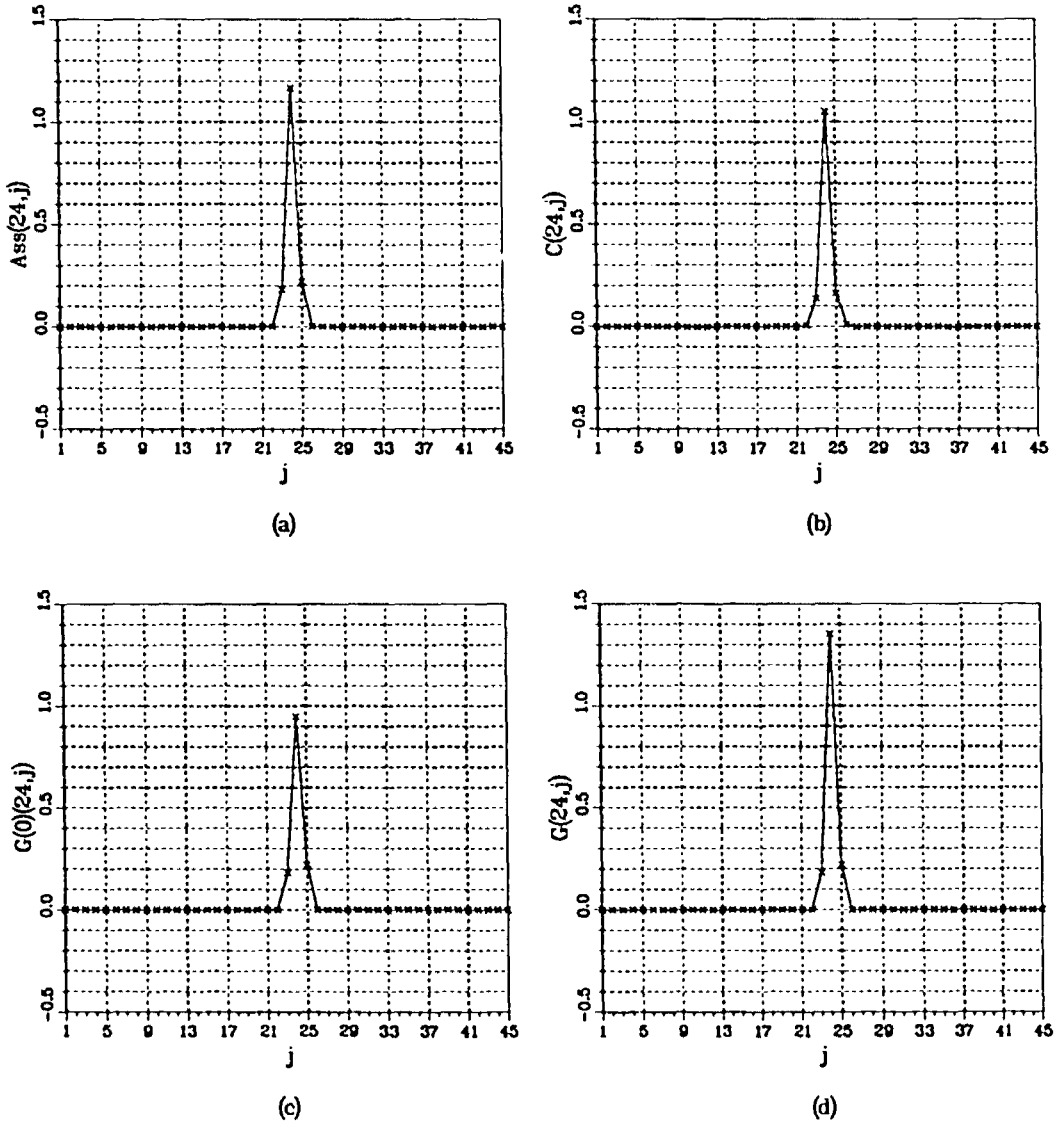


Figure 4.10: Entries of the twenty fourth row of the matrices (a) A_{ss}; (b) C; (c) G(0) and (d) G. The number of nodes on the interfaces is $n_s = 45$.

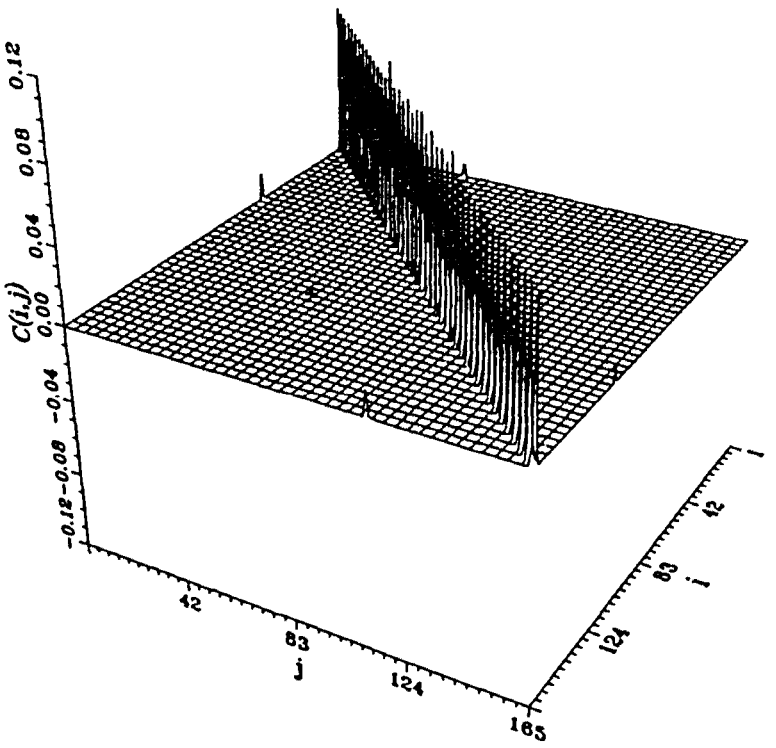


Figure 4.11: The surface generated by the non-dimensionalized geopotential Schur complement matrix C at the end of one hour of model integration. The mesh resolution is 55×51 in the original physical domain and the number of nodes on the interfaces is $n_s = 165$. Note that the mesh lines have been thinned by a factor of four in both directions along i and j.

$\kappa(G(0)^{-1}C) = \|G(0)^{-1}C\|_1\|C^{-1}G(0)\|_1$ and $\kappa(G^{-1}C) = \|G^{-1}C\|_1\|C^{-1}G\|_1$. The results are reported in Table 4.1.

Table 4.1: Condition numbers associated with the 1-norm for three preconditioned Schur complement matrices $A_{ss}^{-1}C$, $G(0)^{-1}C$ and $G^{-1}C$

n_s	$\kappa(A_{ss}^{-1}C)$	$\kappa(G(0)^{-1}C)$	$\kappa(G^{-1}C)$
45	1.18	1.42	1.11
155	1.50	1.47	1.19

For all subsequent convergence tests of the preconditioning iterative method, the 2-norm or Euclidean residual norm will be employed throughout the rest of this section. The stopping criterion is based on the 2-norm of the final residual vector being smaller than 0.1×10^{-10}.

Before confirming numerically that the interface probing preconditioner G is computationally more efficient than $G(0)$ (see Section 4.5.4), illustrate the effect of MILU preconditioning on the iterative solution in the subdomains. Typical convergence behaviors are depicted in Figures 4.12(a) and (b) for two mesh resolutions, namely, 60×55 and 120×115. Timing results are given in Table 4.2. We observe that the number of iterations required for MILU preconditioned CGS algorithm increases by a factor of two when moving from the coarser mesh (case (a)) to the finer mesh (case(b)). This may be explained by the asymptotic relation (4.32) given in Section 4.5.3 since $h_a \approx 2h_b$, where h_a and h_b are the mesh sizes for the smaller and larger mesh resolutions, respectively. However, as we note, a general convergence theory on PCGS method is not available. It is also worth noting that the above result is for MILU preconditioners applied from the left. Right MILU preconditioning is found to be not as good for this particular situation.

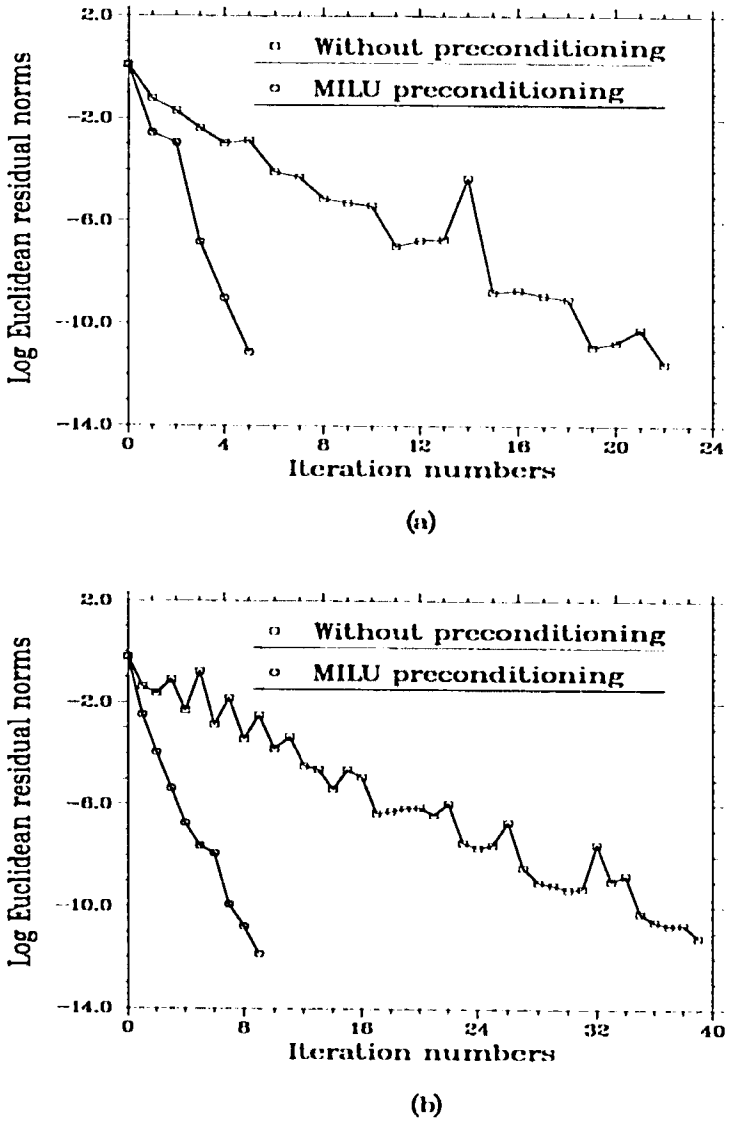


Figure 4.12: The evolution of log_{10} Euclidean residual norms as a function of number of iterations in the subdomain for the non-dimensionalized geopotential matrix system at the end of one hour with and without MILU preconditioning. The mesh resolution is (a) 60×55 and (b) 120×115. There are 780 and 3360 nodes, for cases (a) and (b), respectively, in each of the four subdomains.

Table 4.2: MILU preconditioning in a typical subdomain

Resolutions	No preconditioning	MILU preconditioning
60×55	0.12s (22 iterations)	0.057s (5 iterations)
120×115	0.89s (39 iterations)	0.42s (9 iterations)

Now we turn to numerical tests on various interface preconditioners. Specifically, the following four cases will be considered (see Section 4.5.4):

- SDD I — without any interface preconditioners.
- SDD II — the preconditioner simply taken to be the matrix A_{ss} explained in Section 4.2 (page 70);
- SDD III — the interface probing preconditioner $G(0)$;
- SDD IV — the interface probing preconditioner G;

We present, in Figure 4.13 — 4.16, the log_{10} norms of the residual vectors on the interfaces versus the number of iterations corresponding to each of these cases listed above and for four different mesh resolutions, namely, 30×27, 60×55, 90×83 and finally 120×115. Both the number of iterations and CPU time consumed for each case are recorded in Table 4.3 for the solution of the non-dimensionalized geopotential Schur complement linear system at the end of one hour.

The suitability and efficiency of the interface probing preconditioner G, which is introduced in Section 4.5.4 (page 94) and is based on a modification of the rowsum preserving preconditioner $G(0)$ (see page 92), have been numerically confirmed. We may observe that, as the mesh resolution increases, A_{ss}, as an interface preconditioner, displays poor adaptivity. However, both preconditioners $G(0)$ and G possess

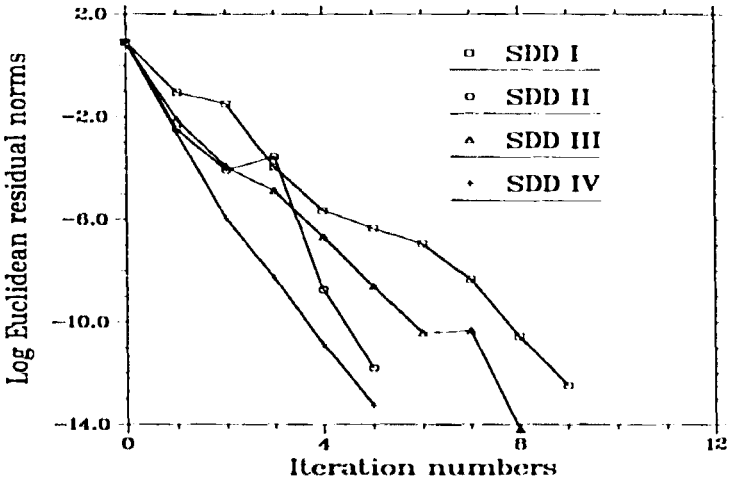


Figure 4.13: The evolution of log_{10} Euclidean residual norms as a function of number of iterations for the Schur complement matrix linear system on the interfaces for the non-dimensionalized geopotential matrix system at the end of one hour of model integration. The mesh resolution is 30×27. For this choice, there are 90 nodes on the interfaces.

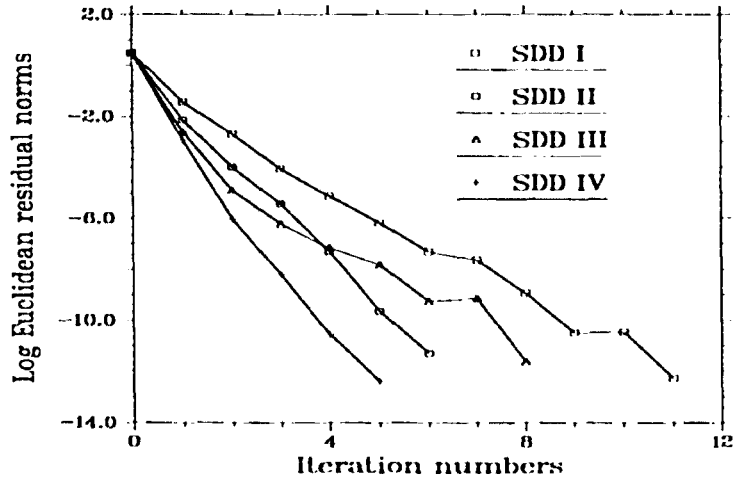


Figure 4.14: The evolution of log_{10} Euclidean residual norms as a function of number of iterations for the Schur complement matrix linear system on the interfaces for the non-dimensionalized geopotential matrix system at the end of one hour of model integration. The mesh resolution is 60×55. For this choice, there are 180 nodes on the interfaces.

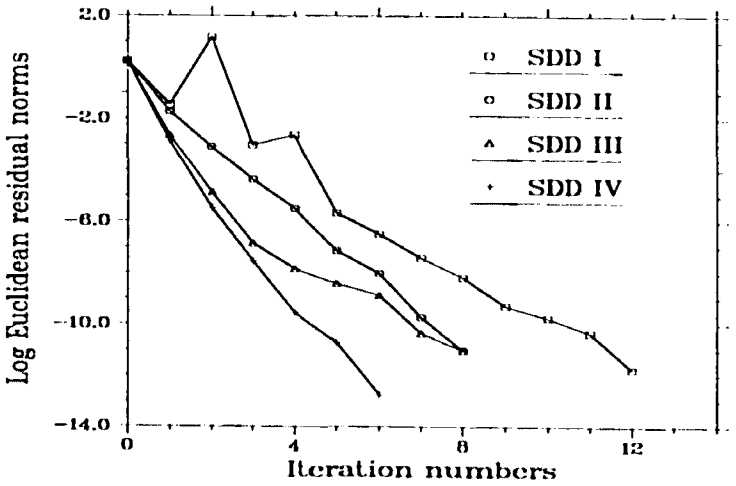


Figure 4.15: The evolution of log_{10} Euclidean residual norms as a function of number of iterations for the Schur complement matrix linear system on the interfaces for the non-dimensionalized geopotential matrix system at the end of one hour of model integration. The mesh resolution is 90×83. For this choice, there are 270 nodes on the interfaces.

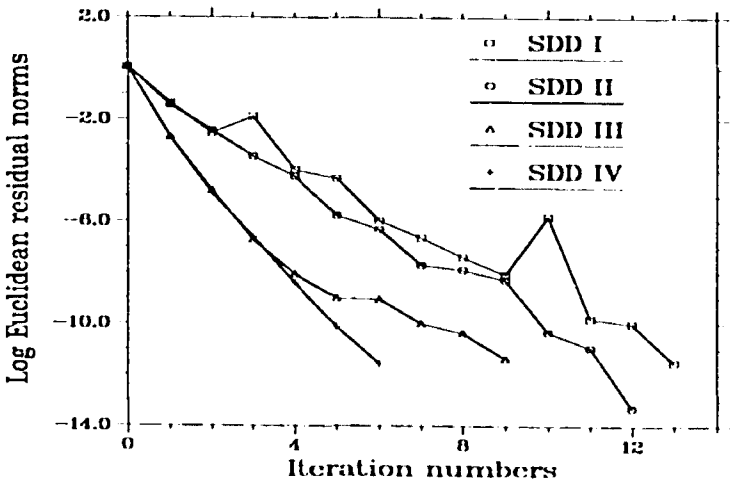


Figure 4.16: The evolution of log_{10} Euclidean residual norms as a function of number of iterations for the Schur complement matrix linear system on the interfaces for the non-dimensionalized geopotential matrix system at the end of one hour of model integration. The mesh resolution is 120×115. For this choice, there are 360 nodes on the interfaces.

Table 4.3: A comparison of CPU time in seconds (number of iterations) for the iterative solution of the Schur complement linear systems on the interfaces

Mesh resolutions	30×27	60×55	90×83	120×115
SDD I	0.68 (9)	3.33 (11)	9.95 (12)	25.10 (13)
SDD II	0.44 (5)	2.31 (6)	7.72 (8)	23.74 (12)
SDD III	0.57 (8)	2.52 (8)	6.63 (8)	14.70 (9)
SDD IV	0.38 (5)	1.70 (5)	5.10 (6)	12.12 (6)

much better adaptivities to mesh refinement. We also notice that, for the case of 30×27 mesh resolution, SDD IV is computationally cheaper than SDD II although they both require the same number of outer iterations. This is due to the fact that, using the former preconditioner, the convergence rate of the iterative solution in each of the subdomains tends to be faster, namely, comparatively fewer inner iterations are required.

4.8 Conclusions

- The Schur domain decomposition method provides a "divide, conquer and combine" algorithmic structure for mapping a whole computational effort onto a number of processors for the parallel numerical solution of PDE's, using either the finite difference or the finite element discretization. The resulting domain decomposition algorithm is easy to implement. However, this method may not be cost effective in the absence of fast subdomain solvers.

- The Schur complement matrix C is so dense that Algorithm 4.1, although an improvement of it has been made here over the traditional one, is not recommended in general, except for special cases (for example, discrete linear

systems with an identical coefficient matrix at each time step in a time dependent problem or Newton's method for solving a nonlinear system of algebraic equations by freezing the Jacobian matrix for a number of steps [87, p. 166]). In other words, the generally preferred Schur domain decomposition approach should be based on Algorithm 4.2 which represents a "divide and feedback" iterative procedure illustrated in Figure 4.5.

- The node renumbering scheme proposed in this chapter not only facilitates the modification of an existing code into a non-overlapping domain decomposition code, but also provides an easy and alternative way for implementing classical multicoloring techniques.

- The efficiency of this Schur approach strongly depends on the number of outer iterations, i.e., the number of iterations required for a Krylov or conjugate gradient-like iterative solver for linear systems on the interfaces to satisfy a prescribed convergence criterion. The predominantly local characteristic of the Schur complement matrix operator justifies the use of interface probing preconditioning ideas for accelerating the convergence of the iterative solution of the Schur complement linear system. However, the most often used rowsum preserving preconditioner $G(0)$, although well-behaved for most elliptic PDE's, does not carry over to the current application and requires a modification as proposed in Section 4.5.4. The application of this modified rowsum preserving preconditioner G results in a much better convergence behavior, especially for finer mesh resolution cases.

CHAPTER 5

THE MODIFIED INTERFACE MATRIX DOMAIN DECOMPOSITION ALGORITHMS AND APPLICATIONS

5.1 Introduction and Motivation

In Chapter 4, we have seen that, as a "divide and feedback" process, the iterative Schur domain decomposition method provides a good algorithmic structure for mapping the computational work involved in the numerical solution of PDE's onto multi-processor computing systems for parallel processing. The computational cost of this approach is determined by two factors, namely, the number of iterations required of an iterative algorithm to solve the Schur complement linear system on the interfaces and the computational costs of the subdomain solvers. One of the drawbacks of the method is that subdomain solutions are usually carried out exactly. For problems where fast direct subdomain solvers are locally exploitable (see, for example, Section 3.2 of Chapter 3), the Schur domain decomposition method with an appropriate interface preconditioner provides an efficient and cost-effective algorithm. Unfortunately, fast subdomain solvers are not available in most application problems. For such cases, the Schur domain decomposition method may not be efficient when considered from the computational complexity point of view.

Domain decomposition ideas based on the modified interface matrix (MIM) approach [34, 174] are proposed to reduce the cost for obtaining subdomain solutions. For this novel approach, the subdomain solutions are still carried out exactly, how-

ever, with the improved intial solutions, they can be obtained faster and less expensively as the iterative procedure continues. Thus, the disadvantage arising from the absence of fast subdomain solvers is mitigated. The ideas proposed for improving the initial solutions in the design of iterative solvers for time-dependent and boundary value problems are not new and have been pointed out, for example, in [10, p. 385]. These ideas will be further studied and implemented here in connection with domain decomposition strategies.

5.2 The Modified Interface Matrix Domain Decomposition Method

5.2.1 The Basic Theory

For the Schur domain decomposition method presented in the previous chapter, the subdomain problems (4.17) are solved only after the solution x_s on the interfaces is obtained. We propose, in this section, a new approach to handle coupling between different subdomains. This new approach is largely based on the following two theorems (Theorem 5.1 and 5.2) and differs from the Schur domain decomposition method in that the approximations to the solutions on the interfaces and in the subdomains are successively improved.

Theorem 5.1 Sequences of approximation $x_i^{(k)}$, for $i = 1, 2, \ldots, n$, and $x_s^{(k)}$ produced by

$$\text{For } k = 0, 1, 2, \ldots$$

$$A_{ii}x_i^{(k+1)} = f_i - A_{is}x_s^{(k)} \qquad \text{for } i = 1, 2, \ldots, n \tag{5.1}$$

$$A_{ss}x_s^{(k+1)} = f_s - \sum_{i=1}^{n} A_{si}x_i^{(k+1)} \tag{5.2}$$

converge to x_i^*, for $i = 1, 2, \ldots, n$, and x_s^* for an arbitrary initial solution $x_s^{(0)}$ on the interfaces if and only if the spectral radius of the matrix $I_{ss} - A_{ss}^{-1}C$ satisfies

$$\rho(I_{ss} - A_{ss}^{-1}C) < 1 \tag{5.3}$$

where I_{ss} is an identity matrix of size $n_s \times n_s$, x_s^* and x_i^*, for $i = 1, 2, \ldots, n$, are the solutions of (4.16) and (4.17).

PROOF. Define an error vector $e_s^{(k)} = x_s^{(k)} - x_s^*$ for the interface approximation. Since $x_i^{(k+1)} = A_{ii}^{-1}(f_i - A_{is}x_s^{(k)})$, for $i = 1, 2, \ldots, n$, it is then straightforward to show that $A_{ss}x_s^{(k+1)} = g + (A_{ss} - C)x_s^{(k)}$, where g is defined by (4.21). It follows that $e_s^{(k+1)} = (I_{ss} - A_{ss}^{-1}C)e_s^{(k)}$. Hence we have $x_s^{(k)} \to x_s^*$, as $k \to \infty$ if and only if (5.3) holds. Now let $k \to \infty$ in (5.1), it follows from (4.17) that $x_i^{(k)} \to x_i^*$, as $k \to \infty$, for $i = 1, 2, \ldots, n$. Q.E.D.

By theorem 5.1, we obtain the following iterative procedure for solving the linear system $Ax = f$. The iteration starts with an arbitrary initial solution $x_s^{(0)}$ on the interfaces $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \cdots \cup \Gamma_{n-1}$, solves the subdomain problems (5.1) in parallel, then updates the approximation on the interfaces by solving (5.2). In this way we solve the subdomain problems and the interface problem successively until the convergence criterion on the interfaces is satisfied.

Compared to the Schur domain decomposition method, the algorithm based on theorem 5.1 is quite straightforward and the condition (5.3) is satisfied in our applications for various mesh resolutions. Specifically, our numerical experiments indicate that for various mesh resolutions of the discretized domain, the spectral radii $\rho(I_{ss} - A_{ss}^{-1}C)$ are about 0.13. Hence the asymptotic rate of convergence $-\ln \rho$, (see, for example, [113]) is about 2.0102. Thus, in order to reduce the norm of the initial error vector on the interfaces by a factor of, say, 10^{-6}, roughly seven iterations are required.

We now proceed to modify this iterative procedure. Since each iteration using (5.1) and (5.2) requires the solutions of all the subdomain problems once, it is important to reduce the number of iterations. It is well known that the smaller the

spectral radius, the faster the rate of convergence. In order to reduce the spectral radius, we construct a matrix K_{ss} such that $A_{ss}+K_{ss}$ is a good approximation of the Schur complement matrix C in the sense that the spectral radius $\rho[(A_{ss}+K_{ss})^{-1}C]$ is close to 1. The matrix $A_{ss}+K_{ss}$ is referred to as the modified interface matrix.

Now assume that the matrix K_{ss} has been chosen, the following result may be proved in a similar way:

Theorem 5.2 Sequences of approximation $x_i^{(k)}$, for $i = 1,2,\ldots,n$, and $x_s^{(k)}$ obtained by

$$\text{For } k = 0,1,2,\ldots$$

$$A_{ii}x_i^{(k+1)} = f_i - A_{is}x_s^{(k)} \qquad \text{for } i = 1,2,\ldots,n \tag{5.4}$$

$$(A_{ss}+K_{ss})\Delta x_s^{(k)} = f_s - A_{ss}x_s^{(k)} - \sum_{i=1}^{n} A_{si}x_i^{(k+1)} \tag{5.5}$$

converge to x_i^*, for $i = 1,2,\ldots,n$, and x_s^* for an arbitrary initial solution $x_s^{(0)}$ on the interfaces if and only if the spectral radius of the matrix $I_{ss}-(A_{ss}+K_{ss})^{-1}C$ satisfies

$$\rho(I_{ss}-(A_{ss}+K_{ss})^{-1}C) < 1 \tag{5.6}$$

where $\Delta x_s^{(k)} = x_s^{(k+1)} - x_s^{(k)}$, I_{ss} is an identity matrix of size $n_s \times n_s$, x_s^* and x_i^*, for $i = 1,2,\ldots,n$, are the solutions of (4.16) and (4.17). Moreover, the following norm relations hold

$$\|(A_{ss}+K_{ss})\Delta x_s^{(k)}\| = \|g - Cx_s^{(k)}\| \tag{5.7}$$

where C is the Schur complement matrix and g is the corresponding right hand side, defined by (4.21).

Based on theorem 5.2, the iteration starts with an arbitrary initial solution $x_s^{(0)}$ on the interfaces and we successively solve the subdomain problems (5.4) in parallel and

the linear system (5.5) on the interfaces until the norm given by (5.7) is sufficiently reduced. A highly simplified control flow (data dependency) graph is presented in Figure 5.1. Notice that the computational work involved in $n-1$ interfaces may also be carried out in parallel with smaller granularity. This fact is hidden in the figure.

It is interesting to make an analogy here and think of each of the equations in (5.4) as the governing equation for the displacement distribution within a substructure subjected to a load f_i originally acting upon it, and the boundary interaction forces $-A_{is}x_s^{(k)}$ due to interface connections between the substructures.

By taking $x_s^{(0)}$ to be some initial solution on the interfaces that differs from the true solution x_s^*, we are actually imposing some constraints in addition to the original constraints (the boundary conditions) to the structure and thus making it stiffer. However, as guaranteed by theorem 5.2, the extra constraints introduced due to the incorrect initial solution will be continually relaxed by solving the linear system (5.5) repeatedly.

A Krylov or conjugate-gradient like algorithm is applied to both (5.4) and (5.5), accelerated by the MILU preconditioner for the former and a modified version of the rowsum preserving interface preconditioner (see Section 5.2.3) for the latter. The stopping criterion for the iterative procedure introduced in Theorem 5.2 is based on the final Euclidean residual norm $\|g - Cx_s^{(k)}\|$ being smaller than a pre-defined small number, say, 0.1×10^{-10}, and this residual norm information is conveniently and less expensively provided by $\|(A_{ss} + K_{ss})\Delta x_s^{(k)}\|$ (see (5.7)).

5.2.2 The Construction of K_{ss}

We now investigate the construction of the matrix K_{ss} designed so that the modified interface matrix $A_{ss} + K_{ss}$ constitutes a good approximation of the Schur

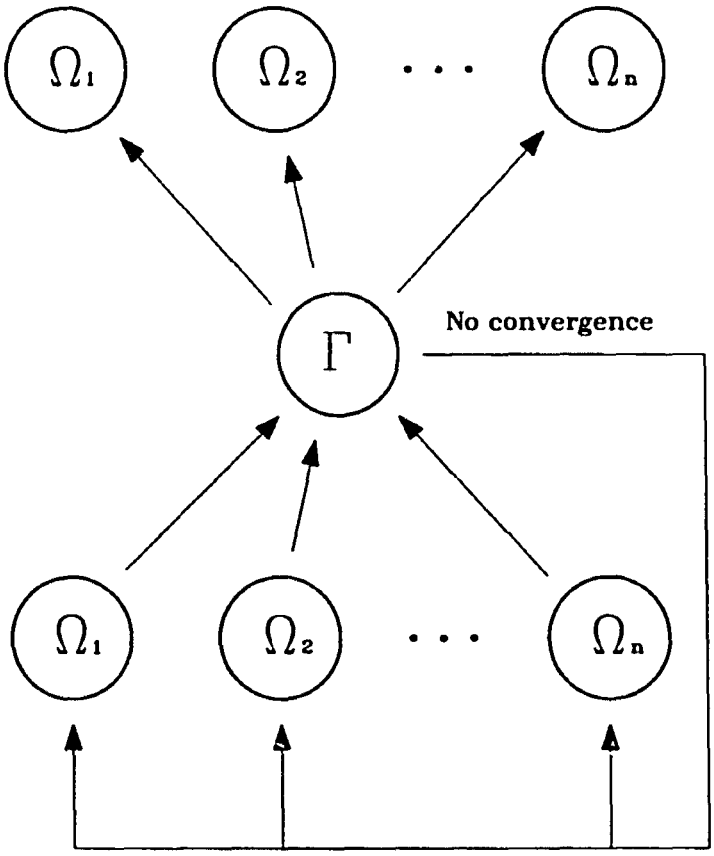


Figure 5.1: A highly simplified control flow (data dependency) graph for the modified interface matrix domain decomposition algorithm.

complement matrix C. Since we are mainly interested in non-symmetric iterative methods for which only matrix-vector products are required for the solution of the system (5.5), it is more efficient to consider the algorithm of computing $K_{ss}w_s$ for a given vector w_s rather than first forming the matrix and then the product.

As a first approach, let us consider a splitting of each of the subdomain matrices A_{ii}

$$A_{ii} = P_{ii} - Q_{ii} = P_{ii}(I_{ii} - R_{ii}) \tag{5.8}$$

where P_{ii} is a nonsingular matrix and I_{ii} is the identity matrix of size $n_i \times n_i$. Assuming that the spectral radius of $R_{ii} = I_{ii} - P_{ii}^{-1}A_{ii}$ is less than 1, we may obtain the following Neumann series expansion

$$A_{ii}^{-1} = [\sum_{k=0}^{\infty}(I_{ii} - P_{ii}^{-1}A_{ii})^k]P_{ii}^{-1}. \tag{5.9}$$

$C - A_{ss}$ can be approximated by using only a finite truncated expansion, i.e.,

$$C - A_{ss} \approx K_{ss} = -\sum_{i=1}^{n} A_{si}X_{is}^{(m+1)} \tag{5.10}$$

where

$$X_{is}^{(m+1)} = [\sum_{k=0}^{m}(I_{ii} - P_{ii}^{-1}A_{ii})^k]P_{ii}^{-1}A_{is}. \tag{5.11}$$

To derive an iterative method for computing $K_{ss}w_s = -\sum_{i=1}^{n} A_{si}X_{is}^{(m+1)}w_s$, we consider solving each of the following linear systems of the form $A_{ii}v_i = u_i$, for $i = 1, 2, \ldots, n$, by a linear and stationary iterative scheme

$$v_i^{(k+1)} = v_i^{(k)} + P_{ii}^{-1}(u_i - A_{ii}v_i^{(k)}) \tag{5.12}$$

$$= (I_{ii} - P_{ii}^{-1}A_{ii})v_i^{(k)} + P_{ii}^{-1}u_i \tag{5.13}$$

From (5.13), we obtain

$$v_i^{(1)} = (I_{ii} - P_{ii}^{-1}A_{ii})v_i^{(0)} + P_{ii}^{-1}u_i \tag{5.14}$$

$$v_i^{(2)} = (I_{ii} - P_{ii}^{-1}A_{ii})^2 v_i^{(0)} + (I_{ii} - P_{ii}^{-1}A_{ii})P_{ii}^{-1}u_i + P_{ii}^{-1}u_i \tag{5.15}$$

$$\begin{aligned} v_i^{(3)} &= (I_{ii} - P_{ii}^{-1}A_{ii})^3 v_i^{(0)} + (I_{ii} - P_{ii}^{-1}A_{ii})^2 P_{ii}^{-1}u_i \\ &\quad + (I_{ii} - P_{ii}^{-1}A_{ii})P_{ii}^{-1}u_i + P_{ii}^{-1}u_i \end{aligned} \tag{5.16}$$

and in general

$$v_i^{(m+1)} = (I_{ii} - P_{ii}^{-1}A_{ii})^{m+1} v_i^{(0)} + \sum_{k=0}^{m} (I_{ii} - P_{ii}^{-1}A_{ii})^k P_{ii}^{-1}u_i \tag{5.17}$$

Based on (5.13) and (5.17), the following iterative algorithm for computing $K_{ss}w_s$ is obtained

$$K_{ss}w_s = -\sum_{i=1}^{n} A_{si}v_i^{(m+1)} \tag{5.18}$$

where the vector $v_i^{(m+1)}$ can be obtained by the following iterative procedure: for $k = 0, 1, \ldots, m$, starting with $v_i^{(0)} = 0$

$$P_{ii}(v_i^{(k+1)} - v_i^{(k)}) = -A_{ii}v_i^{(k)} + u_i \tag{5.19}$$

for $i = 1, 2, \ldots, n$ and $u_i = A_{is}w_s$. Notice that (5.19) can be implemented and $A_{si}v_i^{(m+1)}$ formed, for $i = 1, \ldots, n$, completely in parallel.

Another approach is to use MILU preconditioners in the subdomains $A_{ii} \approx L_iU_i$ to construct the matrix K_{ss}. Specifically, we take

$$K_{ss} = -\sum_{i=1}^{n} A_{si}(L_iU_i)^{-1}A_{is}. \tag{5.20}$$

Here the matrix-vector product may be evaluated as follows:

$$K_{ss}w_s = -\sum_{i=1}^{n} A_{si}v_i \tag{5.21}$$

where v_i can be determined by

$$L_i\hat{v}_i = A_{is}w_s \tag{5.22}$$

and

$$U_i v_i = \hat{v}_i \tag{5.23}$$

for $i = 1, 2, \ldots, n$. The solutions of (5.22) and (5.23) can be carried out and $A_{si}v_i$ formed in parallel.

Numerical results indicate that the spectral radii $\rho(I_{ss} - (A_{ss} + K_{ss})^{-1}C)$ for various resolutions are around .04 by retaining just the first term in the Neumann series and 0.002 by using the MILU factorization. Thus, roughly either only four or two iterations, respectively, will be required in order to reduce the norm of the initial error vector on the interfaces by a factor of 10^{-6}.

5.2.3 The Algorithm

Prior to presenting an algorithm of the MIM domain decomposition approach, we discuss here how to choose the initial guesses for the solution of (5.4) as well as that of (5.5) and how to precondition (5.5).

Rather than making a random guess of the initial vector for an iterative solver to start with, we take $x_i^{(k)}$, for $i = 1, 2, \ldots, n$, as the initial guess for the solution of each of the linear systems in (5.4), for $k = 1, 2, \ldots$, as well as the zero vector as the initial guess for the modified interface matrix linear system (5.4), for $k = 0, 1, \ldots$.

It is clear from theorem 5.2 that, as k increases, $x_i^{(k+1)} - x_i^{(k)} \to 0$ in the subdomains Ω_i, for $i = 1, 2, \ldots, n$, and $\Delta x_s^{(k)} \to 0$ on the interfaces Γ. Consequently, the initial vectors selected this way become better approximations to the subdomain solutions $x_i^{(k+1)}$, for $i = 1, 2, \ldots, n$, as well as to the solution on the interfaces $\Delta x_s^{(k)}$, as the iterative procedure defined by (5.4) and (5.5) proceeds, and thus requiring fewer iterations[1] for the iterative solutions in the subdomains (5.4) and on the inter-

[1]The numerical results on this are very encouraging, see Figures 5.5 and 5.7 on pages 138 and 139.

faces (5.5). As a result, the computational cost of the MIM domain decomposition algorithm decreases as k increases. The reduced cost for the subdomains mitigates the disadvantage of unavailability of fast subdomain solvers.

For any fixed k, we use here essentially the same preconditioner G (see (4.36)) as previously used for the preconditioning of (4.16), with the Schur complement matrix replaced by the matrix $A_{ss} + K_{ss}$. Only a single probing vector of the form $e = \{1, 1, 1, 1, 1, 1, \ldots\}^T$ is required for this construction. The product $K_{ss}e$ is evaluated by using either (5.18) or (5.21). If (5.21) is used, it is easy to see that just one inexact solve is required for the construction of this preconditioner in each subdomain.

Based on the above discussion, we present in the following an algorithm for the MIM domain decomposition approach, where Steps 1 to 3 are required for set-up purposes. The iterative procedure is described by Steps 4 through 6. To fix ideas, we specify CGS to be our iterative solver in the algorithm.

Algorithm MIMDD (Modified Interface Matrix Domain Decomposition)

Step 1. Carry out the MILU factorizations of A_{ii}, $A_{ii} \approx L_i U_i$, for $i = 1, 2, \ldots, n$, in parallel.

Step 2. Construct a modified rowsum preserving interface probing preconditioner for (5.5). A large part of this calculation may be carried out in parallel by using either (5.18) or (5.21).

Step 3. Set $k = 0$. Specify $x_s^{(0)}$ on the interfaces and solve subdomain problems (5.4) in parallel by using a MILU preconditioned CGS solver with a suitable initial solution.

Step 4. Solve (5.5) by the CGS solver with the preconditioner constructed in Step 2, and an initial solution taken to be the zero vector on the interfaces. Notice that the preconditioning system may be solved in parallel and the matrix-vector product $K_{ss}w_s$ may be computed by either (5.18) or (5.21) mostly in parallel, where w_s varies between iterations.

Step 5. Test for convergence on the interfaces (by using (5.7)). If the convergence criterion is met, go to Step 6 and stop: Otherwise, go to Step 6 and then go back to Step 4.

Step 6. Set $k \leftarrow k+1$. Solve subdomain problems (5.4) in parallel by using the MILU PCGS solver with initial solutions $x_i^{(k)}$, for $i = 1, 2, \ldots, n$.

5.3 Numerical Results and Discussions

5.3.1 Accuracy of the Modified Interface Matrix

We now provide some numerical results along with discussions to support the theory and algorithm introduced in Section 5.2. First, the critical factors affecting the successful application of the theory and algorithm discussed there are the spectral radii $\rho(I_{ss} - A_{ss}^{-1}C)$ and $\rho[I_{ss} - (A_{ss} + K_{ss})^{-1}C]$, respectively. Our goal is to render the matrix $A_{ss} + K_{ss}$ to be an accurate representation of the Schur complement matrix C, in the sense that $\rho[I_{ss} - (A_{ss} + K_{ss})^{-1}C]$ is small, without sizably increasing the computational work for the construction of the matrix K_{ss}. Our numerical experiments applied to various mesh resolutions indicate that the condition (5.3) is satisfied and that the advantage of modifying A_{ss} to $A_{ss} + K_{ss}$ is computationally significant.

The spectral radii $\rho(I_{ss}-A_{ss}^{-1}C)$ and $\rho[I_{ss}-(A_{ss}+K_{ss})^{-1}C]$ corresponding to the non-dimensionalized geopotential at the end of one hour (for a time step of half an hour) for several mesh resolutions are summarized in Table 5.1, where the matrix K_{ss} is constructed by MILU factorizations. The spectral radius $\rho[I_{ss}-(A_{ss}+K_{ss})^{-1}C]$, for which K_{ss} is formed by the Neumann series expansion (see (5.10 and (5.11)) can be found in our early paper [174]. Comparisons were also made in [174] between MILU and the Neumann series constructions of K_{ss}, the general conclusion being that MILU is more suitable for this particular application. By retaining more terms in the Neumann series expansion, we can render the spectral radius $\rho[I_{ss}-(A_{ss}+K_{ss})^{-1}C]$ very small, however, the extra CPU time required usually outweighs the gain obtained by the reduction of the number of outer iterations of the MIMDD algorithm. As expected, the spectral radius increases with the mesh refinement, implying that additional outer iterations will be required for satisfying the same prescribed convergence criterion.

Table 5.1: The spectral radii of the matrices $I_{ss}-A_{ss}^{-1}C$ and $I_{ss}-(A_{ss}+K_{ss})^{-1}C$ for three mesh resolutions

Mesh resolutions	$\rho(I_{ss}-A_{ss}^{-1}C)$	$\rho[I_{ss}-(A_{ss}+K_{ss})^{-1}C]$
36×27	0.134	1.930×10^{-3}
56×47	0.134	2.236×10^{-3}
72×63	0.191	3.485×10^{-3}

The modified interface matrix $A_{ss}+K_{ss}$ constructed according to (5.20) constitutes quite a good approximation to the Schur complement matrix C, defined in (4.18) or (4.19) of Chapter 4. A plot of the surface formed by the elements of the modified interface matrix as a function of its indices can hardly be distinguished

from Figure 4.7. In view of this, we present, in Figure 5.2, the surface generated by the entries in the matrix $C - (A_{ss} + K_{ss})$ corresponding to the non-dimensionalized geopotential matrix at the end of one hour of model integration for a time step of half an hour using, for the illustration purpose, a mesh resolution of 25×19. However, similar results are observed for cases corresponding to other mesh resolutions.

5.3.2 The Convergence Behavior

Now we present some convergence results obtained by using the iterative algorithms introduced in Theorems 5.1, which does not modify the interface matrix A_{ss}, and in Theorem 5.2, which is referred to as the modified interface matrix domain decomposition (MIMDD) algorithm [174] (its algorithm is described in detail in Algorithm MIMDD on page 126). The stopping criterion is that the final Euclidean residual norm $\|g - Cx_s^{(k)}\|$ of the Schur complement linear system on the interfaces be less than 0.1×10^{-10}, essentially the same convergence criterion as employed in Chapter 4. However, for the present algorithms, we do not have to deal with the Schur complement matrix for convergence tests, but just to take advantage of the relation given in (5.7) (setting $K_{ss} = 0$ for the unmodified case).

For each of the subdomain problems expressed in (5.1) or (5.4), as usual, the CGS iterative algorithm with an MILU preconditioner is used. On the interfaces, however, for the unmodified case, due to the special structure of the matrix A_{ss}, it is more cost effective to employ the so-called Ahlberg-Nielson-Walsh algorithm, mentioned in Section 4.5.4; for the modified case, we use the CGS algorithm accelerated by the modified row-sum preserving interface probing preconditioner (see Algorithm MIMDD).

In Figure 5.3, we display the convergence histories for both the unmodified and modified interface matrix domain decomposition algorithms on two mesh resolutions.

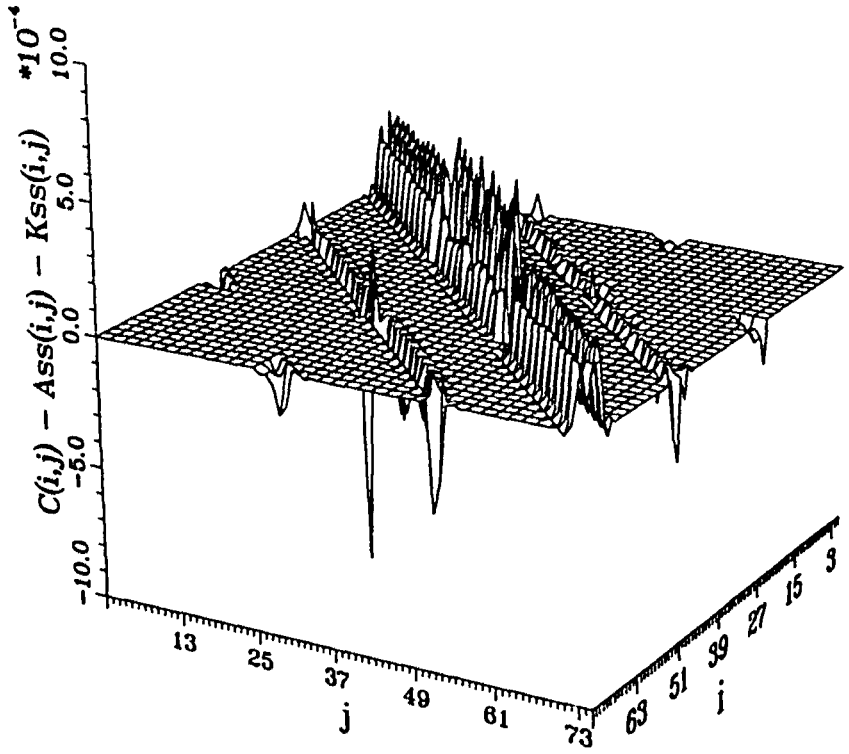
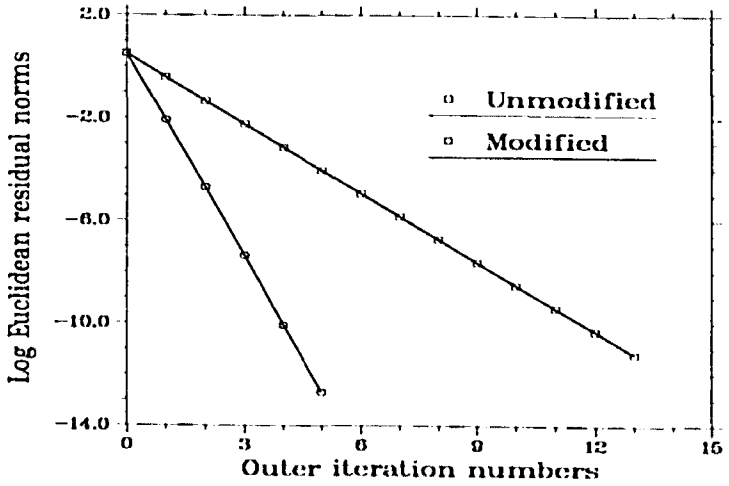


Figure 5.2: The surface generated by the entries of the matrix $C - (A_{ss} + K_{ss})$ for the non-dimensionalized geopotential system at the end of one hour of integration. The mesh resolution is 25×19 for the original domain. For this choice, there are 75 nodes on the interfaces for the four-subdomain domain decomposition. Note that the mesh lines have been thinned by a factor of two in both directions along i and j.

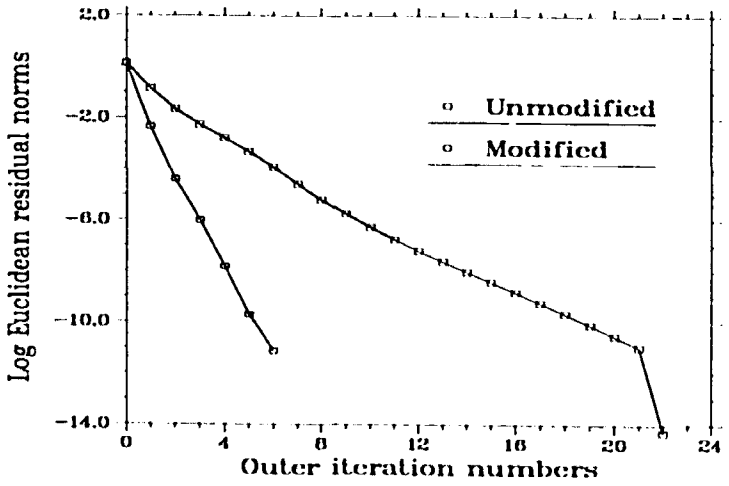
The results correspond to the non-dimensionalized geopotential matrix system at the end of one hour of model integration with a time step of half an hour. We see that the modified version represents a great saving of number of outer iterations and, as a result, a saving of number of subdomain solves. Actually, this is to be expected since the spectral radius $\rho[I_{ss} - (A_{ss} + K_{ss})^{-1}C]$ is of order $O(10^{-3})$, while $\rho(I_{ss} - A_{ss}^{-1}C)$ is of order $O(10^{-1})$. We also note that the unmodified approach is quite sensitive to the mesh refinement. On the other hand, the number of outer iterations required for MIMDD algorithm to converge does not increase very much even for higher mesh resolutions, implying a good adaptivity of the algorithm.

In Table 5.2, we privide some timing results which further confirm that the MIMDD algorithm is to be preferred to the corresponding unmodified version of the algorithm. Each given CPU time corresponds to the integration of the finite element shallow water equations model for one hour with a time step of half an hour. Also included in the table, for comparison purposes, are those timing results using the Schur domain decomposition method with the modified row-sum preserving interface probing preconditioner on the interfaces, denoted by SDD IV (see page 112). The number of outer iterations required for solving the geopotential linear systems at $t = 1$ hour for several mesh resolutions are also included in the table.

From the table, we observe that even the unmodified interface matrix domain decomposition algorithm is competitive with the Schur domain decomposition method, while MIMDD method is the best (i.e., computationally cheapest) among these three algorithms.



(a)



(b)

Figure 5.3: The evolution of log_{10} Euclidean residual norms of the Schur complement matrix linear system on the interfaces as a function of number of outer iterations for using the modified and unmodified interface domain decomposition algorithms. The results correspond to a non-dimensionalized geopotential matrix system at the end of one hour of integration with a time step of half an hour. The mesh resolution is (a) 60×51 and (b) 104×95, respectively.

Table 5.2: A comparison of CPU times in seconds for integration to the end of one hour with a time step of half an hour (numbers of outer iterations for solving the geopotential linear systems at $t = 1$ hour) between the unmodified, modified interface matrix domain decomposition algorithms (timing results for the Schur domain decomposition method are also included for comparison)

Mesh resolutions	Unmodified	Modified	SDD IV
24×15	1.07 (14)	0.78 (5)	0.97
36×27	2.87 (14)	2.17 (5)	2.74
48×39	5.94 (14)	4.52 (5)	5.75
60×51	10.72 (13)	8.59 (5)	11.04
72×63	17.95 (14)	14.63 (5)	17.96
84×75	29.08 (17)	24.01 (5)	28.62
96×87	46.27 (19)	37.31 (6)	46.39
104×95	68.69 (22)	51.23 (6)	69.23

5.3.3 The Significance of Successively Improved Initial Solutions in the Subdomains and on the Interfaces

One of the reasons that the MIMDD algorithm performs better than the traditional Schur domain decomposition method is due to the improved initial solutions in the subdomains and on the interfaces being systematically made as the outer iterations continue.

To show how these improved initial solutions affect the performance of both the MIMDD algorithm and the unmodified version, we present, in Figures 5.4, 5.5, 5.6 and 5.7, histories of improved initial solutions in the subdomains and on the interfaces (for the MIMDD algorithm only) for the non-dimensionalized geopotential matrix system at the end of one hour of model integration with a time step of half an hour, for both unmodified and modified interface matrix domain decomposition algorithms. We present these results corresponding to two mesh resolutions only, namely, 60×51 and 104×95. However, similar behavior can be observed for other mesh resolutions. For the case of higher resolution, we notice that the unmodified interface matrix algorithm fails to reduce quickly the initial residual norms in the second and third subdomains after each outer iteration step. However, the MIMDD algorithm self-adapts the mesh refinement and behaves well even for high mesh resolutions.

Finally, we report timing results to illustrate the significance of the improved initial solutions within each outer iteration. As before, we integrate the shallow water equations for one hour by using both unmodified interface matrix domain decomposition and MIMDD algorithms with or without the updates of the initial solutions in the subdomains. For cases in which no improvements are made, the initial subdomain solution vectors are simply taken to be zeroes. The numerical results are shown in Table 5.3, where the following notations have been adopted

UMIMDD I: unmodified interface matrix DD without improved initial subdomain solutions;

UMIMDD II: unmodified interface matrix DD with improved initial subdomain solutions;

MIMDD I: MIMDD algorithm without improved initial subdomain solutions;

MIMDD II: MIMDD algorithm with improved initial subdomain solutions.

Table 5.3: A comparison of CPU times in seconds between the unmodified and modified interface matrix domain decomposition algorithms with and without improvements of initial solutions made in the subdomains

Mesh resolutions	UMIMDD I	UMIMDD II	MIMDD I	MIMDD II
24×15	1.53	1.07	1.02	0.78
36×27	4.31	2.87	2.85	2.17
48×39	8.98	5.94	6.06	4.52
60×51	15.89	10.72	11.46	8.59
72×63	26.26	17.95	19.05	14.63
84×75	42.36	29.08	31.40	24.01
96×87	68.36	46.27	49.27	37.34
104×95	106.28	68.69	67.79	51.23

As can be expected that, for higher mesh resolutions, the subdomain problems become more costly to solve and hence the differences of the computational cost between UMIMDD I and UMIMDD II or between MIMDD I and MIMDD II become more pronounced. From these examples, it is clear that the importance of the

successive improvements of initial solutions made in each of the subdomains, as proposed in Algorithm MIMDD, can not be overemphasized.

5.4 Conclusions

- A new domain decomposition algorithm (MIMDD algorithm) has been proposed in this chapter along with some supporting theorems. In contrast with the Schur domain decomposition approach, in which the numerical solution on the interfaces is first determined, the MIMDD algorithm starts with an initial guess on the interfaces and then iterates back and forth between the subdomains and the interfaces until a convergence criterion is satisfied on the interfaces. Beginning from the second outer iteration step, the iterative subdomain and interface solvers become increasingly less expensive due to the successively improved initial solutions. The reduced computational cost in obtaining subdomain solutions this way mitigates the disadvantage of the unavailability of fast subdomain solvers for any specific applications. The results obtained by applying this algorithm to our application improve upon those obtained by employing the traditional Schur domain decomposition algorithm.

- With the modified interface matrix constructed by the MILU factorizations, the spectral radii $\rho[I_{ss}-(A_{ss}+K_{ss})^{-1}C]$ for several mesh resolutions tested are of order $O(10^{-3})$, compared with $O(10^{-1})$ for the unmodified version. Moreover, we also note that a significantly increased number of iterations will be required for the unmodified interface matrix approach to converge for higher mesh resolutions. However, the convergence of the MIMDD algorithm is only weakly dependent on the mesh size h, implying a good self-adaptivity to the mesh refinement.

- From Figures 5.4, 5.5, 5.6 and 5.7 as well as Table 5.3, we notice the remarkable effect of the iteratively improved initial solutions on reducing the overall computational complexity. These numerical results suggest a beneficial impact of the application of multigrid iterative solvers in the subdomains. Like any other iterative schemes for the solution of linear systems, multigrid method will perform even better with an appropriate initial solution provided at the finest level. In fact, the nested iteration is often combined with the coarse grid correction scheme (collectively called full multigrid scheme) to provide a good initial solution for the next finer level, starting from the coarsest level (see, for instance, [25, 127, 220]). From our experimental results, we conclude that, except for the first outer iteration, the nested iteration does not seem so important and coarse grid correction cycles may be applied directly for improving the solution at the finest level through a sequence of transferred residual information from the finer-grid to the coarser-grid. The MIMDD algorithm with multigrid subdomain solvers is currently under investigation.

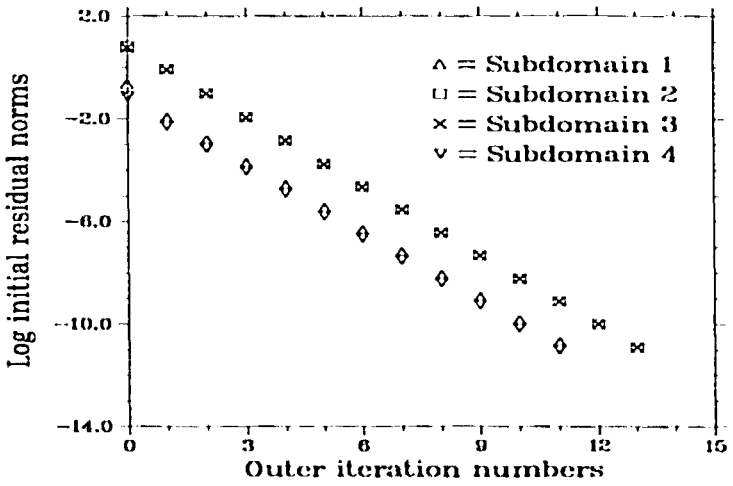


Figure 5.4: The history of improved initial solutions in the subdomains using the unmodified interface matrix domain decomposition algorithm for the non-dimensionalized geopotential matrix system at the end of one hour of integration with a time step of half an hour. The mesh resolution is 60×51.

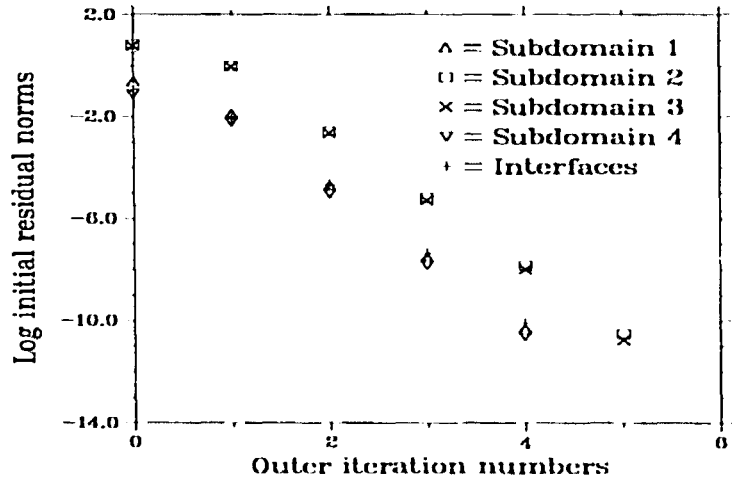


Figure 5.5: The history of improved initial solutions in the subdomains and on the interfaces using the MIMDD algorithm for the non-dimensionalized geopotential matrix system at the end of one hour of integration with a time step of half an hour. The mesh resolution is 60×51.

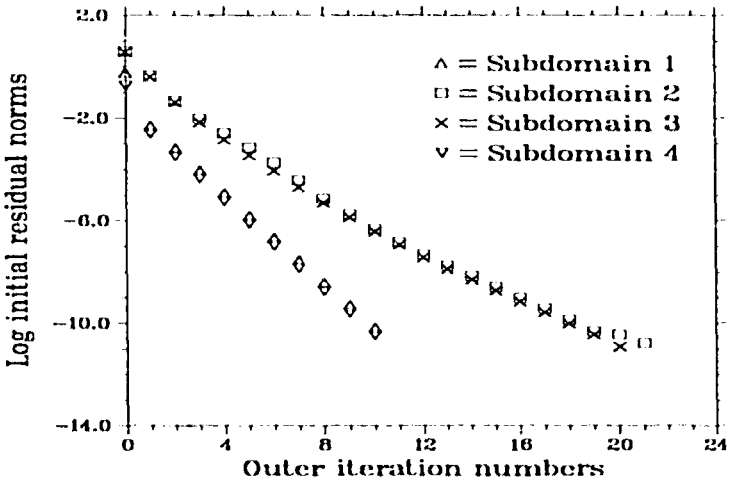


Figure 5.6: The history of improved initial solutions in the subdomains using the unmodified interface matrix domain decomposition algorithm for the non-dimensionalized geopotential matrix system at the end of one hour of integration with a time step of half an hour. The mesh resolution is 104×95.

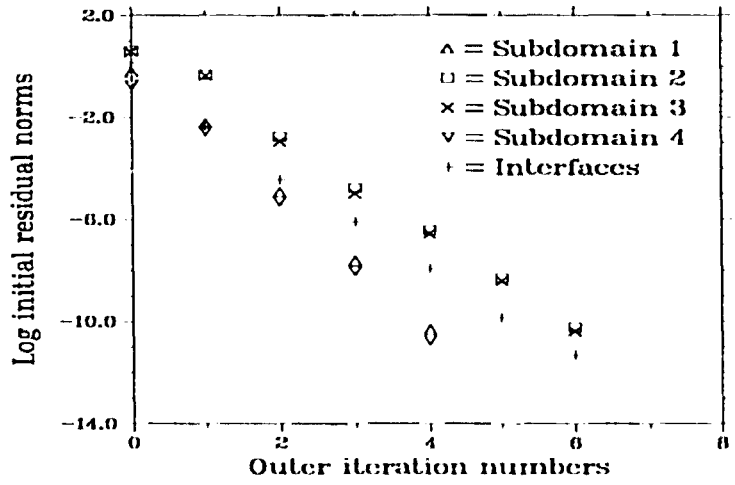


Figure 5.7: The history of improved initial solutions in the subdomains and on the interfaces using the MIMDD algorithm for the non-dimensionalized geopotential matrix system at the end of one hour of integration with a time step of half an hour. The mesh resolution is 104×95.

CHAPTER 6

PARALLEL BLOCK PRECONDITIONING TECHNIQUES AND APPLICATIONS

6.1 Introduction and Motivation

The numerical solution of elliptic or time dependent PDE's with implicit time discretization basically involves two stages, namely, 1) discretization and 2) numerical solution of the resulting systems of algebraic equations. Usually, the execution time for the first stage is only a small fraction of that for the second stage [188]. These algebraic equations may be linear or nonlinear corresponding to the nature of the original PDE's and the accuracy requirement. However, if the original problem indeed necessitates the solution of nonlinear systems of algebraic equations, many solution methods (see [101], [186] and references therein) may be interpreted as successive modifications, through a sequence of linear approximations, of an initial solution of the nonlinear system, until a certain convergence criterion is satisfied. In other words, linear solvers are essential kernels to nonlinear solvers. As a result, to reduce the major computational work involved in the second stage defined above, the availability of a cost effective (often problem dependent) linear solver is absolutely desirable for both linear and nonlinear systems of algebraic equations.

As a matter of fact, due to its important role in scientific and engineering computations, the search for the efficient solution of a linear system of algebraic equations has always been a central issue in numerical analysis. Multigrid methods

([24, 27, 127, 152] and/or domain decomposition techniques constitute modern approaches for the solution of a linear system of algebraic equations. Approaches of this type rely on and take advantage of an underlying continuous problem.

As has been introduced in Chapter 3, although domain decomposition ideas are traceable to the work of Schwarz [209] in 1869 and that of engineers beginning from the 1960's [193, 202, 93], an efficient way to handle the coupling between artificially divided substructures was first proposed by Dryja in 1982 [69], a work considered to be seminal, in the context of the Preconditioned Conjugate Gradient (PCG) linear symmetric iterative algorithms. In essence, this is a divide-and-feedback process which continues until a prescribed convergence criterion on the interfaces is satisfied (see Chapter 4 for details).

Since the feasibility of this process is based on the property that only matrix-vector products are required for the PCG algorithm for solving a symmetric linear system, the idea is readily extendible to the non-symmetric case. However, this approach, which proceeds via the Schur complement matrix, usually requires repeated exact subdomain solutions which are not cheaply implementable for non-separable elliptic operators or for other more complicated cases such as, for instance, the shallow water equations.

To improve the efficiency of the parallel solution of partial differential equations, for which no fast subdomain solvers are available, at least two other approaches have been proposed. One is a domain decomposed preconditioner approach (DDPA) (also called full matrix domain decomposition in [132]) advocated in [22], the other being the recently proposed modified interface matrix domain decomposition (MIMDD) (see [174] and Chapter 5). Both approaches abandon the idea of Schur domain decomposition method that decoupled subdomain problems are independently solved *only after* the interfacial degrees of freedom are specified.

In short, the DDPA consists essentially of the construction of a domain decomposed preconditioner designed so that approximate solutions in the subdomains and on the interfaces can be simultaneously updated at the cost of only inexact subdomain solves. On the other hand, the MIMDD algorithm successively improves the subdomain and interface approximate solutions with iterative improvements of the initial guesses in both the subdomains and interfaces being made. Thus it mitigates the disadvantage due to the absence of fast subdomain solvers.

The MIMDD algorithm has been presented and discussed in the last chapter. In this chapter, we concentrate our attention on the development and application of the DDPA to the finite element shallow water flow simulation. Specifically, we consider three types of domain decomposed (DD) preconditioners and their applications. Solutions to the preconditioning linear systems are provided by inexact subdomain solves. Results concerning performance sensitivities of these preconditioners to inexact subdomain solvers will also be reported. Parallel implementation issues and speed-up results will be presented in Chapter 7.

Non-symmetric linear iterative solvers are important kernels to the current domain decomposition approach. Among many available algorithms, we are especially interested in three of them, namely, GMRES [206], CGS [216] and Bi-CGSTAB [227]. We expect that these three algorithms will be more extensively studied and compared by numerical analysts and be widely applied to many important problems in science and engineering. Although a thorough analysis and comparison of these three methods are beyond the scope of this dissertation, we share here our numerical experience related to their application with DD preconditioners in the context of numerical solution of equations describing the shallow water flow.

6.2 Parallel Domain Decomposed Preconditioners

6.2.1 An Equivalence Theorem and its Significance

The domain decomposition methods based on building DD preconditioners (interpreted as block preconditioning techniques) into the iterative linear solvers were proposed with more complicated and real-life problems in mind. They may be viewed as being motivated by the following theorem due to Eisenstat [131] for the conjugate gradient solution of a symmetric linear system (4.4).

Theorem 6.1 Algorithm PCG applied to $Cx_s = g$ (see (4.16)) with initial guess $x_s^{(0)}$ and preconditioner G is equivalent to algorithm PCG applied to $Ax = f$ (see (4.4)) with initial guess

$$x^{(0)} = (A_{11}^{-1}(f_1 - A_{1s}x_s^{(0)}), \ldots, A_{nn}^{-1}(f_n - A_{ns}x_s^{(0)}), x_s^{(0)})^T$$

and preconditioner

$$B = \begin{bmatrix} A_{dd} & A_{ds} \\ A_{sd} & G + \sum_{i=1}^{n} A_{si}A_{ii}^{-1}A_{is} \end{bmatrix} \tag{6.1}$$

in the sense that, for all $k \geq 0$,

$$x^{(k)} = (A_{11}^{-1}(f_1 - A_{1s}x_s^{(k)}), \ldots, A_{nn}^{-1}(f_n - A_{ns}x_s^{(k)}), x_s^{(k)})^T$$

and there is no advantage to choosing an initial guess more general than $x^{(0)}$ as above, in the sense that $\|x^{(k)} - x\|_A \leq \|w^{(k)} - x\|_A$, where $w^{(k)}$ is the k-th iterate generated by PCG from the initial guess

$$w^{(0)} = x^{(0)} + (\xi_1, \cdots, \xi_n, 0)^T$$

The theorem immediately suggests the algebraic form of a possible preconditioner for the PCG iterative solution of the symmetric system $Ax = f$. Instead of keeping

the subdomain stiffness matrices A_{ii}, for $i = 1, \ldots, n$, we use the approximations B_{ii} to replace A_{ii} in (6.1) such that only inexact subdomain solves are needed for the solution of the preconditioning linear system (say $Bp = q$) at each iteration step. We reach a preconditioner of the following form

$$B = \begin{bmatrix} B_{dd} & A_{ds} \\ A_{sd} & G + \sum_{i=1}^{n} A_{si} B_{ii}^{-1} A_{is} \end{bmatrix} \tag{6.2}$$

Let us take right preconditioning as an example to clarify some basic concepts. As discussed in Section 4.5.2 of Chapter 4, instead of solving the original linear system $Ax = f$, we solve $\grave{A}\grave{x} = f$, where $\grave{A} = AB^{-1}$ and $\grave{x} = Bx$. The matrix-vector product $\grave{A}q$ required in each iteration step is obtained by solving $Bp = q$ and then form Ap. The final solution is given by $x = B^{-1}\grave{x}$. Obviously, an efficient preconditioner B must satisfy the following three requirements

- AB^{-1} is better conditioned;
- The preconditioning matrix B is easy to invert;
- The solution of the preconditioning linear system $Bp = q$ is parallelizable.

The preconditioner given by (6.2) can greatly reduce the condition number of the matrix A if one selects B_{ii}, $i = 1, 2, \ldots, n$, to be good approximations (say, incomplete LU or a few multigrid cycles) of A_{ii} and constructs a good interface preconditioner G. The fact that $Bp = q$ can be solved in parallel will become clear in Section 6.2.2. It may be readily verified that the solution to the preconditioning linear system $Bp = q$ may be obtained at the cost of $2n$ inexact subdomain solves, where n is the number of subdomains.

6.2.2 Three Types of Domain Decomposed Preconditioners

For problems in which the fast subdomain solvers are not available, DDPA may turn out to be more efficient. Instead of solving for the interface unknowns first, this approach simultaneously updates, at the cost of only inexact subdomain solves, the approximate solutions in both the subdomains and on the interfaces. The idea here is to directly solve the linear system $Ax = f$, where the matrix A has a block-bordered structure as shown in (4.5), with an appropriate preconditioner having the same block-bordered structure. Two types of such preconditioners were reviewed in the literature (see [131], for instance).

To derive three types of DD preconditioners which are of interest here, we notice that the block-bordered matrix A in (4.5) may be factorized as

$$A = \begin{bmatrix} A_{dd} & A_{ds} \\ A_{sd} & A_{ss} \end{bmatrix} = \begin{bmatrix} A_{dd} & 0 \\ A_{sd} & C \end{bmatrix} \begin{bmatrix} A_{dd}^{-1} & 0 \\ 0 & C^{-1} \end{bmatrix} \begin{bmatrix} A_{dd} & A_{ds} \\ 0 & C \end{bmatrix} \tag{6.3}$$

where C is the Schur complement matrix given in (4.19).

Searching for possible preconditioners of A, we consider another matrix B factorized in exactly the same way as that in (6.3)

$$B = \begin{bmatrix} B_{dd} & 0 \\ A_{sd} & G \end{bmatrix} \begin{bmatrix} B_{dd}^{-1} & 0 \\ 0 & G^{-1} \end{bmatrix} \begin{bmatrix} B_{dd} & A_{ds} \\ 0 & G \end{bmatrix} \tag{6.4}$$

where B_{dd} and G are approximations of the matrices A_{dd} and C, respectively. The matrix B may also be written as

$$B = \begin{bmatrix} B_{dd} & A_{ds} \\ A_{sd} & B_{ss} \end{bmatrix} \tag{6.5}$$

where

$$B_{ss} = G + A_{sd} B_{dd}^{-1} A_{ds} = G + \sum_{i=1}^{n} A_{si} B_{ii}^{-1} A_{is} \tag{6.6}$$

It is easy to see that (6.5) is exactly the same as (6.2), although we derive it in a different way.

The first type of DD preconditioners considered here are of the structurally symmetric form given by (6.5), where A_{ds} and A_{sd} are given by (4.7) and (4.8). The matrix B_{dd} has the same structure as that given by (4.6) except that each B_{ii}, $i = 1, 2, \ldots, n$, is now an approximation of A_{ii}. For example, B_{ii} might be the relaxed incomplete LU factorization (RILU) [11] of A_{ii}. B_{ss} is given by (6.6), where G is an appropriate preconditioner to the Schur complement matrix C.

It may be verified that the solution of the preconditioning linear system $Bp = q$ is equivalent to solving the following linear systems

$$Gp_s = q_s - \sum_{i=1}^{n} A_{si} B_{ii}^{-1} q_i \tag{6.7}$$

$$B_{ii} p_i = q_i - A_{is} p_s, \qquad \text{for } i = 1, 2, \ldots, n \tag{6.8}$$

where the meaning of p_i and p_s is clear.

Instead of solving a linear system with a coefficient matrix B_{ii} exactly, we may equivalently solve the original linear system with coefficient matrix A_{ii} approximately. Therefore, the preconditioning system $Bp = q$ may be solved in the following fashion:

1. Solve approximately in each subdomain

$$A_{ii} p_i^{(1)} = q_i \tag{6.9}$$

for $i = 1, \ldots, n$ in parallel;

2. Solve the interface preconditioning system

$$Gp_s = q_s - \sum_{i=1}^{n} A_{si} p_i^{(1)} \tag{6.10}$$

3. Solve approximately in each subdomain

$$A_{ii}p_i^{(2)} = -A_{is}p_s \tag{6.11}$$

for $i = 1, \ldots, n$ in parallel;

4. Form

$$p_i = p_i^{(1)} + p_i^{(2)}, \qquad \text{for } i = 1, \ldots, n. \tag{6.12}$$

The second type of DD preconditioners, applicable only to a nonsymmetric linear system, is obtained by taking the rightmost factor in (6.4). It assumes the following block upper triangular form

$$B = \begin{bmatrix} B_{dd} & A_{ds} \\ 0 & G \end{bmatrix}. \tag{6.13}$$

Now the solution of the preconditioning system $Bp = q$ requires only one inexact subdomain solve in each subdomain, compared with two such solves in the previous case with B given by (6.5). Obviously, the solution p of $Bp = q$ can be obtained by first solving the preconditioning system on the interfaces

$$Gp_s = q_s \tag{6.14}$$

and then approximately solving in each subdomain

$$A_{ii}p_i = q_i - A_{is}p_s \tag{6.15}$$

for $i = 1, \ldots, n$ in parallel.

The third type of DD preconditioners is obtained by considering the leftmost factor in (6.4). This type of preconditioners assumes the following block lower triangular form

$$B = \begin{bmatrix} B_{dd} & 0 \\ A_{sd} & G \end{bmatrix}. \tag{6.16}$$

We need to solve approximately

$$A_{ii}p_i = q_i \tag{6.17}$$

for $i = 1, \ldots, n$ in parallel and then the preconditioning linear system on the interfaces

$$Gp_s = q_s - \sum_{i=1}^{n} A_{si}p_i. \tag{6.18}$$

A couple of observations arise immediately for this type of preconditioners. First, similar to the second type of preconditioners, it applies only to non-symmetric systems of algebraic equations due, typically, to the discretization of the convection terms. Second, the computational work involved here for this third type of DD preconditioners is only part of that required for the first type — compare (6.17) and (6.18) with (6.9) and (6.10). However, a greatly improved computational efficiency results. This is to be expected theoretically and is confirmed by numerical computations (see Section 6.4).

6.2.3 Analysis of Preconditioners

A question arises as to what constitutes an appropriate algebraic form for the interface preconditioner G for each of the three types of DD preconditioners. Let us first consider right preconditioning. Results corresponding to left preconditioning will be given toward the end of the section. For preconditioners of the first two types, G may be constructed as a preconditioner to the Schur complement matrix. This construction is, however, not appropriate for preconditioners of the third type and it leads to a deterioration in performance (see Section 6.4). In order to see this, we provide formulas for AB^{-1} below in which a matrix M is used to replace G in (6.6), (6.13) and (6.16). Due to the importance of these formulas, we derive them in the form of a lemma.

The following lemma may be readily verified following the procedure demonstrated in [18, pp. 71-72].

Lemma 6.1 Let

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \tag{6.19}$$

If A, A_{11} and A_{22} are nonsingular, then

$$A^{-1} = \begin{bmatrix} X & -A_{11}^{-1}A_{12}W \\ -WA_{21}A_{11}^{-1} & W \end{bmatrix} = \begin{bmatrix} X & -XA_{12}A_{22}^{-1} \\ -A_{22}^{-1}A_{21}X & W \end{bmatrix} \tag{6.20}$$

where

$$W = (A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1} = A_{22}^{-1} + A_{22}^{-1}A_{21}XA_{12}A_{22}^{-1} \tag{6.21}$$

$$X = A_{11}^{-1} + A_{11}^{-1}A_{12}WA_{21}A_{11}^{-1} = (A_{11} - A_{12}A_{22}^{-1}A_{21})^{-1}. \tag{6.22}$$

On applying the Lemma to (6.5), (6.13) and (6.16) to obtain the inverses and by post-multiplying (4.5) by these inverses, we obtain

$$AB^{-1} = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix} \tag{6.23}$$

where P_{ij}'s, $i, j = 1, 2$, are determined below.

- For the first type of DD preconditioners

$$P_{11} = A_{dd}B_{dd}^{-1} + (A_{dd}B_{dd}^{-1} - I_{dd})A_{ds}M^{-1}A_{sd}B_{dd}^{-1} \tag{6.24}$$

$$P_{12} = (I_{dd} - A_{dd}B_{dd}^{-1})A_{ds}M^{-1} \tag{6.25}$$

$$P_{21} = [I_{ss} - (A_{ss} - A_{sd}B_{dd}^{-1}A_{ds})M^{-1}]A_{sd}B_{dd}^{-1} \tag{6.26}$$

$$P_{22} = (A_{ss} - A_{sd}B_{dd}^{-1}A_{ds})M^{-1} \tag{6.27}$$

- For the second type of DD preconditioners

$$P_{11} = A_{dd}B_{dd}^{-1} \tag{6.28}$$

$$P_{12} = (I_{dd} - A_{dd}B_{dd}^{-1})A_{ds}M^{-1} \tag{6.29}$$

$$P_{21} = A_{sd}B_{dd}^{-1} \tag{6.30}$$

$$P_{22} = (A_{ss} - A_{sd}B_{dd}^{-1}A_{ds})M^{-1} \tag{6.31}$$

- For the third type of DD preconditioners

$$P_{11} = (A_{dd} - A_{ds}M^{-1}A_{sd})B_{dd}^{-1} \tag{6.32}$$

$$P_{12} = A_{ds}M^{-1} \tag{6.33}$$

$$P_{21} = (I_{ss} - A_{ss}M^{-1})A_{sd}B_{dd}^{-1} \tag{6.34}$$

$$P_{22} = A_{ss}M^{-1} \tag{6.35}$$

By examining (6.27), (6.31) and (6.35), one may see that, for the first two types of preconditioners, the optimal interface preconditioner M (in the sense that $P_{22} = I_{ss}$) should be constructed so that

$$M = A_{ss} - A_{sd}B_{dd}^{-1}A_{ds} = A_{ss} - \sum_{i=1}^{n} A_{si}B_{ii}^{-1}A_{is}. \tag{6.36}$$

Hence, the matrix M essentially consists of the approximate Schur complement matrix of the problem. For preconditioners of the third type, however, $M = A_{ss}$ is an optimal choice. Under these choices, $P_{21} = 0$ and $P_{22} = I_{ss}$ for the first type of preconditioners. $P_{22} = I_{ss}$ for the second type. $P_{21} = 0$ and $P_{22} = I_{ss}$ for preconditioners of the third type.

At every time step, a certain amount of work is necessary for the construction of the matrix M for each of the first two types of DD preconditioners. However, this computational work is not required for the third type of DD preconditioners. After

one has constructed the preconditioner B, the major computational work required for solving the preconditioning linear systems of the form $Bp = q$ at each iteration step, corresponding to each of the three types of DD preconditioners is summarized in Table 6.1. It is clear that, if the number of iterations required for the residual norm to decrease by a predetermined order of magnitude is roughly the same for all three types of DD preconditioners, the iterative method accelerated by preconditioners of the third type will consume the least amount of CPU time, while application of preconditioners of the first type turns out to be the most computationally expensive.

Table 6.1: A comparison of the amount of work required for solving preconditioning linear system $Bp = q$ corresponding to the three types of DD preconditioners

First type	$2n$ inexact subdomain solves
Second type	n inexact subdomain solves
Third type	n inexact subdomain solves

Although the matrix M is explicitly available for preconditioners of the third type, for the first two types of DD preconditioners, nn_s inexact subdomain solves must be carried out for the construction of M so that $P_{22} = I_{ss}$. Here n is the number of subdomains and n_s is the number of degrees of freedom on the interfaces. The exact construction of the matrix M according to (6.36) is computationally too expensive. To reduce the number of subdomain solves and to improve computational efficiency, we must be satisfied with an approximate construction of M. Fortunately, similar to what we have observed in Chapter 4, the action of the matrix operator M is, in fact, predominantly local and the matrix M, although dense, allows itself to be approximated by a very low-bandwidth sparse matrix G.

The particular structure of the matrix M can be examined in a three dimensional space by plotting a surface formed of the entries of M viewed as a function of its two indices. If we approximate each subdomain stiffness matrix A_{ii} by its ILU factorization $B_{ii} = L_i U_i$, the surface M constructed based on (6.36) can not be distinguished from that in Figure 4.7 or 4.11 for the same mesh resolution. This clearly indicates a strong coupling between neighboring nodes and very weak dependence between nodes which are a mesh-size distance apart on the interfaces. This suggests that the idea of interface probing techniques [45], which was developed mainly for elliptic PDE's, may be extended to the current problem. See [44] and Section 4.5.4 in Chapter 4 for a discussion on interface probing ideas.

Similar to the idea introduced in Chapter 4, we construct an approximation G to the matrix M by using a modified version of the rowsum preserving interface probing preconditioner (see Section 4.5.4, Chapter 4). Following [35], we denote this particular construction by MIP(0). For convenience, we set $M = G = A_{ss}$ for the third type of preconditioners. Thus, for these three types of DD preconditioners, the matrix G preserves the block diagonal structure of A_{ss} (see (4.13) and (4.36)). The linear system (6.10), (6.14) or (6.18) can be split up into $n-1$ smaller systems and the solution may be obtained in parallel.

With this particular interface probing construction MIP(0) of the matrix M, we can readily observe that the number of subdomain solves involved in using each of these preconditioners, for the solution of a linear system at each time step, is $(2k_1+3)n$, $(k_2+2)n$ and $(k_3+1)n$, respectively, where k_1, k_2 and k_3 are numbers of iterations required to achieve convergence for an iterative method (where only one matrix-vector multiplication is needed in each iteration, like GMRES) using the first, second and third types of DD preconditioners, respectively. Please note that,

in the above counts of subdomain solutions, the work required for recovering the final solution $x = B^{-1}\tilde{x}$ is also included.

As to inexact subdomain solvers, we employ the so-called relaxed incomplete LU factorization (RILU) [8, 11] to approximate the subdomain stiffness matrices A_{ii}, $i = 1, 2, \ldots, n$, namely, $B_{ii} = L_i U_i = A_{ii} + R_{ii}$, where R_{ii} is an error matrix. In short, given a matrix $A_{n\times n}$, the RILU factorization of A can be obtained in the following $n-1$ steps of transformation,

$$A = A^1 \rightarrow A^2 \rightarrow \cdots \rightarrow A^n = U$$

where, for $k = 1, 2, \ldots, n-1$, the following calculations are carried out

$$l_{ik} = a_{ik}^{(k)} / a_{kk}^{(k)}$$

$$a_{ij}^{(k+1)} = \begin{cases} a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}, & \text{if } (k+1 \le j \le n) \cap ((i,j) \in J) \cap i \ne j; \\ 0, & \text{if } (k+1 \le j \le n) \cap ((i,j) \notin J); \\ a_{ii}^{(k)} - l_{ik} a_{ki}^{(k)} + \omega \sum_{(p=k+1 \text{ and } (i,p)\notin J)}^{n} (a_{ip} - l_{ik} a_{kp}^{(k)}), & \text{if } j = i \end{cases}$$

here J is given in (4.31) and $0 \le \omega \le 1$.

To conclude this section, we privide some results corresponding to three types of domain decomposed preconditioners applied from the left. We may verify that if

$$B^{-1}A = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \tag{6.37}$$

then entries Q_{ij}'s, $i, j = 1, 2$, of the matrix $B^{-1}A$ are given by

- For the first type of DD preconditioners

$$Q_{11} = B_{dd}^{-1} A_{dd} + B_{dd}^{-1} A_{ds} M^{-1} A_{sd} (B_{dd}^{-1} A_{dd} - I_{dd}) \tag{6.38}$$

$$Q_{12} = B_{dd}^{-1} A_{ds} [I_{ss} + M^{-1} (A_{sd} B_{dd}^{-1} A_{ds} - A_{ss})] \tag{6.39}$$

$$Q_{21} = M^{-1} A_{sd} (I_{dd} - B_{dd}^{-1} A_{dd}) \tag{6.40}$$

$$Q_{22} = M^{-1} (A_{ss} - A_{sd} B_{dd}^{-1} A_{ds}) \tag{6.41}$$

- For the second type of DD preconditioners

$$Q_{11} = B_{dd}^{-1}(A_{dd} - A_{ds}M^{-1}A_{sd}) \tag{6.42}$$

$$Q_{12} = B_{dd}^{-1}A_{ds}(I_{ss} - M^{-1}A_{ss}) \tag{6.43}$$

$$Q_{21} = M^{-1}A_{sd} \tag{6.44}$$

$$Q_{22} = M^{-1}A_{ss} \tag{6.45}$$

- For the third type of DD preconditioners

$$Q_{11} = B_{dd}^{-1}A_{dd} \tag{6.46}$$

$$Q_{12} = B_{dd}^{-1}A_{ds} \tag{6.47}$$

$$Q_{21} = M^{-1}A_{sd}(I_{dd} - B_{dd}^{-1}A_{dd}) \tag{6.48}$$

$$Q_{22} = M^{-1}(A_{ss} - A_{sd}B_{dd}^{-1}A_{ds}) \tag{6.49}$$

We notice that, for left preconditioning, the approximate construction of an interface preconditioner is required for the first and third types of DD preconditioners, but not for the second type.

6.3 Iterative Methods For the Solution of Non-Symmetric Linear Systems of Algebraic Equations

While the PCG algorithm accelerated by suitable preconditioners seems to be the most competitive for the solution of a positive definite, symmetric system of linear algebraic equations, competitive non-symmetric linear solvers abound. According to [200], CGN (the conjugate gradient algorithm applied to the normal equation of the original non-symmetric system), CGS (Conjugate Gradient Squared) and GMRES (Generalized Minimal Residual) seem to be the most often used algorithms. A fast

squared Lanczos method for non-symmetric linear systems was recently proposed in [49]. This new algorithm was claimed to be very fast and robust compared to GMRES. More recently, Bi-CGSTAB, a fast and smoothly convergent variant of the bi-conjugate gradient method, was developed [227].

There is only limited numerical experience at present with most of these non-symmetric iterative solvers, especially when applied to real life applications. In their efforts to generate part of production code for the modelling of weak plasma turbulence, Radicati, Robert and Succi [200] studied CGN, BCG (Biconjugate Gradient), CGS and GMRES algorithms as applied to non-symmetric time-dependent linear systems arising from discretization of their problem. According to their report, the CGS and GMRES algorithms yield the best performances and are highly competitive to each other.

Both the theory and application of iterative solutions of symmetric positive definite linear systems may be considered to be in a quite satisfactory state. However, both the state of theory and numerical experience with iterative solvers for non-symmetric linear systems are far from being satisfactory. Quite a number of methods have been proposed in the literature (see [205] for a good survey) for solving non-symmetric linear systems. Apparently, a clear winner remains to be identified.

In our work, we choose, among many others, three relatively new and competitive iterative algorithms, namely, GMRES [206], CGS [216] and Bi-CGSTAB [227] for solving the geopotential and velocity non-symmetric linear algebraic systems, arising from the finite element discretization of the shallow water equations, with three types of DD preconditioners discussed above. We choose to use preconditioning from the right only, since it does not yield results very different from those obtained by preconditioning from the left for our block preconditioning computations and

moreover a linear system preconditioned from the right preserves the residuals of the original linear system.

Various extensions have been made to generalize iterative algorithms for symmetric linear systems to the non-symmetric case. GMRES algorithm is such an extension of the conjugate residual method for symmetric indefinite linear systems [47]. Mathematically, GMRES method is equivalent to ORTHODIR [126] and the generalized conjugate residual method [80] and is closely related to Arnoldi's method [203]. GMRES is a rather robust algorithm and never breaks down the way Arnoldi's algorithm does [28].

On the other hand, the CGS and Bi-CGSTAB algorithms are extensions of the BCG (bi-conjugate gradient) method [88] for non-symmetric linear systems, which is an extension of the CG (conjugate gradient) method for solving symmetric, positive definite linear systems. Although Bi-CGSTAB is a relatively new method, the CGS algorithm has been studied for quite a while. The broad consensus reached is that the CGS method is very competitive with GMRES (see, among others, [200, 199, 29]). In some cases, the CGS algorithm outperforms GMRES [17]. The comparison of performance of the GMRES, CGS and Bi-CGSTAB algorithms is currently of great interest in the area of numerical linear algebra.

The GMRES algorithm starts with an initial guess x_0 for the linear system $Ax = b$ and computes the initial residual $r_0 = b - Ax_0$. At the k-th iteration, a correction vector z_k is chosen such that

$$z_k = \arg \min_{z \in K_k(r_0)} \|b - A(x_0 + z)\| \tag{6.50}$$

where $K_k(r_0)$ is the Krylov subspace

$$K_k(r_0) = \text{span}\{r_0, Ar_0, \ldots, A^{k-1}r_0\}. \tag{6.51}$$

The approximate solution at k-th iteration is then $x_k = x_0 + z_k$. It is clear from (6.50) that the residual norm will never increase from one iteration to the next, which explains the smooth convergence behavior of the GMRES method (see Section 6.4). In contrast, Arnoldi's algorithm generates residual norms which exhibit a saw-tooth pattern.

The implementation of the GMRES algorithm involves solution of a least squares problem (6.50) on the Krylov subspace (6.51) at each iteration. This is done by generating a particular orthonormal basis $\{v_1, v_2, \ldots, v_k\}$ of $K_k(r_0)$ and an upper Hessenberg matrix H_k using a modified Gram-Schmidt process in the Arnoldi's method [206]. This is the approach we take in our numerical experiment, although another implementation approach is also possible using the Householder transformation [232]. To reduce computer storage requirements, we may use restarted GMRES(m), which is, nevertheless, usually accompanied by more iterations before convergence. In our experiment, however, GMRES is used without restarting.

The CGS algorithm was derived from the BCG algorithm by squaring the residual and direction matrix polynomials. The BCG algorithm generates two sequences of residuals r_i and $\tilde{r}_i$ by relations similar to the CG algorithm in which $(r_i, \tilde{r}_j) = 0$, for $i \neq j$. It was shown in [216] that $r_i = P_i(A)r_0$ and $\tilde{r}_i = P_i(A^T)\tilde{r}_0$ where $P_i(A)$ is a polynomial of degree i in the coefficient matrix A. The motivation for formulating CGS is the observation that $\rho_i = (\tilde{r}_i, r_i)$ in BCG can be expressed as

$$\rho_i = (\tilde{r}_i, r_i) = (P_i(A^T)\tilde{r}_0, P_i(A)r_0) = (\tilde{r}_0, P_i^2(A)r_0) \tag{6.52}$$

and a similar relation prevails for the direction vectors p_i and $\tilde{p}_i$. A detailed algorithm ready for computer implementation was given in [174].

Similar to BCG, for an $n \times n$ system matrix, CGS converges to the exact solution in m ($m \leq n$) iterations in the absence of round-off errors if it does not

break down. However, the CGS algorithm is generally more efficient than the BCG method since the transpose of the coefficient matrix explicitly appearing in BCG is no longer required in CGS. Moreover, CGS is known to amplify the effects of the Lanczos method and as a result it converges faster (roughly twice as fast theoretically, although often it fails to do so in practical applications).

The Bi-CGSTAB algorithm as proposed by van der Vorst [227] is expected to render the convergence behavior smoother by removing local peaks often appearing in the convergence curve for CGS. Bi-CGSTAB replaces the relations $r_i = P_i(A)r_0$ and $p_{i+1} = T_i(A)r_0$ in BCG with the recurrence relations $r_i = Q_i(A)P_i(A)r_0$ for the residual vector and $p_{i+1} = Q_i(A)T_i(A)r_0$ for the direction vector, in which $Q_i(x) = \Pi_{k=1}^{i}(1 - \omega_k x)$. The constants ω_k are computed so that the residual r_i is minimized in 2-norm. The new algorithm was found to converge faster and yield a smoother residual history than CGS for certain classes of problems.

It should be noted that other versions of Bi-CGSTAB have appeared in the literature since the publication of van der Vorst's paper — see [213] and references therein. However, we do not test other versions of the Bi-CGSTAB algorithm in this dissertation. To avoid confusion, we present below the unpreconditioned version of the Bi-CGSTAB algorithm, which is based on [227] cast in an algebraic form ready for computer implementation

$$r = b, \quad x = x_0, \quad r = r - Ax$$

$$\text{Choose } \tilde{r} \text{ such that } \delta 1 = (\tilde{r}, r) \neq 0, \quad p = r$$

$$p1 = Ap \tag{6.53}$$

$$\delta 0 = (\tilde{r}, p1), \quad \alpha = \delta 1/\delta 0, \quad s = r - \alpha p1$$

$$t = As, \quad \eta 0 = (t, t), \quad \eta 1 = (t, s), \quad \omega = \eta 1/\eta 0$$

$$x = x + \alpha p + \omega s$$

If the convergence criterion is met then stop, otherwise continue

$$r = s - \omega t, \quad \delta 1 = (\tilde{r}, r), \quad \beta = \delta 1/(\omega \delta 0), \quad p = r + \beta(p - \omega p 1)$$

Goto (6.53)

where the right hand side b and the initial guess x_0 are input vectors and (.) denotes the usual Euclidean inner product. This algorithm will be accelerated with right preconditioning in our numerical experiments.

6.4 Numerical Results and Discussions

In this section, we provide some carefully selected numerical examples along with discussions on the use of three types of DD preconditioners introduced in Section 6.2.2 and analyzed in Section 6.2.3. All numerical experiments are carried out on the CRAY Y-MP/432 vector parallel computer for the shallow water equations discussed in Section 4.6 of Chapter 4. A 3-D view of the initial geopotential is displayed in Figure 4.6. We scale our problem again by choosing $\varphi_0 = 10^2 m^2/s^2$. The corresponding dimensionless constants may be found in Section 4.7 of Chapter 4.

In the first set of numerical experiments, we test the relative efficiencies of the three aforementioned types of DD preconditioners. We use RILU (relaxed incomplete LU) factorization along with forward and back substitutions as inexact solvers in the subdomains.

The modified rowsum preserving interface probing preconditioner MIP(0) is used for the first two types of DD preconditioners on the interfaces. We also present computational results corresponding to preconditioners of the third type with G constructed, however, by MIP(0) instead of being equal to the preferred A_{ss}. The presentation of these numerical results is followed by a discussion.

The RILU [11] factorization consists of a combination or average of the ILU [154] and the MILU [109], in which the discarded fill-ins in the ILU factorization times a parameter ω, $0 \leq \omega \leq 1$, are added back to the diagonal entries (see Section 6.2.3). Obviously, $\omega = 0$ and 1 correspond to, respectively, ILU and MILU.

We compute and compare different values of ω and find that the optimal ω in terms of the number of iterations is in the interval $(0, 0.5)$, depending on which type of DD preconditioners and which iterative method (GMRES, CGS, or Bi-CGSTAB) is used. This result is in contrast to 0.95 found in a recent work [231] for some other computations in the original whole domain. On the other hand, in terms of CPU time, we find that the RILU with $\omega = 0$ yields the overall best performance. Hence, unless otherwise stated, the results presented below correspond to applying the RILU with $\omega = 0$, i.e., the ILU factorization to our problem.

6.4.1 The Convergence Behavior

The convergence behavior is visualized only for a mesh resolution of 80×75, which corresponds to 6000 nodes and 11692 triangular elements in the whole domain. The stopping criterion is that the final Euclidean residual norm of (4.4), namely, $\|Ax^{(k)} - f\|_2$ be smaller than 0.1×10^{-10}. A time-step size $\Delta t = 1800$ s is used throughout unless otherwise stated.

First, in Figure 6.1, we display the convergence history for each of the iterative methods, namely, GMRES, CGS and Bi-CGSTAB without preconditioning. Figures 6.2 to 6.5 present computational results of GMRES, CGS and Bi-CGSTAB with preconditioners of all three types. Results in Figure 6.5 are special, as we purposefully construct the matrix G by the modified rowsum preserving idea MIP(0) in order to show some deterioration, as mentioned before, in both computational time and number of iterations. Thus we emphasize the importance of the analysis carried

out in Section 6.2.3, where it was pointed out that $G = A_{ss}$ should be chosen. The computational work measured in terms of CPU seconds is reported for each preconditioned case in Table 6.2. Also included in the table are computational results corresponding to a coarser grid of 40×35 and a finer grid 120×115, respectively, on the whole domain.

Table 6.2: A comparison of CPU time (number of iterations) required for solving the geopotential linear system at the end of one hour with a half an hour time step using GMRES, CGS and Bi-CGSTAB algorithms accelerated by three types of DD preconditioners

Preconditioner types		First type	Second type	Third type	Third type with MIP(0)
40×35 mesh resolution	GMRES	0.178 (12)	0.102 (12)	0.088 (11)	0.101 (12)
	CGS	0.199 (7)	0.099 (6)	0.106 (7)	0.111 (7)
	Bi-CGSTAB	0.219 (7)	0.099 (6)	0.092 (6)	0.111 (7)
80×75 mesh resolution	GMRES	0.89 (14)	0.53 (15)	0.47 (14)	0.52 (15)
	CGS	1.08 (9)	0.54 (8)	0.50 (8)	0.59 (9)
	Bi-CGSTAB	0.97 (8)	0.54 (8)	0.50 (8)	0.58 (9)
120×115 mesh resolution	GMRES	2.56 (18)	1.49 (19)	1.41 (19)	1.48 (19)
	CGS	3.24 (12)	1.64 (11)	1.56 (11)	1.62 (11)
	Bi-CGSTAB	2.74 (10)	1.63 (11)	1.55 (11)	1.60 (11)

From these results, we conclude that the three non-symmetric iterative methods GMRES, CGS and Bi-CGSTAB are very competitive with each other. GMRES requires the largest number of iterations to attain convergence. However, this does not mean that GMRES is the most expensive algorithm to use since only one matrix-vector multiplication is required for each GMRES iteration, while two such

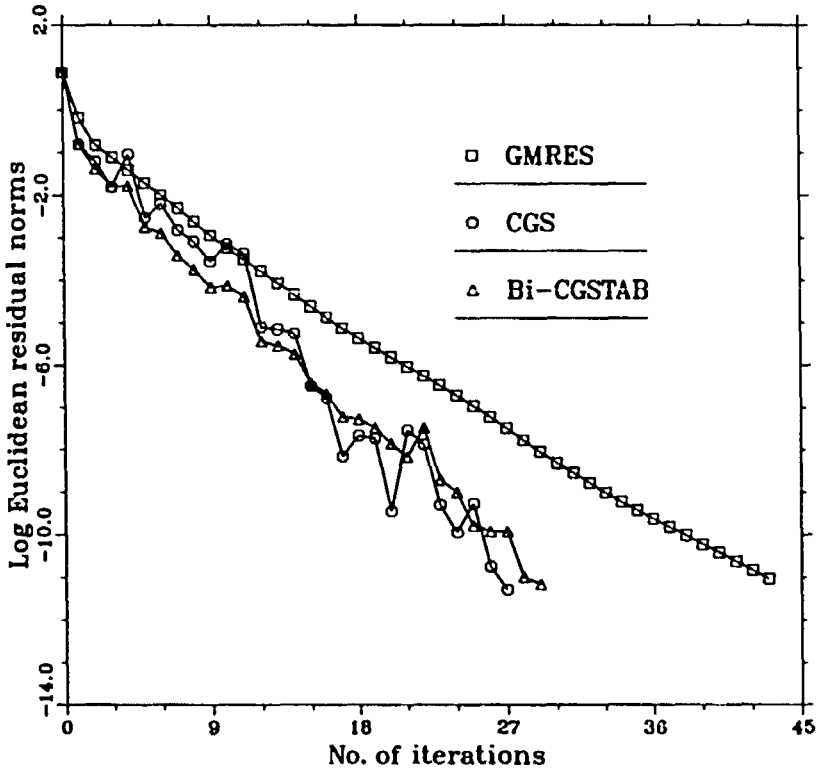


Figure 6.1: The evolution of $\log_{10}$ Euclidean residual norms as a function of the number of iterations for the iterative solution of the non-dimensionalized geopotential linear system at the end of one hour of model integration using GMRES, CGS and Bi-CGSTAB non-symmetric iterative linear solvers without preconditioning.

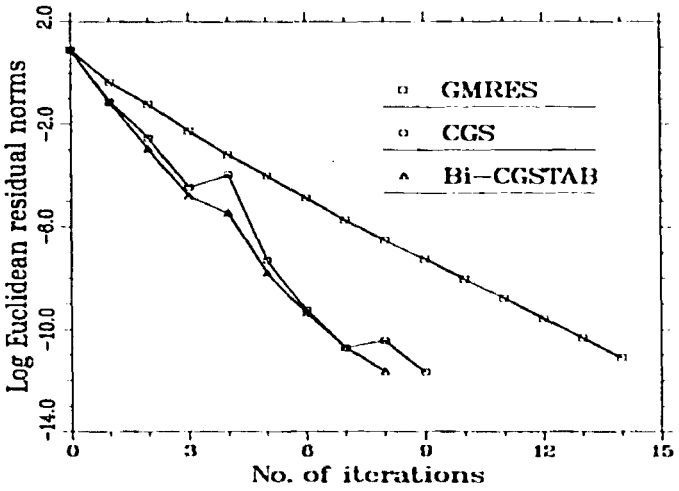


Figure 6.2: The evolution of $\log_{10}$ Euclidean residual norms as a function of the number of iterations for the iterative solution of the non-dimensionalized geopotential linear system at the end of one hour using GMRES, CGS and Bi-CGSTAB non-symmetric iterative linear solvers with a preconditioner of the first type.

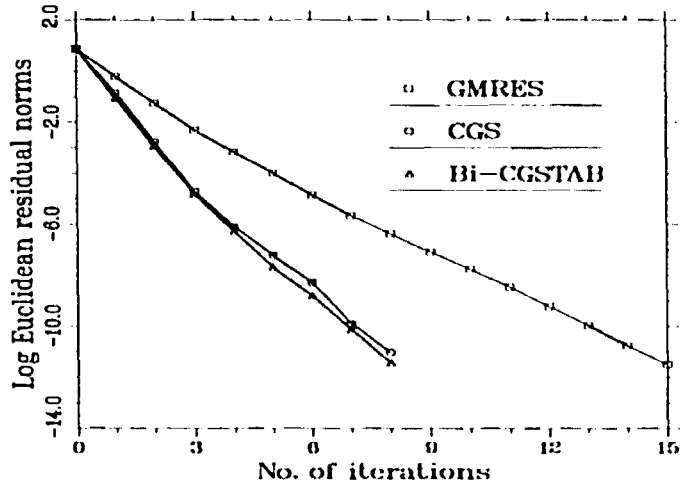


Figure 6.3: The evolution of $\log_{10}$ Euclidean residual norms as a function of the number of iterations for the iterative solution of the non-dimensionalized geopotential linear system at the end of one hour of model integration using GMRES, CGS and Bi-CGSTAB non-symmetric iterative linear solvers with a preconditioner of the second type.

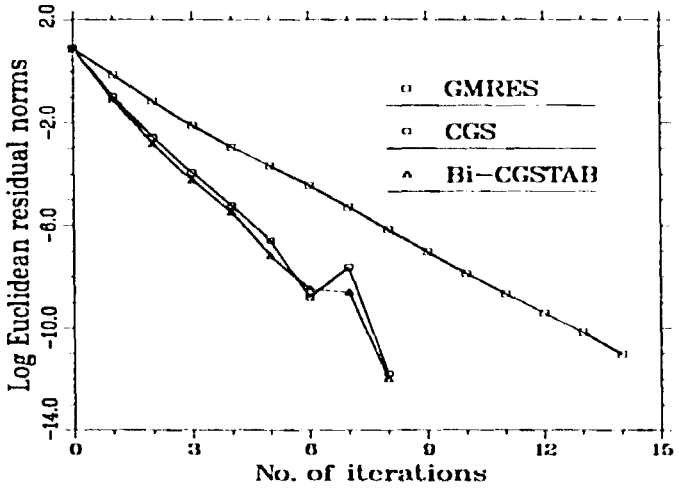


Figure 6.4: The evolution of $\log_{10}$ Euclidean residual norms as a function of the number of iterations for the iterative solution of the non-dimensionalized geopotential linear system at the end of one hour of model integration using GMRES, CGS and Bi-CGSTAB non-symmetric iterative linear solvers with a preconditioner of the third type.

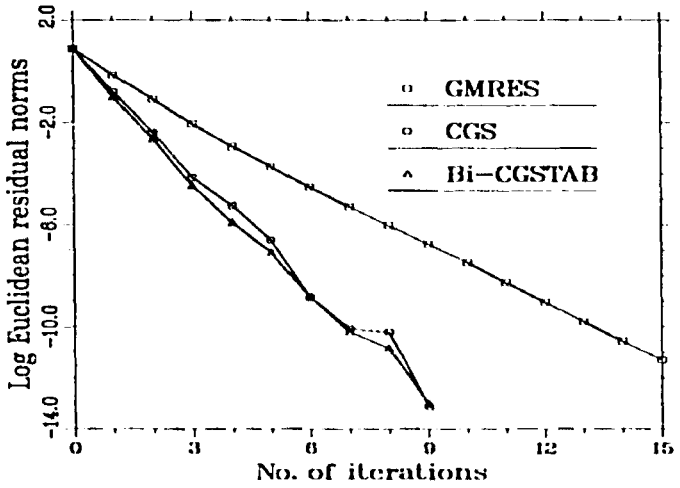


Figure 6.5: The evolution of $\log_{10}$ Euclidean residual norms as a function of the number of iterations for the iterative solution of the non-dimensionalized geopotential linear system at the end of one hour of model integration using GMRES, CGS and Bi-CGSTAB non-symmetric iterative linear solvers with a preconditioner of the third type and with interface probing construction of G.

operations are required for each CGS or Bi-CGSTAB iteration. However, GMRES imposes a higher demand for storage, which may be alleviated by restarting the procedure, often at the cost of requiring more iterations for convergence.

As has been observed for many other applications, the non-smooth convergence behavior of the CGS algorithm is also clearly seen in the current set of experiments. Designed to cure this disadvantage, Bi-CGSTAB converges much more smoothly. However, the GMRES method generates the smoothest residual convergence history. In fact, due to its minimization property at each iteration step, the residual norms produced by GMRES are always decreasing, or at least non-increasing.

We observe that no sizable difference exists in terms of the number of iterations required to attain convergence between these three types of preconditioners. Preconditioners of the first type, first proposed for a symmetric linear system arising from the discretization of some self-adjoint elliptic PDE's, can not reduce the number of iterations to such extent as to offset the disadvantage of two inexact subdomain solves in each subdomain for solving the preconditioning linear system $Bp = q$. As a result, they turn out to be the most expensive for the current application. Preconditioners of the second and third types behave much better in terms of CPU time due to only one inexact subdomain solves being required in each subdomain for the solution of $Bp = q$. Moreover, an approximate construction of G is not required on the interfaces for the third type of DD preconditioners.

6.4.2 Sensitivities of the Three Types of DD Preconditioners to Inexact Subdomain Solvers

In the above set of numerical experiments, we obtain the result that the number of iterations required for convergence is almost the same for the three types of preconditioners with ILU subdomain solvers and thus preconditioners of the third

type are the computationally least expensive. A question whcih naturally arises is whether similar conclusions will hold for the same types of preconditioners but with more or less accurate subdomain solvers compared to ILU. This essentially requires us to test sensitivities of these preconditioners to inexact subdomain solvers.

To test the sensitivities of the preconditioners given in (6.5), (6.13) and (6.16) to inexact subdomain solvers, we use the idea of m-step preconditioning [1]. For this preconditioning approach, we consider a splitting of the matrix $A_{ii} = P_i - Q_i$ and define $G_i = P_i^{-1}Q_i$. As an approximation of the subdomain matrix A_{ii}, we take $B_{ii} = P_i(\sum_{k=0}^{m-1} G_i^k)^{-1}$, for $i = 1, 2, \ldots, n$. The solution of $B_{ii}p_i = q_i$ may be easily verified to be equivalent to m iterations of the following linear stationary iterative scheme $P_i p_i^{k+1} = Q_i p_i^k + q_i$ with initial solution $p_i^0 = 0$.

The P_i may be taken to be any easily invertible simple matrix as long as the spectral radius $\rho(G_i) < 1$. For example, P_i may be chosen to be the ILU factorization, $P_i = L_i U_i$, of A_{ii}, or simply the diagonal part $P_i = D_i$ of the matrix A_{ii}. Once P_i has been chosen, B_{ii} often becomes a better inexact subdomain solver as m increases, in the sense that the number of iterations to reach a prescribed convergence criterion for solution of the original linear system $Ax = f$ decreases. By gradually increasing the number m, we were able to observe the relative performance behavior of these DD preconditioners corresponding to increasingly accurate subdomain solvers.

For the current experiment, we decided to take P_i as the lower triangular part (including the diagonal part) of the matrix A_{ii}. This corresponds to m Gauss-Seidel linear stationary iterations in each subdomain (see Section 4.5.2 of Chapter 4). However, this choice of P_i does not guarantee proper subdomain solvers for all mesh resolutions in our case. For example, we found that underrelaxed SOR iterations with $\omega = 0.5$ is appropriate for an 80×75 mesh resolution. In Table 6.3, 6.4 and 6.5, we present results, obtained for the 40×35 mesh resolution, of GMRES, CGS

and Bi-CGSTAB iterations corresponding to three types of DD preconditioners, respectively.

Table 6.3: Numbers of GMRES iterations as a function of m using three types of DD preconditioners

$m=1$	$m=2$	$m=3$	$m=4$	$m=5$	$m=6$	$m=10$	$m=20$
26	15	12	11	10	10	9	8
27	15	12	11	10	10	9	8
28	15	12	11	10	10	10	10

Table 6.4: Numbers of CGS Iterations as a function of m using three types of DD preconditioners

$m=1$	$m=2$	$m=3$	$m=4$	$m=5$	$m=6$	$m=7$
15	8	7	6	6	5	5
14	8	7	6	6	5	5
16	9	6	7	6	7	6

Table 6.5: Numbers of Bi-CGSTAB Iterations as a function of m using three types of DD preconditioners

$m=1$	$m=2$	$m=3$	$m=4$	$m=5$	$m=6$
14	8	7	6	6	5
14	8	7	6	6	5
14	8	6	6	6	6

From these three tables, we see that preconditioners of the third type can accelerate the convergence of three iterative methods at about the same rate as the other two types of preconditioners, except for cases when subdomain solvers may be considered to be exact or nearly exact. Similar observations are obtained for the case of mesh resolutions higher than 40×35. Since, in practice, the inexact subdomain solvers are far from being exact, we consider preconditioners of the third type to be the best. When the subdomain solvers are based on the complete LU factorizations, the solvers may be considered to be exact. Table 6.6 summarizes results corresponding to using exact subdomain solvers.

Table 6.6: Numbers of iterations when using exact subdomain solvers

Preconditioner types		First type	Second type	Third type
40×35 mesh resolution	GMRES	8	8	10
	CGS	5	5	6
	Bi-CGSTAB	5	5	6
80×75 mesh resolution	GMRES	10	10	14
	CGS	6	6	8
	Bi-CGSTAB	5	6	8

Clearly, for exact or almost exact subdomain solvers, fewer iterations are required if the first two types of preconditioners are used. However, the gain in the number of iterations is far from offsetting the additional computational cost required for constructing exact or almost exact subdomain solvers.

6.4.3 Extensions to the Cases of More Than Four Subdomains

For implementation on a large number of processors or, ambitiously, for massively parallel processing implementation, it is very desirable for domain decomposition algorithms to possess convergence rates that do not deteriorate as the number of subdomains increases. Unfortunately, it is often the case, rather than the exception, that the number of iterations will increase as the number of subdomains increases, even though the discrete problem size is kept fixed and the stopping criterion remains the same, for both overlapping and non-overlapping domain decomposition cases (see numerical results in, among others, [14, 129, 131, 157, 158, 159, 191]).

In few cases, the optimal preconditioners, in the sense that the convergence rate is independent of both the mesh size h and the typical subdomain size H, are known (see, for example, [214]). In other cases, nearly optimal preconditioners have been constructed with the property that the condition number of the preconditioned matrix is proportional to $(1 + \log(H/h))^m$, $m = 2$ or 3 (see [73, 74] and references cited therein). One of the most important reasons for the success of most of these preconditioners is the introduction, aimed at enhancing communications amongst subdomains, of a much smaller global problem corresponding to the discretization on a coarse grid. However, we notice that these optimal or nearly optimal domain decomposition algorithms may not be computationally the cheapest ways to obtain solutions to specific problems. The convergence rate alone does not tell the whole story of computational complexity.

In the following, we provide some numerical results corresponding to the convergence rates of the GMRES algorithm with each of the three types of DD preconditioners for $n = 2, 4, 8$ and 16 subdomains and for various mesh resolutions (see Table 6.7).

Table 6.7: Iteration counts of the GMRES algorithm accelerated by three types of DD preconditioners for various mesh resolutions and numbers of subdomains

Mesh Resolutions	Types of Preconditioners	$n = 2$	$n = 4$	$n = 8$	$n = 16$
36×31	First	12	12	12	13
	Second	12	12	12	14
	Third	11	11	11	11
85×79	First	15	15	15	15
	Second	15	15	16	17
	Third	15	15	16	18
120×111	First	18	18	18	19
	Second	19	19	20	22
	Third	19	19	20	23
150×143	First	21	21	21	23
	Second	23	23	24	27
	Third	24	24	25	27

We observe that the number of iterations increases only very mildly for each fixed-size problem as the number of subdomains increases from two to sixteen. Similar numerical results were reported in [157, 158, 159]. Thus, it is a worthwhile effort to implement these domain decomposition algorithms on such parallel computers as CRAY C90 which has a total maximum number of sixteen processors able to perform in parallel.

However, as pointed out in [106] (see also Section 2.6.2 of Chapter 2 on page 36), larger problems should be solved as more processors become available. In other

words, the problem size should not be fixed, but should scale with the number of processors involved in the actual computation. We believe that more meaningful numerical experiments consist of tests of the convergence rate as a function of the number of subdomains while keeping the problem size fixed in each subdomain (not the entire domain).

In the following two sets of the numerical experiments, we only focus on preconditioners of the third type, since similar results were observed with the other two types of preconditioners. The time-step size used for producing these results (Table 6.8 and 6.9) is $\Delta t = 1000$ s.

We start with a two-subdomain computation with a mesh size of 70×65 in the original domain. Then we refine the mesh by a factor of two in both horizontal and vertical directions. Thus, the discrete problem size after mesh refinement is four times large. To keep the problem size in each subdomain unchanged, we need to further divide each of the previous two subdomains into four pieces, or equivalently, to decompose the original domain into eight subdomains. Numerical results are reported in Table 6.8. We comment that, in the table, the mesh is not exactly refined by a factor of two in the vertical direction, however close. This is due to the fact that we want each subdomain to have exactly the same number of nodes for the eight-subdomain domain decomposition.

Similarly, the mesh refinement by a factor of two along each grid line for a four-subdomain domain decomposition will require sixteen subdomains in order to keep the discrete size of each subdomain problem fixed. The comment given above for Table 6.8 applies also to Table 6.9.

Table 6.8: Iteration counts: two subdomains vs. eight subdomains with a scaled discrete problem size

Mesh resolutions	Number of subdomains	Iterative algorithm	Number of iterations
70×65	2	GMRES	11
		CGS	6
		Bi-CGSTAB	6
140×127	8	GMRES	19
		CGS	13
		Bi-CGSTAB	10

Table 6.9: Iteration counts: four subdomains vs. sixteen subdomains with a scaled discrete problem size

Mesh resolutions	Number of subdomains	Iterative algorithm	Number of iterations
70×63	4	GMRES	11
		CGS	6
		Bi-CGSTAB	6
140×127	16	GMRES	20
		CGS	13
		Bi-CGSTAB	11

6.5 Conclusions

- Many hybrid methods of non-overlapping domain decomposition result from various combinations of linear iterative methods and DD preconditioners (consisting of subdomain solvers and interface preconditioners). When H/h is relatively large, where H characterizes the subdomain length scale and h is the mesh size, inexact subdomain solvers are to be preferred to exact ones for saving CPU time.

- Three types of DD preconditioners were found to work reasonably well with GMRES, CGS and Bi-CGSTAB nonsymmetric iterative methods in the context of solving the shallow water equations, with the newly proposed third type turning out to be computationally the least expensive and the first type most expensive.

- For all cases, GMRES requires roughly twice as many iterations as required by CGS or Bi-CGSTAB. However, these three algorithms were found to be approximately equally efficient in terms of CPU time. Note that only one matrix-vector multiplication per iteration will drive the GMRES algorithm to convergence, while two such multiplications have to be performed at each iteration for CGS and Bi-CGSTAB.

- Bi-CGSTAB algorithm was not found to converge much faster that CGS in this particular application. However, Bi-CGSTAB converges much more smoothly than CGS. GMRES generated the smoothest residual convergence history, due to its minimization requirement of the residual norm at each iteration step.

- As subdomain solvers become more accurate, preconditioners of the third type can accelerate the convergence of GMRES, CGS and Bi-CGSTAB at about

the same rate as the other two types of preconditioners, except for cases when subdomain solvers may be considered to be exact or nearly exact. However, the use of exact or nearly exact subdomain solvers is not recommended since the gain in the number of iterations far from offsets the additional computational cost for constructing them. Thus, we consider the third type of DD preconditioners to be the best.

- Increasing the number of subdomains will result in a deterioration in convergence rate for most iterative domain decomposition algorithms, even though the original problem size is fixed. When these three types of DD preconditioners are extended for use with more than four subdomains, the numbers of iterations will only slightly increase for the fixed-size problems related to the current application.

CHAPTER 7

PARALLEL IMPLEMENTATION ISSUES AND RESULTS

7.1 Introduction

The finite element solution of PDE's, involves the following typical computational stages

- input or preparation of the data required by the code, such as the global numbering of the nodes, the calculation of coordinates of these nodes, etc.;
- construction and representation of the triangulation if triangular elements are used;
- specification of the initial conditions;
- calculation of the element matrices and the corresponding force vectors for each time step;
- assembly of the element matrices and the force vectors to obtain the global (stiffness) matrices and the force vectors for each time step;
- the solution of the global matrix systems at each time step.

Among the aforementioned stages, the last stage, i.e., the solution of resulting algebraic equations, is usually the most computational expensive part of the calculations and, as a result, a sizable amount of research has been focused on developing

efficient and cost effective solvers for large non-linear and linear systems of algebraic equations for over the last thirty years (see [188, 220, 238] and references cited therein).

Since the solution of finite element global matrix systems constitutes the major workload, the parallelization of the solution process is critical. Domain decomposition techniques including Schur domain decomposition method, the modified interface matrix domain decomposition algorithm and the parallel block preconditioning techniques, as developed in previous chapters, essentially reduce the solution of a large linear system into that of solving several disjoint smaller subsystems defined in the subdomains that are amenable to efficient parallelization with relatively little difficulty.

However, to achieve a high parallel efficiency, the parallelization of other stages, especially the assembly of element matrices into a global stiffness matrix, turns out to be equally important. The reason for this is provided by Amdahl's law on the theoretical speed-up for parallel computing (see Chapter 2 and [56]). Amdahl's law points out that even a small percentage of the total work (measured in CPU time) not being processed in parallel will drastically degrade the parallel performance result, namely, the sought-after speed-up, and the situation gets worse when more physical processors are involved. For example, suppose that 20% of the computation is not multitasked, then Amdahl's law predicts that the best speed-up obtainable is as low as 2.5 for four processors, 3.33 for eight processors and only 5 even if an infinite number of processors could be invoked for the calculation. This implies that a good speed-up due to parallelism may not be achieved unless computations related to the finite element discretization are also efficiently parallelized.

The details of implementing the MIMDD algorithm discussed in Chapter 5 on the CRAY Y-MP/432 vector parallel computer and speed-up results were pub-

lished in [174], where macrotasking techniques were employed for large-granularity domain by domain computations and microtasking techniques were exploited for small-granularity element by element calculations.

For the rest of this chapter, we will briefly comment on the relative advantages and disadvantages of three parallel processing software packages currently available on the CRAY Y-MP. Then we describe a multicoloring technique for removing contention delays in the parallel assembly process of the finite elements and provide details concerning the parallel implementation of block preconditioning techniques with the third type of domain decomposed preconditioners (see Chapter 6). Speed-up results for several mesh resolutions are then reported.

7.2 Macrotasking, Microtasking and Autotasking on the CRAY Y-MP

Due to the fact that three possible approaches to parallelization, namely, macrotasking, microtasking and autotasking [56, 57], coexist on the CRAY Y-MP for parallel computations, the first issue to be resolved is to decide, among these possibilities, which parallel software to employ. Below, we provide some general comments based both on our own as well as other researchers' working experience related to multitasking on the CRAY Y-MP, which we hope may serve as a guidance or reference to other CRAY multitasking users.

The earliest implementation of multitasking ideas on CRAY computers was carried out by macrotasking techniques. Macrotasking was designed to efficiently exploit, in a dedicated computing mode, the parallelism at subroutine levels for programs which require large memory and long running time. Macrotasking is implemented by explicitly inserting library calls to multitasking subroutines into the code. However, there are several disadvantages for macrotasking which are listed below

1. Extensive data scope analysis is required. The coding or modification from an existing code into a macrotasked code can be expensive in both time and human effort.

2. Macrotasked programs are hard to test and debug compared with microtasking. The extra effort of applying conditional multitasking techniques is often essential for isolating macrotasking errors.

3. The multitasking overhead associated with macrotasking is high. In the case of relatively small task size, parallelism may not be exploited efficiently by macrotasking.

Microtasking is a much more flexible technique and may be viewed as an improvement over macrotasking. The implementation of microtasking is simply the insertion of preprocessor directives (Fortran comments) into the code for parallelism. The preprocessor will interpret these directives to the compiler and generate the appropriate library subroutine calls. Some advantages of using microtasking are summarized here:

1. Much less data scope analysis is involved for microtasking. The conversion of an existing code to a microtasked code is quite straightforward and takes much less time compared to the case of macrotasking. The converted code is still standard Fortran.

2. The multitasking overhead for microtasking is very small. As a result, small task-size (or granularity) problems, say, a set of nested loops, can be multitasked quite efficiently.

3. Microtasking is able to perform automatic dynamic load balancing for small granularity parallelism (do-loop level parallelism) and hence reduces possible synchronization delays.

4. If the data scope permits, large task-size problems may be multitasked by either microtasking or macrotasking. However, in most cases, microtasking is found to perform as well as or even better than macrotasking (see also the experimental results in [164]).

However, there are also a couple of restrictions for microtasking, which may turn out to be disadvantageous for some applications:

1. Microtasking is not allowed in the main program.

2. Microtasking must extend to the subroutine boundaries.

Fortunately, these disadvantages may be overcome by the combined use of microtasking and autotasking techniques.

Building on the experience gathered from macrotasking and microtasking, autotasking should be preferred to microtasking since (amongst other issues):

1. As its designers pointed out, autotasking combines the best aspects of microtasking.

2. A considerable amount of do-loop level parallelism in the code is automatically detected and exploited by autotasking.

3. Autotasking is allowed in the main program (not for microtasking).

4. Autotasking can exploit parallelism at the do-loop level without extending to subroutine boundaries (microtasking can't).

5. Large-granularity parallelism can also be efficiently exploited by autotasking by manually building in the `case/end case` structure (equivalent to `process/also process/end process` structure for microtasking) within a parallel region.

6. The ATEXPERT post-processing tool [55] has been developed for accurately predicting and graphically displaying autotasking performance in a dedicated system, based on the execution results of a single run on an arbitrarily loaded, non-dedicated CRI (CRAY Research Inc.) system.

However, it should be cautioned at this point that we do not exclude the possibility of better exploitation of multitasking capabilities on CRAY by appropriately applying autotasking, microtasking and macrotasking on the same code to adapt to some particular applications. However, extensive data scoping, coding, debugging and numerical experiments in a dedicated computing environment may be required for this purpose.

7.3 A Multicolor Numbering Scheme for Removing Contention Delays in the Parallel Assembly of Finite Elements

In general, the quality of parallelization on CRAY Y-MP and other shared memory parallel computers is usually determined by the following:

- Level (granularity) of parallelism exploited.
- Frequency of calls to the multitasking library.
- The memory contention delays.
- Load balancing.

The noteworthy memory contention delays for the finite element computation are due to the assembly process. However, the possibility of memory contention delays may be removed by assembling the element properties in a particular order according to a multicolor numbering scheme to be discussed below. Hence, a large number of calls to the multitasking subroutine that performs locking operations is avoided and, as a result, a considerable amount of overhead is removed.

As is well known, for the finite element discretization, the setup of local stiffness matrices and their assembly into a single global stiffness matrix is the only part of the calculations that requires being repeatedly carried out as the computation proceeds in time. The efficient parallelization of this part is thus highly critical to the overall parallel performance. For the serial computation, the element stiffness matrices are first calculated and then immediately distributed into appropriate locations in the global stiffness matrix. The process continues element by element. Since each element makes its own independent contribution, this element by element computation can be carried out concurrently as long as the simultaneous alteration to any entry in the global matrix is guarded against.

In the jargon of parallel computing, there exists a critical region in the code segment for the aforementioned element by element calculations. A critical region is defined as a segment of code that accesses a shared resource, e.g., the memory. An interprocess mechanism, namely, the lock, is required and provided by shared memory multiprocessors for the sake of synchronization. However, the locking operation introduces three adverse effects, listed below, which tend to degrade parallel performance results:

- The computation in the critical region is serial;

- Other processors have to wait and stay idle while one processor is working within the critical region;
- Invoking the lock multitasking subroutine consumes a certain amount of CPU time, i.e., the overhead.

Within the critical region, an entry in a local stiffness matrix is added to a corresponding entry in the global stiffness matrix and then the newly obtained entry is inserted back into the memory space previously occupied by the entry in the global matrix prior to modification. The operations involved here constitute a tiny part of the calculation and seem to be negligible. The waiting time for other processors to enter the critical region is also very small, as argued in [165], when the cost of generating the element matrices is greater than that of the assembly into the global matrix. However, the overhead introduced by ne (total number of elements) calls (if assembled by elements) or n (total number of nodes) calls (if assembled by nodes) to the multitasking subroutine generating the locking mechanism is not so small, especially for fine meshes, and the combined effect of these three possible sources causing parallel performance degradation can not be neglected.

As was pointed out in [85] (see also some references therein), the critical region in the assembly process is not inherent in the element-by-element calculations and may be bypassed by assembling the element matrices in a particular order. The basic observation is that the necessity of introducing a critical region into the assembly process is due to the possible simultaneous contributions to a common node by more than one of its surrounding elements. The critical region may be removed if the assembly process can be carried out in groups so that, within each group, no two or more elements connected to a common node are able to make contributions

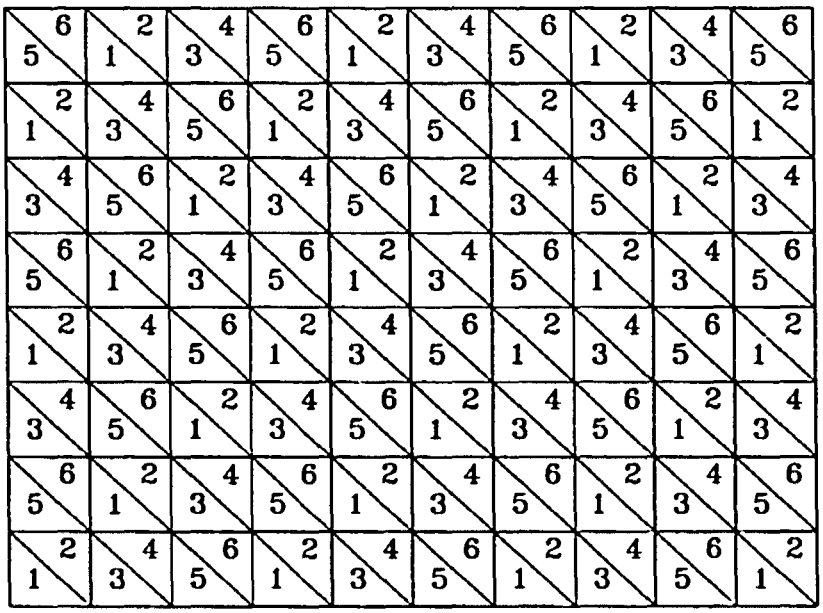


Figure 7.1: Multicolor numbering of elements for a triangular finite element mesh. Each integer stands for a unique chosen color. A node in the mesh is surrounded by elements of different colors.

to that node. This idea may be realized by using a multicolor numbering of the elements to be assembled.

For the triangular linear element mesh used in the current problem [169], six colors are required to guarantee that any node in the physical domain is surrounded by elements of different colors. The ideas are well expressed by Figure 7.1, where each integer represents a unique color. The elements in the mesh may now be divided into six groups. Elements of the same color comprise one group and different groups have different colors. The assembly is carried out one group after the other. Within each group, the elements are internally disjoint and so the parallel assembly process is carried out asynchronously. Notice that the multicoloring scheme achieves not only the removal of critical regions, but also a sharp reduction in the number of synchronization points from nc or n to 6 for the entire assembly process.

7.4 Implementation Details and Results

7.4.1 Parallelization of Subdomain by Subdomain and Element by Element Calculations

The domain decomposition code corresponding to the use of the third type of domain decomposed preconditioners was carefully tuned and tested on the four-processor CRAY Y-MP/432 for a number of mesh resolutions. Autotasking parallel software capabilities on the CRAY were employed exclusively for the reasons listed in Section 7.2, although macrotasking and microtasking were exploited in out earlier work [174].

The main computational work for those three non-symmetric iterative algorithms, namely, GMRES, CGS and Bi-CGSTAB, is associated with matrix-vector products, which are obtained through approximately solving the problem defined

on each subdomain. The parallelization of these tasks is carried out at subroutine levels. Since we do not find sizable differences between the speed-up results obtained by the aforementioned three iterative solvers, the parallel performance results to be reported in what follows correspond to the Bi-CGSTAB iterative method.

As pointed out in Chapter 2, the load balancing is one of the most important factors determining parallel efficiency. To ensure a good load balancing, the original domain for our problem is divided into regular subdomains with the same numbers of nodes. Domain decomposition offers the opportunity to carry out subdomain by subdomain calculations, which may be parallelized at the subroutine level using `case/end case` autotasking directives. This pair of directives serve as a separator between adjacent code blocks that are concurrently executable and may only appear in a parallel region. To this end, we need to mark the start and end of a parallel region (including redundant and partitioned code segments) by using another set of autotasking directives `parallel/end parallel`. At the beginning of this parallel region, we have to specify two types of variables, namely, shared and private. The former data items are known to all processors, while the latter data items are only visible to each processor, which makes it possible for the same subroutine to operate on different sets of data simultaneously. A typical segment of code involving the use of `case/end case` directives appears as follows

```
cmic$ parallel shared(p,t1,t2,t3,t4,l1,l2,l3,l4,l5,
cmic$+                ic1,ic2,ic3,q1,q2,q3,q4,
cmic$+                decom1,decom2,decom3,decom4,
cmic$+                loca1,loca2,loca3,loca4,loca5)
cmic$+         private(p1,i,in,,w1,w2)
cmic$+         maxcpus(4)
```

```
*
*     A segment of code illustrating the use of
*     case/end case directives.
*
cmic$ case
      w1=timef()
      do 10 i=1,l1
        p1(i)=p(i)
10    continue
      call prec(l1,decom1,loca1,p1)
      call prod51(l5,l1,p1,t1)
      call prod11(l1,l1,p1,q1)
      w2=timef()
      print*,'Wall-clock_1 ',.001*(w2-w1)
cmic$ case
      w1=timef()
      do 20 i=1,l2
        in=i+ic1
        p1(i)=p(in)
20    continue
      call prec(l2,decom2,loca2,p1)
      call prod52(l5,l2,p1,t2)
      call prod22(l2,l2,p1,q2)
      w2=timef()
      print*,'Wall-clock_2 ',.001*(w2-w1)
cmic$ case
```

```
      w1=timef()
      do 30 i=1,l3
        in=i+ic2
        p1(i)=p(in)
30    continue
      call prec(l3,decom3,loca3,p1)
      call prod53(l5,l3,p1,t3)
      call prod33(l3,l3,p1,q3)
      w2=timef()
      print*,'Wall-clock_3 ',.001*(w2-w1)
cmic$ case
      w1=timef()
      do 40 i=1,l4
        in=i+ic3
        p1(i)=p(in)
40    continue
      call prec(l4,decom4,loca4,p1)
      call prod54(l5,l4,p1,t4)
      call prod44(l4,l4,p1,q4)
      w2=timef()
      print*,'Wall-clock_4 ',.001*(w2-w1)
cmic$ end case
cmic$ end parallel
```

Within the parallel region, the case/end case directives create four processes (including an original master or parent process and three slave or child processes),

each of which is making three consecutive subroutine calls independent of other three coexistent processes. Note that these processes can call the same subroutine `prec` simultaneously. With given shared data items, the subroutine `prec` will perform the required operations and return with an array `p1`, the private data items, which are stored on a separate stack for each processor that executes the subroutine. However, if some shared variables need to be modified within a parallel region, the directive pair `guard/end guard`, an interprocess synchronization mechanism (the lock) provided by all shared memory parallel computers with different forms, must be invoked in order to prevent the same memory location to be accessed by more than one process at a time.

Corresponding to the finite element discretization is the element by element calculation, which may be parallelized at the do-loop level. This is done by building the `do parallel` directives into a parallel region confined by a `parallel/end parallel` directive pair or simply using the `do all` directive. The work distribution policy for a parallel loop is either to specify the number of chunks or to employ the stripmining technique, depending on the situation. However, the general rule is to vectorize the innermost loop and to multitask the outer loop in a nested set of loops or to split a single loop into inner and outer loops, and then vectorize the inner loop and multitask the outer one. It should be emphasized that the multicoloring scheme described in Section 7.3 avoids the possibility of entries in the global matrix being simultaneously updated by several processes, as a result, no `guard/end guard` directives are required in the assembly process.

In addition, many recurrence relations which appeared in the original single domain finite element code (see [170]) are not intrinsic and have been removed. For example, the global nodal numbers are fully determined by the local element nodal numbers, once a particular element has been selected. Hence the relations between

the global numbering of the nodes, local numbering of the element vertices and element numbering can be calculated independent of the calculations made for the previous elements.

7.4.2 Comments on the Speed-Up and Results

Before presenting the speed-up results, a word about speed-up seems necessary. The term speed-up is sometimes misleading. Speed-up may be defined to be the ratio of the wall clock time elapsed in a dedicated mode of computation, for computing the same problem, using the best serial algorithm and the parallel algorithm. However, the optimal serial algorithm is usually unknown, especially for modeling a meaningful physical process. In our problem, the use of GMRES, CGS or Bi-CGSTAB preconditioned by ILU in the original domain consumes more CPU time than the use of the same iterative method and preconditioner treated in a domain decomposed way. In other words, apart from the parallelization issues, consideration of computational complexity alone justifies the use of domain decomposition for some problems.

Since the best serial treatment of the present problem can not be determined, the speed-up results reported here refer to measurements of the wall clock time in a dedicated computing environment relative to the uni- and multi-processor implementation of the same domain decomposition algorithm (see also, among others, [103, 132, 165]). This definition properly incorporates communication overhead and synchronization delays and shows how well the domain-to-processor mappings are done. However, it should be borne in mind that this definition has a serious drawback. Following this definition, a parallel algorithm achieving a perfect speed-up may actually take longer time to execute than a serial algorithm for solving the same problem.

We integrated the finite element model of the shallow water equations for four different mesh resolutions, namely, 19×15, 34×27, 49×43 and 64×55 for a period of five hours with corresponding time step sizes of $\Delta t = 1800$ s, 1000 s, 600 s and 400 s, respectively. The experimental results are summarized in Table 7.1. To appreciate the speed-up result of 3.6 corresponding to the mesh resolution 64×55, we point out that, by Amdahl's law, this speed-up means that 96% — 97% of the total computational work (measured in CPU time) is parallelized on a four-processor machine.

Table 7.1: Parallel performance results for four different mesh resolutions using the Bi-CGSTAB algorithm preconditioned by the third type of domain decomposed preconditioners on the four-processor CRAY Y-MP/432

Mesh resolutions	19×15	34×27	49×43	64×55
Serial seconds	1.03	6.26	35.19	108.07
Parallel seconds	0.38	2.03	10.29	29.77
Speed-up ratios	2.7	3.1	3.4	3.6

It should also be pointed out that the automatic do-loop level parallelization as detected and exploited by the autotasking preprocessor (FPP) does not yield a speed-up larger than two. The reasons are the following

- automatic autotasking works best when most of the work in a code is in nested do-loops which do not contain call statements;
- autotasking is unable to detect parallelism across subroutine boundaries.

It is thus clear that, to exploit subroutine level parallelism as offered by domain decomposition algorithms, one has to manually insert appropriate autotasking directives into the code. In general, to achieve a high efficiency of parallelization with

autotasking utilities, a fair amount of user assistance is still absolutely required, while an analysis generated automatically by FPP can serve as a reference for the design of a parallel code.

To conclude this section, we mention that the FEUDX finite element code [169, 170] was ported to MasPar, another parallel machine architecture (see [180]).

7.5 Conclusions

- Among three parallel processing software packages coexistent on the CRAY Y-MP/432, autotasking techniques seem to be the best for implementing domain decomposition algorithms, where the number of child processes to be created is known in advance. Autotasking is able to efficiently exploit both small and large granularity parallelism. Microtasking offers another good choice, but is not as flexible as autotasking in programming.

- Autotasking, as its name suggests, can be fully automatic. However, a fair amount of user assistance is required to achieve better performance. It is mandatory to tell autotasking that several pieces of work (defined on different subdomains, say) to be performed in subroutines can be carried out in parallel by using the `case/end case` directive pairs. By appropriately distinguishing and defining shared and private variables at the beginning of the parallel region, the same subroutine may be reused and accessed simultaneously by different processes (a property known as reentrancy or multithreading). In fact, this should be the case for a parallel algorithm in which each processing element performs all the operations on the partitioned data rather than the partitioned operations on the whole set of data.

- For the finite element numerical solution of PDE's, it is important to parallelize efficiently the assembly process in order to achieve a better speed-up result. The critical regions in the assembly process may be removed and the number of synchronization points drastically reduced by assembling the elemental contributions in a specific order as specified by a multicolor numbering scheme.

CHAPTER 8

SUMMARY, CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

We have introduced the following six domain decomposition methods in the previous chapters

- the multiplicative Schwarz overlapping domain decomposition method;
- the additive Schwarz overlapping domain decomposition method;
- the iteration-by-subdomain nonoverlapping domain decomposition method;
- the Schur nonoverlapping domain decomposition method;
- the modified interface matrix nonoverlapping domain decomposition method;
- the parallel block preconditioning techniques.

Of course, there are still other domain decomposition methods. By modifying one or more ingredients of these general methods, there exist almost infinitely many variants of the so-called iterative domain decomposition algorithms. The consideration of the implementation details of these domain decomposition algorithms would add another dimension to the variety.

In this dissertation, we have first closely examined the Schur domain decomposition method and its application to the finite element numerical simulation of the

shallow water flow. Our motivation for studying domain decomposition algorithms is purely due to their potential for parallelism.

To introduce parallelism into the linear systems of algebraic equations at each time step corresponding to a spatial finite element and an implicit temporal discretizations, we partitioned the whole computational domain (an approximation to the original physical domain) into subdomains with the artificially introduced interfaces. After identifying two types of nodal variables, namely, internal and interfacial variables, and numbering the global nodes in a substructured way, we are able to arrive at a much smaller Schur complement linear system of equations involving the interfacial variables only. Obviously, the discrete subdomain problems are trivially decoupled and the internal nodal values can be determined independently of each other once the interfacial degrees of freedom are specified by solving the Schur complement linear system at each time step.

Thus, the efficient solution of the Schur complement linear system is a key issue for the Schur domain decomposition method. A preconditioned Krylov or conjugate gradient-like iterative method is almost always employed for the iterative solution of this linear system. The multiplications of the Schur complement matrix by a vector at each iteration step are obtained by concurrently solving all the discrete subdomain problems. If the convergence criterion is not satisfied on the interfaces, the domain is decomposed again and subdomain problems are solved. This may be described as a divide and feedback process. This process immediately indicates that the determining factors for the efficiency of the Schur domain decomposition method are (1) the number of iterations required for achieving convergence on the interfaces and (2) the cost of subdomain solvers. As is well known, the number of iterations required on the interfaces can be reduced by using an appropriate interface preconditioner.

To construct a good interface preconditioner, we first note that, corresponding to many elliptic PDE's, the action of the Schur complement matrix operator is predominantly local and, fortunately, this remains to be the case in the context of the finite element shallow water flow modeling. This clearly suggests the use of an interface probing preconditioner, which, being purely algebraic, possess a wide applicability. However, we have shown that a modified version of this popular rowsum preserving interface probing preconditioner behaves better for the current application.

For many specific application problems, the unavailability of fast direct solvers in the subdomains is common. To mitigate this potential disadvantage of the Schur domain decomposition method, we propose a new algorithm, namely, the modified interface matrix domain decomposition (MIMDD) algorithm. In contrast with the Schur domain decomposition approach, in which the numerical solution on the interfaces is first determined by solving the Schur complement linear system, the MIMDD algorithm starts with an initial guess on the interfaces and then iterates back and forth between the subdomains and the interfaces until a convergence criterion is satisfied on the interfaces. Beginning from the second outer iteration step, it becomes increasingly less expensive to obtain solutions on the subdomains and the interfaces due to the availability of successively improved initial solutions from the previous outer iteration. The results obtained by applying this algorithm to our application improve upon those obtained by employing the traditional Schur domain decomposition algorithm.

The efficiency of the MIMDD algorithm is determined by (1) how small the spectral radius $\rho[I_{ss}-(A_{ss}+K_{ss})^{-1}C]$ is and (2) the amount of computational work required for obtaining the multiplication of the matrix K_{ss} by a vector. Numerical results confirm that, by using MILU, $\rho[I_{ss}-(A_{ss}+K_{ss})^{-1}C]$ can be made as small as

$O(10^{-3})$ without much extra computational work. Moreover, the MIMDD algorithm with $A_{ss}+K_{ss}$ constructed this way has good self-adaptivity to mesh refinement. On the other hand, the availability of good initial subdomain solutions from the previous outer iteration will also make the coarse grid correction multigrid subdomain solvers more efficient.

The block preconditioning method, an alternative domain decomposition approach, consists of the construction of a domain decomposed preconditioner such that approximate solutions in the subdomains and on the interfaces can be simultaneously updated at the cost of only inexact subdomain solves. It should be pointed out that both the block preconditioning and MIMDD methods abandon the idea of obtaining the interfacial solutions first for decoupling the subdomain problems, as in the Schur domain decomposition method.

The discrete nonsymmetric linear systems ($Ax = b$) at each time step of the model integration for the geopotential and velocity fields are solved by a Krylov or conjugate-gradient like iterative method. Here, we have employed three popular and competitive iterative methods, namely, GMRES, CGS and Bi-CGSTAB. We expect that these three algorithms will be more extensively studied and compared by numerical analysts and be widely applied to many other related computational problems in science and engineering. Although a thorough analysis and comparison of these three important solvers are beyond the scope of this dissertation, we have essentially paid much attention to their relative efficiencies as applied to the finite element shallow water flow problem. Extensive conclusions have been provided at the end of Chapter 6.

The ultimate efficiency for any Krylov iterative method is largely determined by a preconditioner B. With this preconditioner, instead of solving $Ax = b$, we solve $\hat{A}\hat{x} = b$, where $\hat{A} = AB^{-1}$ and $\hat{x} = Bx$, for right preconditioning. This

preconditioner should be such that (1) the condition number of AB^{-1} is much smaller than that of A; (2) the preconditioning matrix is computationally cheaper to invert and (3) the solution of the preconditioning linear system $Bp = q$ is parallelizable.

In Chapter 6, three types of domain decomposed preconditioners have been described in detail and applied to the parallel finite element solution of the shallow water equations. Since domain decomposed preconditioners have good parallelization properties, most of the further research effort for this type of algorithms is aimed at reducing computational complexity instead of achieving parallelism. The third type of domain decomposed preconditioners proposed here turns out to be computationally cheaper than the other two types. We also point out that, for a linear system in which the coefficient matrix is symmetric, corresponding to, for example, the discretization of the self-adjoint elliptic problems, one has to use the first type of domain decomposed preconditioners to preserve symmetry. For the non-symmetric case, even though all three types of domain decomposed preconditioners may be applied, it turns out that the first type of domain decomposed preconditioners is computationally much more expensive to use than the other two types in our applications.

As a rule, the convergence rate of many iterative domain decomposition algorithms will decrease, even though the problem size is fixed, as the number of subdomains is increased. Consequently, it is important to devise algorithms whose convergence rate has little or no dependency on the number of subdomains. Otherwise, the increased computational complexity will partially undo the gains obtained by parallel processing. Our numerical results have shown that, when the number of subdomains increases from two to sixteen, the numbers of iterations for using these three types of domain decomposed preconditioners, as applied to the finite element shallow water flow problem, just mildly increase. As a result, we expect a

good parallelization efficiency when the codes are ported to the CRAY C90, which has a total number of sixteen powerful processors able to work simultaneously on solving the same problem. Of course, as more processors become available, the goal is not to solve the problem with a fixed mesh resolution, but rather to deal with discrete problems corresponding to much higher mesh resolutions. To minimize the loss of gains due to parallel processing, the algorithm must be designed to be less dependent not only on the subdomain size H, but also on the mesh size h.

Domain decomposition offers an opportunity for carrying out subdomain by subdomain calculations, which can be parallelized quite efficiently at the subroutine level on the shared memory parallel computers or can be accommodated in separate parallel processing elements (local processor and memory) on a distributed memory parallel computer. However, the efficient parallelization of the finite element discretization process is also very important for ensuring a high overall efficiency of parallelism. The multicolor numbering scheme described in Chapter 7 for the parallel assembly of elemental contributions removes not only the critical regions inside the assembly process, but also minimizes the number of the synchronization points.

A possible future research direction consists in applying domain decomposition and parallel computing techniques to the 4-D variational data assimilation problem in meteorology. This requires us to properly define a cost functional J consisting of a weighted lack of fit between the model and observations. The adjoint model techniques are used to obtain the gradient of this cost functional with respect to the initial conditions and/or boundary conditions (see [178, 242, 243] and references therein).

For parallel data assimilation, a parallel numerical optimization (descent) algorithm is the overall framework. Domain decomposition, adjoint algorithms (par-

allelized) and others are built into this framework to efficiently provide, at each minimizing step, the solution of the model and its gradient (so as to form a descent direction) allowing the approximation to the minimizer to be updated. This process is carried out iteratively until convergence (i.e., until we find an accurate enough approximation to the minimizer). We will devise and test different domain decomposition algorithms which can be efficiently built into this framework.

We know that, in typical domain decomposition algorithms, the amount of information in one subdomain (processor) which is required for computation in another subdomain (processor) is very small compared with the amount of data residing locally in each subdomain (processor). This small amount of information is associated with grid points on or near the interfaces. Moreover, in the case of a large number of subdomains, the number of subdomains (processors) with direct data dependencies is small compared with the total number of subdomains (processors). Thus, domain-based decomposition algorithms potentially lead to a high ratio of time spent on computation versus that spent on communication & synchronization and, as a result, a high parallelization efficiency is expected for implementation on a large number of processors or for a massively parallel processing architecture.

However, as we pointed out earlier, one of the problems left is that domain decomposition algorithms suffer from a possible deterioration in convergence rate as the number of subdomains increases, since the exchange of information between remotely located subdomains is slow. If this problem is not fixed, the increased serial complexity of the algorithm will probably undo the gain of parallel processing. A possible way to improve scalability of domain decomposition algorithms to a large number of processors is by introducing a global coarse grid, which serves as a mechanism to accelerate transfer of information among subdomains. In the

future, we shall closely investigate this type of algorithms and carry out practical applications.

Finally, we will also try to extend and develop domain decomposition algorithms for application to problems involving three dimensional space domains. This is a much more challenging problem than its two dimensional counterpart.

APPENDIX A

THE FINITE ELEMENT SOLUTION OF THE SHALLOW WATER EQUATIONS

Similar to substructuring domain decomposition methods, the finite element method originated from matrix structural analysis. Although the terminology "finite element method" was coined in 1960 by Clough [50], the basic idea behind the method is not new and can be traced back to the work of Courant [54] in 1943 who used the approximation of a function in R^2 by continuous piecewise linear functions on a triangulation and the principle of minimum potential energy to study the Saint Venant torsion problem, which was explained earlier in chapter 3.

There was an explosion of research activities in the area of the finite element method beginning approximately from the 1960s. The method was first widely studied and systematically applied to the solution of solid mechanics problems in the 1960s. Mainly due to the efforts of three groups of researchers, namely, mathematicians, physicists and engineers, there have been rapid, although sporadic, developments and breakthroughs in the field of finite element methods. An excellent paper which reviews the development of the finite element method up to 1980 is available [51]. We also refer interested readers to [244] for motivations, general principles and a good exposition of several different approaches as well as [224] for a more recent comprehensive review in the context of solving PDEs by the finite element method.

Today the finite element method is a dominant numerical technique in solid mechanics, structural engineering, aeronautical engineering and many others. However, a sizable amount of research still continues in this area for designing and analyzing accurate and cost effective algorithms for CFD problems, especially for compressible flows (see [163] for the most recent state of the art) and in the area of adaptive h-p mesh refinement.

In this appendix we describe the finite element modeling of the 2-D shallow water flow in a limited-area domain defined by (4.42) — (4.46) in Chapter 4. The essential components of a two stage Numerov-Galerkin finite element method (see [174] and references therein) will also be summarized.

A.1 The Finite Element Approximation

For the finite element method, instead of working directly on the originally given PDEs (the strong statement of the problem), one derives a weak or variational form, equivalent to the original PDEs, involving two Sobolev function spaces, namely, the trial solution space of functions V and the weight or test function space W. The former is usually a function space whose member satisfies the essential boundary conditions (in contrast with the so-called natural boundary conditions) of the given problem. The latter is usually a function space whose member satisfies the homogeneous essential boundary conditions, see [123] and references therein.

As a first step in developing any finite element algorithm, one constructs finite dimensional function spaces V^h and W^h which approximate V and W, respectively. Specifically, in the finite element method, an n-dimensional approximate trial function space is

$$V_n^h = g^h + \text{span}\{N_1, \ldots, N_n\} \tag{A.1}$$

where the function g^h is incorporated into V_n^h space to account for non-homogeneous essential boundary conditions (see [123]). An n-dimensional test function space may be defined as

$$W_n^h = \text{span}\{\bar{N}_1, \bar{N}_2, \ldots, \bar{N}_n\}. \tag{A.2}$$

For the Bubnov-Galerkin or simply the Galerkin finite element method employed in this application, $\bar{N}_i = N_i$, $i = 1, 2, \ldots, n$, while for Petrov-Galerkin finite element method (see [86] and references therein), $\bar{N}_i \neq N_i$. N_i and $\bar{N}_i$, $i = 1, 2, \ldots, n$, are usually piecewise smooth functions with only local supports. Systems of linear or nonlinear algebraic equations will be obtained upon applying either Galerkin or Petrov-Galerkin formulation. The solution to the weak form of the original PDEs is approximated by functions, which are locally smooth but globally "rough", in the finite dimensional finite element space V_n^h.

The shallow water equations model (4.42) is approximated in this application by the Galerkin finite element method with a piecewise linear triangular finite element space. Since the highest derivative in each of the shallow water equations is of order one, the linear function space approximation will guarantee the necessary continuity and completeness requirements (see [246]).

Suppose the problem domain Ω has been partitioned into E small elements Ω_e, $e = 1, \ldots, E$, and total number of grid nodes is M_{nd}. Each of the three dependent variables φ, u and v is approximated over the domain Ω by a linear combination of the basis functions $N_m(x, y)$, $m = 1, 2, \ldots, M_{nd}$, in the finite element space V^h, namely

$$\varphi \approx \hat{\varphi} = \sum_{m=1}^{M_{nd}} \varphi_m(t) N_m(x, y) \tag{A.3}$$

$$u \approx \hat{u} = \sum_{m=1}^{M_{nd}} u_m(t) N_m(x, y) \tag{A.4}$$

$$v \approx \hat{v} = \sum_{m=1}^{M_{nd}} v_m(t) N_m(x,y) \tag{A.5}$$

where $\varphi_m(t)$, $u_m(t)$, $v_m(t)$ are the unknown nodal values of φ, u and v respectively, $N_m(x,y), m = 1, \ldots, M_{nd}$, are the global shape functions defined in such a way that the following properties are satisfied:

- linear in x and y;
- $N_i(x_j, y_j) = \delta_{ij}$;

where δ_{ij} is the Kronecker delta, i.e., $\delta_{ij} = 1$ if $i = j$, whereas $\delta_{ij} = 0$ if $i \neq j$. In words, $N_i(x,y)$ takes on the value 1 at node (x_i, y_i) and 0 at all other nodes (see Figure A.1)

From the element point of view, the global shape functions can also be equivalently defined in terms of the local shape functions $N_i^e(x,y)$, namely,

$$N_i(x,y) = \begin{cases} N_i^e(x,y) & \text{on } \Omega_e's \\ 0 & \text{on } \Omega - \Omega_e's \end{cases} \tag{A.6}$$

where Ω_e's are those triangular elements which share a common node i. $N_i^e(x,y)$ is the local element shape function defined on the element Ω_e only. It is linear in x and y and has the value unity at node i and zeros at nodes j and k, where i, j, k are three vertices of the triangle and are usually numbered counter-clockwise.

The Galerkin finite element approximation over the whole domain Ω requires that:

$$\int\int_\Omega \left(\frac{\partial \hat{u}}{\partial t} + \hat{u}\frac{\partial \hat{u}}{\partial x} + \hat{v}\frac{\partial \hat{u}}{\partial y} + \frac{\partial \hat{\varphi}}{\partial x} - f\hat{v} \right) N_l dx dy = 0 \tag{A.7}$$

$$\int\int_\Omega \left(\frac{\partial \hat{v}}{\partial t} + \hat{u}\frac{\partial \hat{v}}{\partial x} + \hat{v}\frac{\partial \hat{v}}{\partial y} + \frac{\partial \hat{\varphi}}{\partial y} + f\hat{u} \right) N_l dx dy = 0 \tag{A.8}$$

$$\int\int_\Omega \left(\frac{\partial \hat{\varphi}}{\partial t} + \frac{\partial(\hat{\varphi}\hat{u})}{\partial x} + \frac{\partial(\hat{\varphi}\hat{v})}{\partial y} \right) N_l dx dy = 0 \tag{A.9}$$

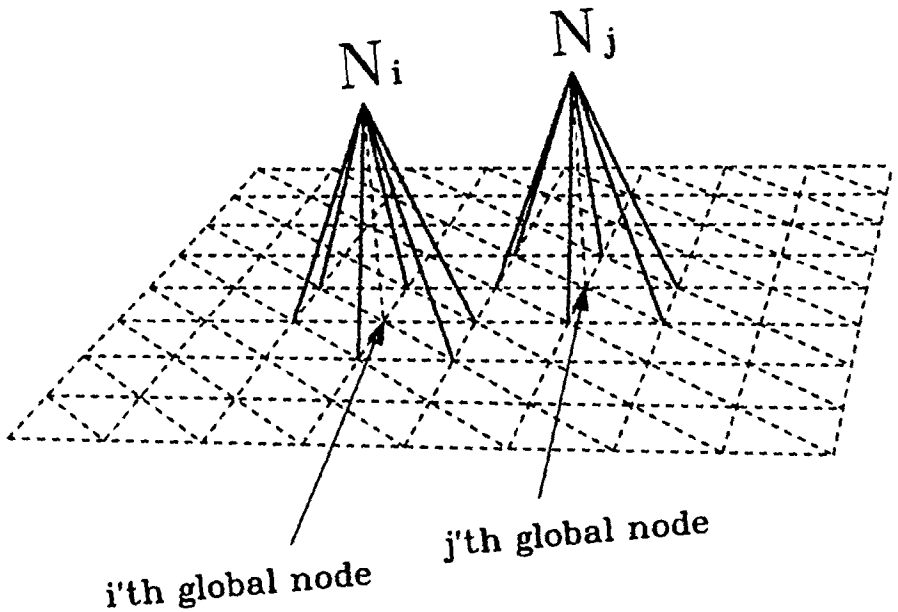


Figure A.1: The global linear shape functions on a triangular element mesh.

where $l = 1, \ldots, M_{nd}$. By using Green's Theorem and taking into account the periodic boundary conditions in the x direction and the boundary conditions on v, the equation (A.9) can be shown to be equivalent to:

$$\int\int_{\Omega} \left(\frac{\partial \hat{\varphi}}{\partial t} N_l - \hat{\varphi}\hat{u} \frac{\partial N_l}{\partial x} - \hat{\varphi}\hat{v} \frac{\partial N_l}{\partial y} \right) dxdy = 0 \tag{A.10}$$

By substituting equations (A.3), (A.4) and (A.5) into (A.10), (A.7) and (A.8), the continuity and momentum equations in x and y directions become respectively:

$$M\dot{\varphi} - K_1\varphi = 0 \tag{A.11}$$

$$M\dot{u} + K_2 u + K_3\varphi - K_4 v = 0 \tag{A.12}$$

$$M\dot{v} + K_2 v + K_5\varphi + K_4 u = 0 \tag{A.13}$$

where

$$\varphi = (\varphi_1, \varphi_2, \ldots, \varphi_{M_{nd}})^T \tag{A.14}$$

$$u = (u_1, u_2, \ldots, u_{M_{nd}})^T \tag{A.15}$$

$$v = (v_1, v_2, \ldots, v_{M_{nd}})^T \tag{A.16}$$

M and $K_1, \ldots, K_5$ are $M_{nd} \times M_{nd}$ matrices and their typical elements are given by the following formulas:

$$M_{lm} = \int\int_{\Omega} N_l N_m dxdy \tag{A.17}$$

$$(K_1)_{lm} = \sum_{k=1}^{M_{nd}} \int\int_{\Omega} \left(\frac{\partial N_l}{\partial x} u_k N_k N_m + \frac{\partial N_l}{\partial y} v_k N_k N_m \right) dxdy \tag{A.18}$$

$$(K_2)_{lm} = \sum_{k=1}^{M_{nd}} \int\int_{\Omega} \left(N_l u_k N_k \frac{\partial N_m}{\partial x} + N_l v_k N_k \frac{\partial N_m}{\partial y} \right) dxdy \tag{A.19}$$

$$(K_3)_{lm} = \int\int_{\Omega} N_l \frac{\partial N_m}{\partial x} dxdy \tag{A.20}$$

$$(K_4)_{lm} = \int\int_{\Omega} f N_l N_m dxdy \tag{A.21}$$

$$(K_5)_{lm} = \int\int_{\Omega} N_l \frac{\partial N_m}{\partial y} dxdy \tag{A.22}$$

The definite integral $\int\int_{\Omega}(\ldots)dxdy$ above can be evaluated by summing the contributions from the individual elements. For instance,

$$M_{lm} = \sum_{e=1}^{E} M_{lm}^{e} \tag{A.23}$$

where

$$M_{lm}^{e} = \int\int_{\Omega_e} N_l^e N_m^e dxdy \tag{A.24}$$

The expression for a typical entry in the e-th element matrix is given by (A.24). In the finite element method, once the element matrix M^e is determined for a typical element, the global system matrix M can be obtained by systematically assembling (adding) non-zero entries of M^e into appropriate entries of the matrix M. This is known as the assembly process or, in structural engineering, the direct stiffness method [226]. It is much more economical to deal with a local element matrix M_{local}^{e}, whose size is only 3×3 in this case, which contains only non-zero entries in the matrix M^e. The assembly of M_{local}^{e} into the matrix M is accomplished through a well defined correspondence among three sets of numbering for the global nodes, local nodes and elements, respectively (see [245], for example).

A.2 Time Integration

The θ scheme (see, for example, [122]) will be used to integrate the equations (A.11), (A.12) and (A.13) in time. To this end, let us introduce a parameter θ such that $t_\theta = t_n + \theta\Delta t$, where $0 \leq \theta \leq 1$ and $n = 0, 1, \ldots$, we can write equation (A.11) as

$$M\dot{\varphi}^{\theta} - K_1^{\theta}\varphi^{\theta} = 0 \tag{A.25}$$

where the superscript θ indicates the time-dependent nodal vectors φ, u, v evaluated at t_θ and the $\dot{\varphi}^\theta$ and φ^θ are, respectively,

$$\dot{\varphi}^\theta = \frac{\varphi^{n+1} - \varphi^n}{\Delta t} \tag{A.26}$$

$$\varphi^\theta = (1-\theta)\varphi^n + \theta\varphi^{n+1} \tag{A.27}$$

Here we take $\theta = 1/2$ which is usually called the Crank-Nicolson scheme and use the following second-order approximation in time to quasi-linearize the advective terms in the shallow water equations

$$u^{n+1/2} = u^* = (3/2)u^n - (1/2)u^{n-1} + O(\Delta t^2) \tag{A.28}$$

$$v^{n+1/2} = v^* = (3/2)v^n - (1/2)v^{n-1} + O(\Delta t^2) \tag{A.29}$$

The continuity equation becomes, upon substituting equations (A.26), (A.27), (A.28) and (A.29) into (A.25),

$$M(\varphi^{n+1} - \varphi^n) - \frac{\Delta t}{2}\overline{K}_1(\varphi^{n+1} + \varphi^n) = 0 \tag{A.30}$$

where

$$(\overline{K}_1)_{lm} = \sum_{k=1}^{M} \int\int_\Omega \left(\frac{\partial N_l}{\partial x} u_k^* N_k N_m + \frac{\partial N_l}{\partial y} v_k^* N_k N_m \right) dxdy \tag{A.31}$$

By a similar treatment, the momentum equation in the x direction becomes:

$$M(u^{n+1} - u^n) + \frac{\Delta t}{2}\overline{K}_2(u^{n+1} + u^n) + \frac{\Delta t}{2}K_3(\varphi^{n+1} + \varphi^n) - \Delta t K_4 v^* = 0 \tag{A.32}$$

where

$$(\overline{K}_2)_{lm} = \sum_{k=1}^{M} \int\int_\Omega \left(N_l u_k^* N_k \frac{\partial N_m}{\partial x} + N_l v_k^* N_k \frac{\partial N_m}{\partial y} \right) dxdy \tag{A.33}$$

The momentum equation in the y direction is:

$$M(v^{n+1} - v^n) + \frac{\Delta t}{2}\overline{K}_3(v^{n+1} + v^n) + \frac{\Delta t}{2}K_5(\varphi^{n+1} + \varphi^n) + \frac{\Delta t}{2}K_4(u^{n+1} + u^n) = 0 \tag{A.34}$$

where

$$(\overline{K}_3)_{lm} = \sum_{k=1}^{M} \int\int_{\Omega} \left(N_l \frac{u_k^{n+1} + u_k^n}{2} N_k \frac{\partial N_m}{\partial x} + N_l v_k^* N_k \frac{\partial N_m}{\partial y} \right) dxdy \qquad (A.35)$$

Now if we define the following matrices

$$A^n = \frac{2M}{\Delta t} - \overline{K}_1 \qquad (A.36)$$

$$B^n = \frac{2M}{\Delta t} + \overline{K}_2 \qquad (A.37)$$

$$C^n = \frac{2M}{\Delta t} + \overline{K}_3 \qquad (A.38)$$

$$f_\varphi^n = 2\overline{K}_1 \varphi^n \qquad (A.39)$$

$$f_u^n = -K_3(\varphi^{n+1} + \varphi^n) - 2\overline{K}_2 u^n + 2K_4 v^* \qquad (A.40)$$

$$f_v^n = -K_5(\varphi^{n+1} + \varphi^n) - K_4(u^{n+1} + u^n) - 2\overline{K}_3 v^n \qquad (A.41)$$

then equations (A.30), (A.32) and (A.34) become

$$A^n \Delta\varphi^n = f_\varphi^n \qquad (A.42)$$

$$B^n \Delta u^n = f_u^n \qquad (A.43)$$

$$C^n \Delta v^n = f_v^n \qquad (A.44)$$

where $\Delta\varphi^n = \varphi^{n+1} - \varphi^n$, $\Delta u^n = u^{n+1} - u^n$, $\Delta v^n = v^{n+1} - v^n$. The three linear equations (A.42), (A.43) and (A.44), after proper boundary conditions are incorporated, can be solved one by one at each time step.

To conclude this section, we mention that the θ scheme ($\theta = 1/2$) was also found to be appropriate in a slightly different application.

To simulate the tidal and surge flows due to sea-bed eruptions, Taylor and Davis [223], using Galerkin finite element method, solved the depth-averaged shallow water equations. They investigated three time marching algorithms, namely

1. An Adams-Moulton multi-step predictor-corrector procedure;

2. The finite difference trapezoidal integration (resulting in Crank-Nicolson scheme);

3. The finite element in time.

It was found that small time steps were necessary to achieve a satisfactory accuracy if method (1) was employed for time marching. The method (3) was found to be unable to handle the wave amplitude correctly and introduce spurious damping & phase retardation. The authors concluded that scheme (2) was satisfactory.

A.3 Properties of Global Stiffness Matrices and the Data Structure

We point out a couple of important properties of the global stiffness matrices in this section.

Although the matrix M (whose typical entries are given by (A.17) is symmetric, $\overline{K}_1$, $\overline{K}_2$ and $\overline{K}_3$ are not symmetric, as may be confirmed from their definitions (A.31), (A.33) and (A.35). As a result, the three global stiffness matrices A^n, B^n and C^n defined in (A.36), (A.37) and (A.38) are nonsymmetric. This means that one of the most favorable iterative algorithms, namely, the preconditioned conjugate gradient method (PCGS) [52] may not be applied for solving the linear systems (A.42), (A.43) and (A.44) at each time step. The nonsymmetric nature of the global stiffness matrices in this application may, in fact, be ascribed to the presence of advective terms $u\partial/\partial x$ and $v\partial/\partial y$ in the original shallow water equations.

Typical of the finite element method, most of the entries in the global stiffness matrices are zero, due to the fact that the global shape functions $N_i(x, y)$, $i = 1, 2, \ldots, M_{nd}$, have only local supports, namely, $N_i = 0$ outside a neighborhood of

the node i. Thus, although the size of the linear system will be of order $O(h^{-2})$, the number of indices j such that $a_{ij} \neq 0$ is $O(1)$ for each index i, where h is the mesh size.

For example, the non-zero regions of two global shape functions N_i and N_j depicted in Figure A.1 are spatially disjoint, it follows that entries $(\cdots)_{ij}$ or $(\cdots)_{ji}$ in the global matrices must vanish. For a given global node l, the non-zero portion of the shape function N_l overlaps the non-zero portions of at most six shape functions associated with its neighboring nodes (see Figure 4.2), it is thus clear that there are at most seven non-zero entries in any row of the stiffness matrix.

To save the computer storage and, at the same time, to keep the data structure as simple as possible, we store the global stiffness matrix, say A^n in (A.36), into a $M_{nd} \times 7$ two dimensional array `coef`, so that a maximum of seven non-zero entries in each row of the matrix may be accommodated. Except for diagonal entries of the original matrix which are always stored in $\texttt{coef}(i, 7)$, $i = 1, 2, \ldots, M_{nd}$, the original column number for any non-zero entry in `coef` is determined by another $M_{nd} \times 6$ two dimensional integer array `locat`. Specifically, for any given non-zero entry $\texttt{coef}(i, l)$, $i = 1, 2, \ldots, M_{nd}$ and $l = 1, 2, \ldots, 6$, we can always find its corresponding entry $(A^n)_{ij}$ in the matrix A^n, where $j = \texttt{locat}(i, l)$. Conversely, for any given non-zero entry $(A^n)_{ij}$, $i, j = 1, 2, \ldots, M_{nd}$, we may find a corresponding l, $l = 1, 2, \ldots, 6$, such that $j = \texttt{locat}(i, l)$. In general, there exists a one-to-one correspondence, via an integer array `locat`, between non-zero entries of the original square matrix A^n and a rectangular matrix `coef` with only seven columns. This constitutes a great saving of storage of up to $M_{nd} \times (M_{nd} - 7)$ floating-point numbers at the expense of introducing an integer array of size $M_{nd} \times 6$.

Table A.1 lists the contents of this two dimensional integer array `locat` which is obtained by running the code for a mesh resolution of 16×15 with a row-wise

numbering of the global nodes shown in Figure A.2. The typical sparse matrix structure corresponding to the finite element discretization on this seven point stencil shown in Figure A.2 is illustrated in Figure A.3, where each $\times$ represents a non-zero entry of the matrix. However, without loss of generality, we consider a much smaller computational domain which consists of only the first three grid lines in Figure A.2. Hence, the size of the matrix is much smaller (only 45×45), so that the matrix structure in Figure A.3 is clearly discernible.

Since there is an extreme unbalance, especially for high resolutions, between numbers of rows and columns in the array `coef` and only the innermost loop is able to be vectorized within a group of nested loops, it is important, for the manipulation of the array `coef` within a nested loops, to iterate the first index i of `coef`(i,l), $i = 1,2,\ldots,M_{nd}$, in the innermost loop. As an example, we consider a matrix-vector multiplication problem $y = A^n x$. There are two ways to do this and both will, of course, give the correct results.

Version 1:

```
      do 10 i = 1, n
        do 20 l = 1, 6
          j = locat(i,l)
          if (j .eq. 0 ) goto 20
          y(i) = y(i) + coef(i,l) * x(j)
20      continue
        y(i) = y(i) + coef(i,7) * x(i)
10    continue
```

Version 2:

```
      do 10 l = 1, 6
        do 20 i = 1, n
          j = locat(i,l)
```

Table A.1: locat(i,l), $i = 1,2,\ldots,M_{nd}$ and $l = 1,2,\ldots,6$, for the global numbering shown in Figure A.2

	$l=1$	$l=2$	$l=3$	$l=4$	$l=5$	$l=6$
$i=1$	2	16	15	30	0	0
$i=2$	1	16	17	3	0	0
$i=3$	2	17	18	4	0	0
$i=4$	3	18	19	5	0	0
$i=5$	4	19	20	6	0	0
...	...	...	...	...	...	...
$i=14$	13	28	29	15	0	0
$i=15$	14	29	30	1	0	0
$i=16$	1	2	17	30	31	45
$i=17$	16	2	3	18	31	32
$i=18$	17	3	4	19	32	33
...	...	...	...	...	...	...
$i=220$	219	205	206	221	0	0
$i=221$	220	206	207	222	0	0
$i=222$	221	207	208	223	0	0
$i=223$	222	208	209	224	0	0
$i=224$	223	209	210	225	0	0
$i=225$	224	210	196	211	0	0

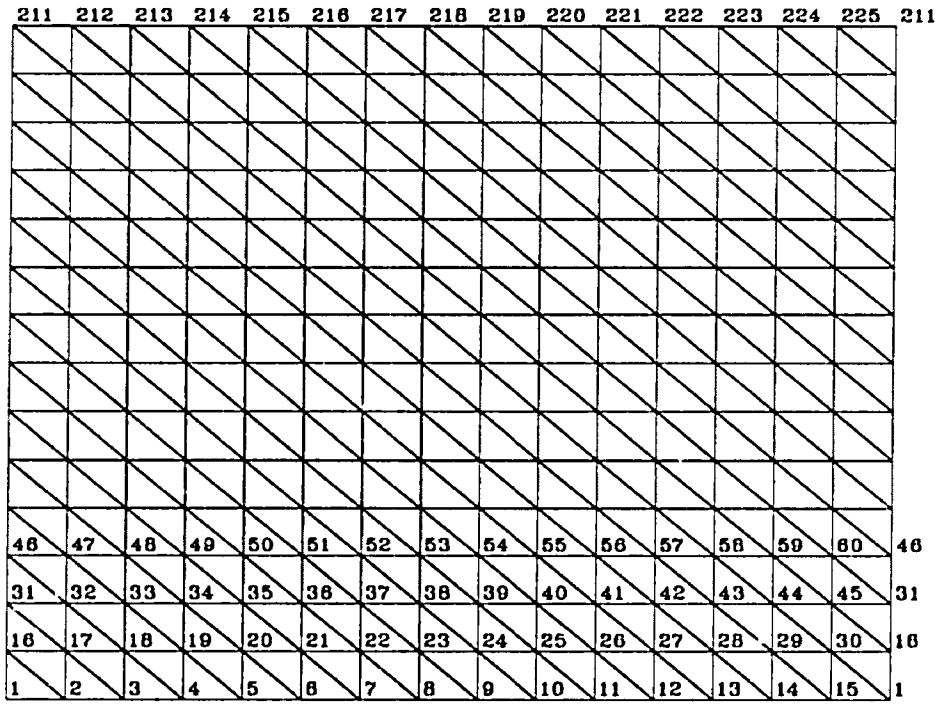


Figure A.2: A row–wise global numbering of nodes

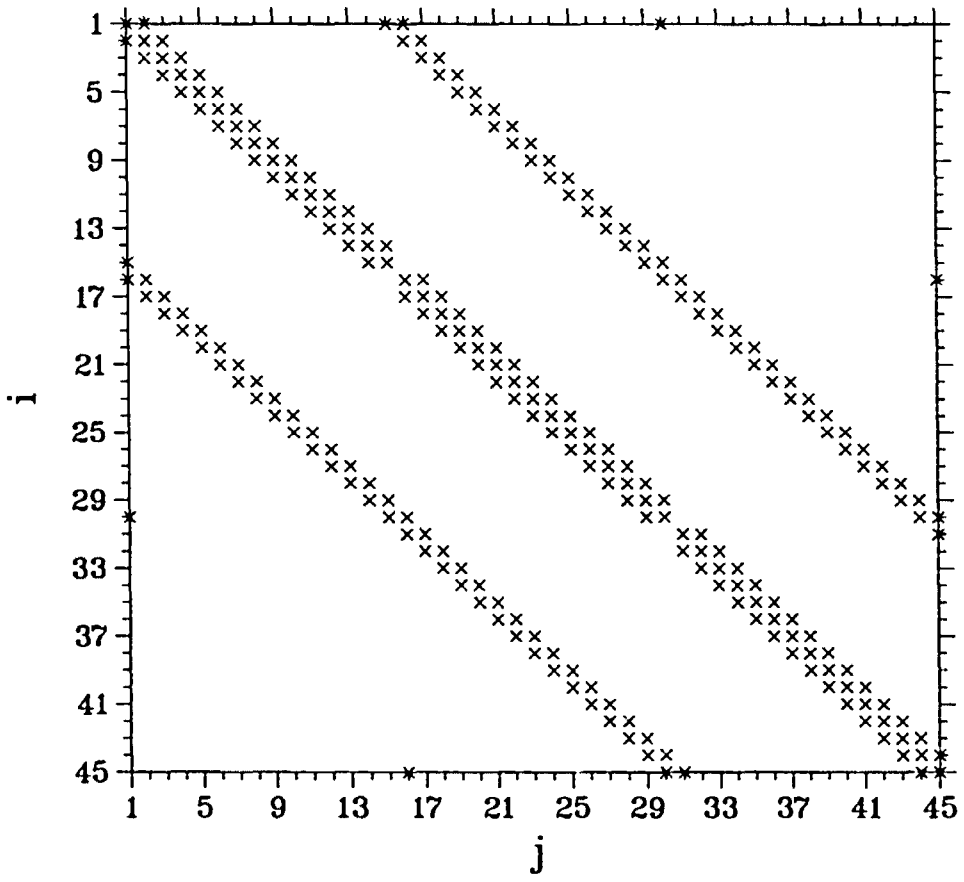


Figure A.3: The sparse matrix structure corresponding to the discretization on a seven point stencil linear triangular finite element mesh shown in Figure A.2, assuming that the computational domain consists of only the first three grid lines. Here, each $\times$ represents a non-zero entry in the global stiffness matrix.

```
            if (j .eq. 0 ) goto 20
            y(i) = y(i) + coef(i,1) * x(j)
20        continue
10      continue
        do 30 i = 1, n
          y(i) = y(i) + coef(i,7) * x(i)
30      continue
```

where n is the total number of nodes M_{nd}.

The dependence analysis may be carried out by invoking FPP[1] on the CRAY YMP. It turns out that the loop 20 in version 1 can also be vectorized, although the vector length is very short. It is apparent that the loop 20 in version 2 can be vectorized with a much longer vector length (equal to the number of rows in the global stiffness matrix) and hence is much more efficient. A comparison of CPU time between these two versions run for different mesh resolutions is provided in Table A.2.

To conclude this section, we notice that there are many other storage schemes for sparse matrices. For instance, three one dimensional arrays may be set up to store a sparse matrix ([77, 111]). If the nonzero elements of a matrix are near each other in every row, the so-called profile stotage is recommended ([114]).

[1]which looks for possibilities of vectorization and parallelization and generates a transformed FORTRAN source code with autotasking and compiler directives in it.

Table A.2: A comparison of CPU time (seconds) on CRAY Y-MP/432 between two codes which compute the multiplication of a global stiffness matrix by a vector for several mesh resolutions

Mesh resolutions	16×15	31×27	46×39	61×51
Version 1	6.4×10^{-4}	2.3×10^{-3}	5.0×10^{-3}	8.7×10^{-3}
Version 2	9.3×10^{-5}	3.2×10^{-4}	6.8×10^{-4}	1.2×10^{-3}
Mesh resolutions	81×75	115×99	161×147	220×203
Version 1	1.7×10^{-2}	3.2×10^{-2}	6.8×10^{-2}	1.3×10^{-1}
Version 2	2.3×10^{-3}	4.3×10^{-3}	8.9×10^{-3}	1.7×10^{-2}

A.4 Element Matrices

Consider a typical element Ω_e with nodes i, j and k numbered counter-clockwise. Assume the shape function defined on this element associated with the node i is

$$N_i^e(x,y) = \frac{1}{2A_e}(a_i + b_i x + c_i y) \tag{A.45}$$

where

$$A_e = \frac{1}{2}\begin{vmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{vmatrix} = \text{area of the e-th triangular element } \Omega_e \tag{A.46}$$

where the constants a_i, b_i and c_i are determined by the following requirement that $N_i^e(x_i, y_i) = 1$, $N_i^e(x_j, y_j) = N_i^e(x_k, y_k) = 0$. Thus,

$$a_i = x_j y_k - x_k y_j \tag{A.47}$$

$$b_i = y_j - y_k \tag{A.48}$$

$$c_i = x_k - x_j \tag{A.49}$$

Similarly, the other two local shape functions N_j^e and N_k^e associated with nodes j and k can be assumed to be

$$N_j^e(x,y) = \frac{1}{2A_e}(a_j + b_j x + c_j y) \tag{A.50}$$

$$N_k^e(x,y) = \frac{1}{2A_e}(a_k + b_k x + c_k y) \tag{A.51}$$

where the constants a_j, b_j, c_j, a_k, b_k and c_k are determined by equations (A.47), (A.48) and (A.49) through cyclic permutation. The element matrices can be evaluated by Sylvester's formula

$$\int\int_{\Omega_e} N_1^\alpha N_2^\beta N_3^\gamma dxdy = \frac{\alpha!\,\beta!\,\gamma!}{(\alpha+\beta+\gamma+2)!} \cdot 2A_e \tag{A.52}$$

The following element matrices may be obtained

$$M^e = \frac{A_e}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \tag{A.53}$$

$$K_3^e = \frac{1}{6}\begin{bmatrix} b_i & b_j & b_k \\ b_i & b_j & b_k \\ b_i & b_j & b_k \end{bmatrix} \tag{A.54}$$

To obtain an explicit expression for the element matrix K_4^e, notice that f is a linear function in x and y, as a result it can be represented exactly by

$$f = f_i N_i + f_j N_j + f_k N_k \tag{A.55}$$

over each element. Consequently, we have

$$K_4^e = \frac{A_e}{60}\begin{bmatrix} 6f_i + 2f_j + 2f_k & 2f_i + 2f_j + f_k & 2f_i + f_j + 2f_k \\ 2f_i + 2f_j + f_k & 2f_i + 6f_j + 2f_k & f_i + 2f_j + 2f_k \\ 2f_i + f_j + 2f_k & f_i + 2f_j + 2f_k & 2f_i + 2f_j + 6f_k \end{bmatrix} \tag{A.56}$$

$$K_5^e = \frac{1}{6}\begin{bmatrix} c_i & c_j & c_k \\ c_i & c_j & c_k \\ c_i & c_j & c_k \end{bmatrix} \tag{A.57}$$

$$\overline{K_1^e} = \frac{1}{24}\begin{bmatrix} b_i d_i & b_i d_j & b_i d_k \\ b_j d_i & b_j d_j & b_j d_k \\ b_k d_i & b_k d_j & b_k d_k \end{bmatrix} + \frac{1}{24}\begin{bmatrix} c_i c_i & c_i c_j & c_i c_k \\ c_j c_i & c_j c_j & c_j c_k \\ c_k c_i & c_k c_j & c_k c_k \end{bmatrix} \tag{A.58}$$

$$\overline{K_2^e} = \frac{1}{24}\begin{bmatrix} b_i d_i & b_j d_i & b_k d_i \\ b_i d_j & b_j d_j & b_k d_j \\ b_i d_k & b_j d_k & b_k d_k \end{bmatrix} + \frac{1}{24}\begin{bmatrix} c_i c_i & c_j c_i & c_k c_i \\ c_i c_j & c_j c_j & c_k c_j \\ c_i c_k & c_j c_k & c_k c_k \end{bmatrix} \tag{A.59}$$

where

$$d_i = 2u_i^* + u_j^* + u_k^* \tag{A.60}$$

$$d_j = u_i^* + 2u_j^* + u_k^* \tag{A.61}$$

$$d_k = u_i^* + u_j^* + 2u_k^* \tag{A.62}$$

$$c_i = 2v_i^* + v_j^* + v_k^* \tag{A.63}$$

$$c_j = v_i^* + 2v_j^* + v_k^* \tag{A.64}$$

$$c_k = v_i^* + v_j^* + 2v_k^* \tag{A.65}$$

The element matrix $\overline{K_3^e}$ is the same as $\overline{K_2^e}$ except that u_α^*, $\alpha = i, j, k$ in equations (A.60), (A.61) and (A.62) will be replaced by $(u_\alpha^{n+1} + u_\alpha^n)/2$.

A.5 Truncation Error for the Single Stage Galerkin Finite Element Method

In this section and the rest of this appendix, we try to summarize the essential components of a two stage Numerov-Galerkin finite element method (see [169] for

a full documentation of the method and [170] for computer implementation issues. Key features of this approach are also presented in [174]).

The single-stage Galerkin method [58] was applied to the nonlinear advective terms of form $v\nabla v$. If we consider the advective operator

$$L(u,v) = u\frac{\partial v}{\partial x} \tag{A.66}$$

then, as was shown in [58], we may consider a direct Galerkin approximation using two functions

$$u = e^{ikx} \quad \text{and} \quad v = e^{ilx} \tag{A.67}$$

and with

$$\xi = kh \quad \text{and} \quad \eta = lh \tag{A.68}$$

where h is a positive mesh length. One can show the asymptotic truncation error (T.E.) of $u\partial v/\partial x$ is (by assuming Fourier modes)

$$|\text{T.E.}| \sim \frac{[4\eta^4 + 8\eta^2\xi + 7\eta^2\xi^2 - 2\eta\xi^3]}{720} \tag{A.69}$$

For $\xi = \eta$, we obtain

$$|\text{T.E.}| \sim \frac{17}{720}\eta^4 \tag{A.70}$$

A.6 Truncation Error for the Two Stage Numerov-Galerkin Finite Element Method

In this approach one calculates the Galerkin approximation to $\partial v/\partial x$ which we denote by Z:

$$\frac{1}{6}Z_{j-1} + \frac{2}{3}Z_j + \frac{1}{6}Z_{j-1} = \frac{1}{2}h^{-1}(V_{j+1} - V_{j-1}) \tag{A.71}$$

then we calculate the product

$$W = u\frac{\partial v}{\partial x}. \tag{A.72}$$

The following formula can be obtained

$$\frac{1}{6}W_{j+1}+\frac{2}{3}W_j+\frac{1}{6}W_{j-1}=\frac{1}{12}(U_{j-1}Z_{j-1}+U_{j-1}Z_j + U_jZ_{j-1}+U_jZ_{j+1}+U_{j+1}Z_j+U_{j+1}Z_{j+1})+\frac{1}{2}U_jZ_j \tag{A.73}$$

It can be shown that this two stage Numerov-Galerkin algorithm has an asymptotic truncation error (denoted by N.G.T.E.) of

$$\frac{|\text{T.E.}|}{|\text{N.G.T.E.}|}\sim\frac{[2\xi^3\eta+3\xi^2\eta^2+2\xi\eta^3-4\eta^4]}{720} \tag{A.74}$$

and if $\xi=\eta$

$$\frac{|\text{T.E.}|}{|\text{N.G.T.E.}|}\sim\frac{3}{720}\eta^4 \tag{A.75}$$

namely, an error at least six times smaller than (A.70) (see [169]).

A.7 Numerical Implementation of the Numerov-Galerkin Method

In our approach we combine the two-stage Galerkin method with a high-order compact implicit difference approximation to the first derivative.

This approximation has a truncation-error of $o(h^{4l})$ and uses a finite-difference stencil of $2l+1$ grid points — at the price of solving a $2l+1$ banded matrix (See [177, 208]). The compact Numerov $O(h^8)$ approximation to $\partial v/\partial x$ is given by

$$\frac{1}{70}\left[\left(\frac{\partial v}{\partial x}\right)_{i+2}+16\left(\frac{\partial v}{\partial x}\right)_{i+1}+36\left(\frac{\partial v}{\partial x}\right)_{i}+16\left(\frac{\partial v}{\partial x}\right)_{i-1} + \left(\frac{\partial v}{\partial x}\right)_{i-2}\right]=\frac{1}{84h}[-5v_{i-2}-32v_{i-1}+32v_{i+1}+5v_{i+2}] \tag{A.76}$$

where $h=\Delta x=\Delta y$.

The estimation of $\partial v/\partial x$ necessitates solving a pentadiagonal system of the form:

$$\frac{1}{70}\begin{bmatrix} 36 & 16 & 1 & & & & \\ 16 & 36 & 16 & 1 & & & \\ 1 & 16 & 36 & 16 & 1 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 1 & 16 & 36 & 16 & 1 \\ & & & 1 & 16 & 36 & 16 \\ & & & & 1 & 16 & 36 \end{bmatrix} \begin{bmatrix} \frac{\partial v}{\partial x} \end{bmatrix}$$

$$= \frac{1}{84h}\begin{bmatrix} 0 \\ -5v_0 - 32v_1 + 32v_3 + 5v_4 \\ -5v_1 - 32v_2 + 32v_4 + 5v_5 \\ \cdot \\ \cdot \\ \cdot \\ -5v_{N_y-4} - 32v_{N_y-2} + 32v_{N_y} + 5v_{N_y+1} \\ -5v_{N_y-3} - 32v_{N_y-2} + 32v_{N_y} + 5v_{N_y+1} \end{bmatrix} \tag{A.77}$$

for $j = 1, 2, \ldots, N_y$.

Here we interpolate for the boundary values of v_0 and v_{N_y+1} using

$$v_0 = 4v_1 - 6v_2 + 4v_3 - v_4 \tag{A.78}$$

$$v_{n_y+1} = 4v_{N_y} - 6v_{N_y-1} + 4v_{N_y-2} - v_{N_y-3} \tag{A.79}$$

while for the intermediate expression Z we have

$$\begin{aligned} Z_1 &= \left(\frac{\partial v}{\partial y}\right)_1 \\ &= \frac{-25v_1 + 48v_2 - 36v_3 + 16v_4 - 3v_5}{12h} + o(h^4) \end{aligned} \tag{A.80}$$

$$
\begin{aligned}
Z_{n_y} &= \left(\frac{\partial v}{\partial y}\right)_{N_y} \\
&= \frac{3v_{N_y-4} - 16v_{N_y-3} + 36v_{N_y-2} - 48v_{N_y-1} + 25v_{N_y}}{12h} \\
&\quad + o(h^4) \qquad \text{(A.81)}
\end{aligned}
$$

For the second stage of the Numerov-Galerkin finite element, we solve a tridiagonal system of the form

$$
\begin{bmatrix}
4 & 1 & & & \\
1 & 4 & 1 & & \\
 & \ddots & \ddots & \ddots & \\
 & & 1 & 4 & 1 \\
 & & & 1 & 4
\end{bmatrix}
[w_j] = \frac{1}{2}
\begin{bmatrix}
\\
v_{j-1}Z_{j-1} + v_jZ_{j-1} + v_{j-1}Z_j + \\
v_{j+1}Z_j + v_jZ_{j+1} + v_{j+1}Z_{j+1} + 6v_jZ_j \\
\\
\end{bmatrix} \qquad \text{(A.82)}
$$

In the second stage we interpolate the values of Z_0 and Z_{N_y+1} in a way similar to equations (A.80) and (A.81). A pentadiagonal and a cyclic pentadiagonal matrix solver (induced by the periodic boundary conditions) were developed in [171] following [230] and generalizing [5], respectively.

A.8 Some Numerical Results

In this section, we present some finite element solutions of the shallow water equations introduced in Chapter 4. The pre-selected reference geopotential φ_0 is chosen to be 10^4 so that the non-dimensionalized geopotential has an order of magnitude of $O(1)$. Grammeltvedt's initial condition (see Figure A.4) has been used throughout. The contour lines of the geopotential at the end of i'th day are presented, $i = 1, 2, \ldots, 10$, in Figure A.5 – A.14. All solutions are obtained with a mesh resolution of 35×27 and with a time step $\Delta t = 1800$ seconds or half an hour. Thus, to get the solution corresponding to the end of ten days, 480 iterations are required.

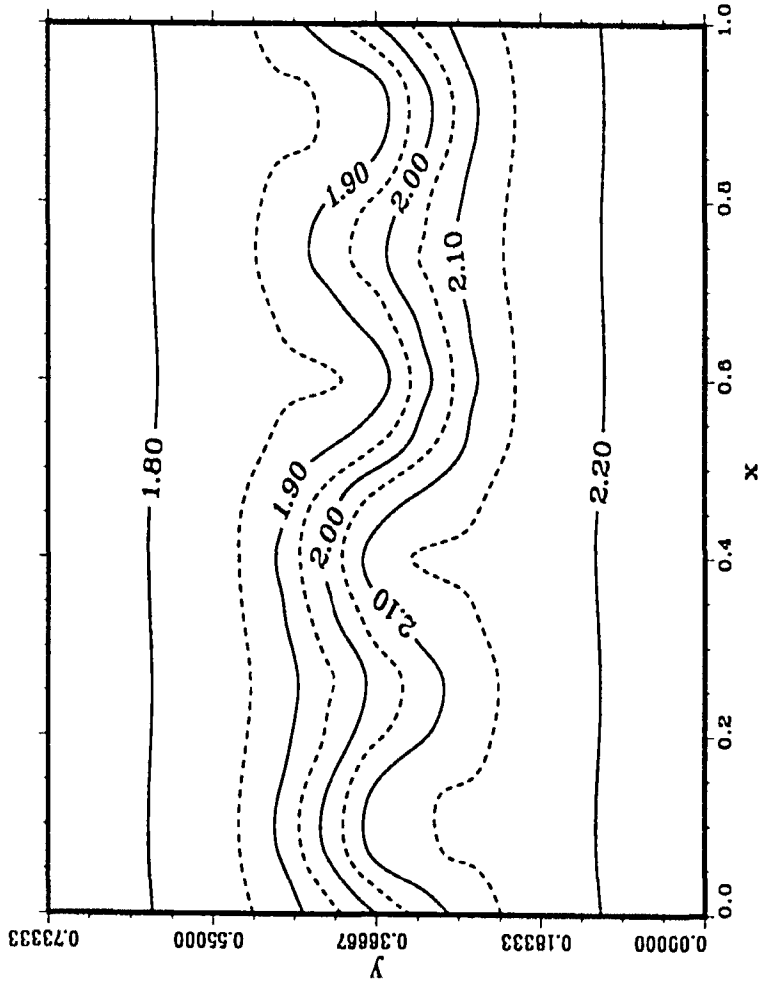


Figure A.3: Grammeltvedt's initial geopotential.

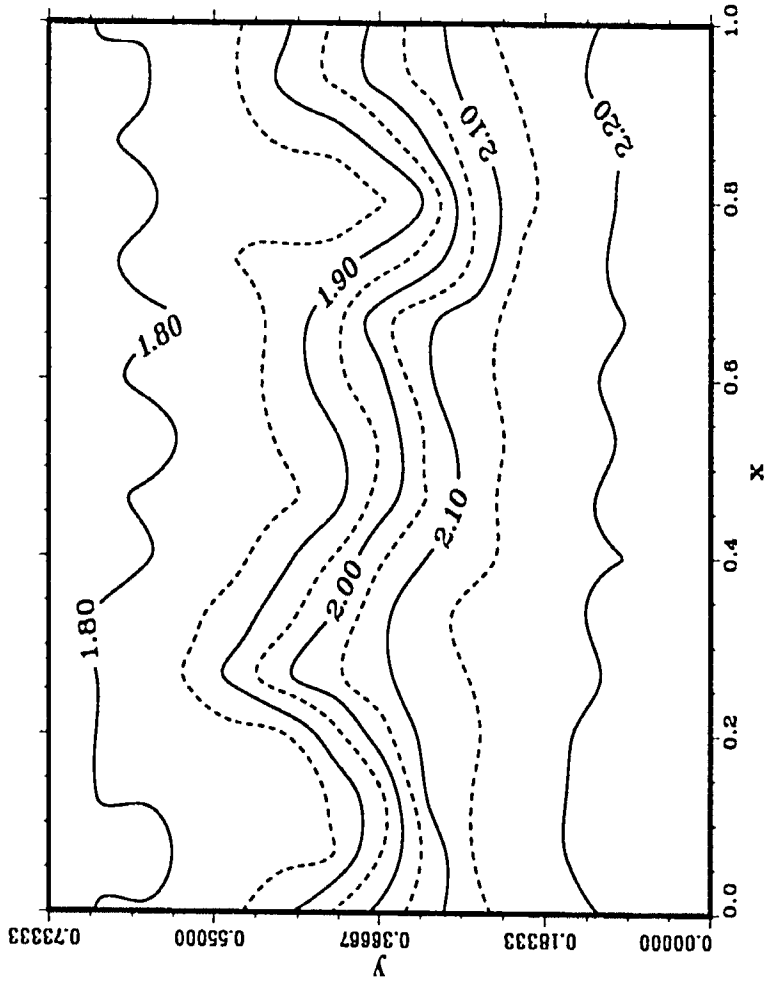


Figure A.4: The geopotential at the end of 1 day.

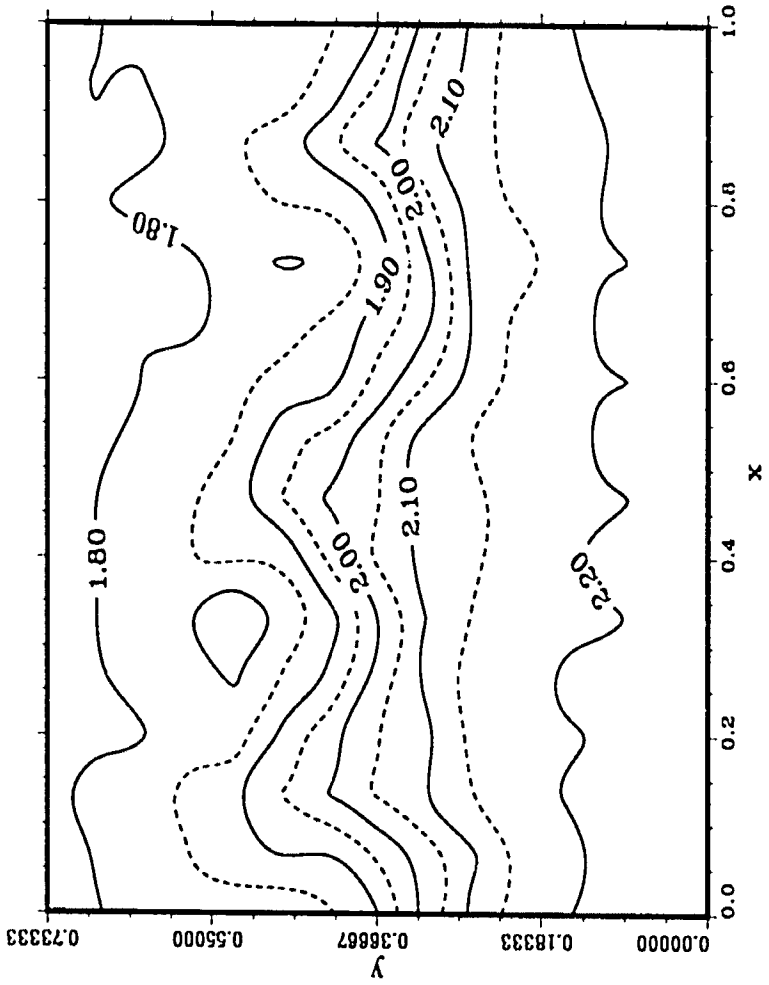


Figure A.5: The geopotential at the end of 2 days.

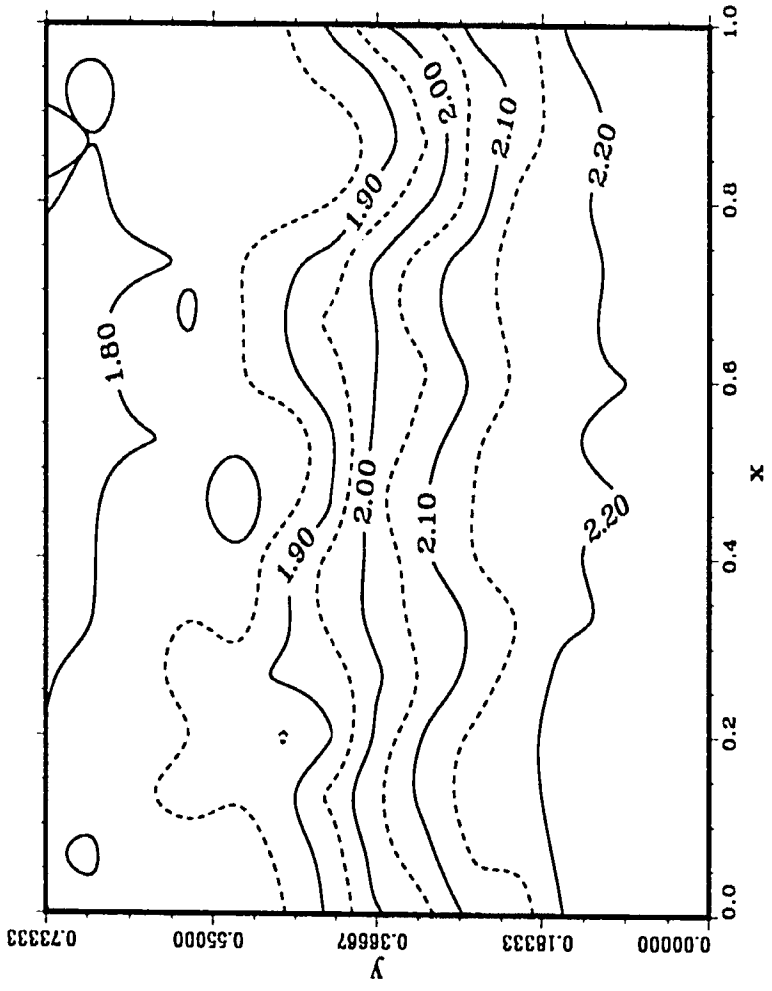


Figure A.6: The geopotential at the end of 3 days.

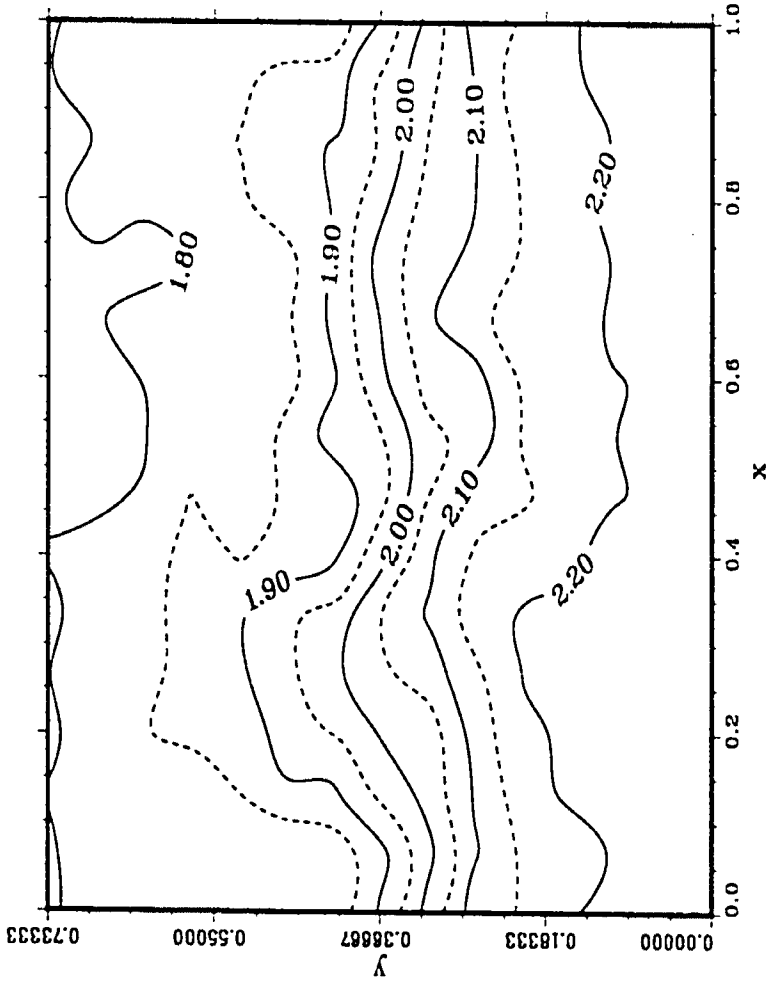


Figure A.7: The geopotential at the end of 4 days.

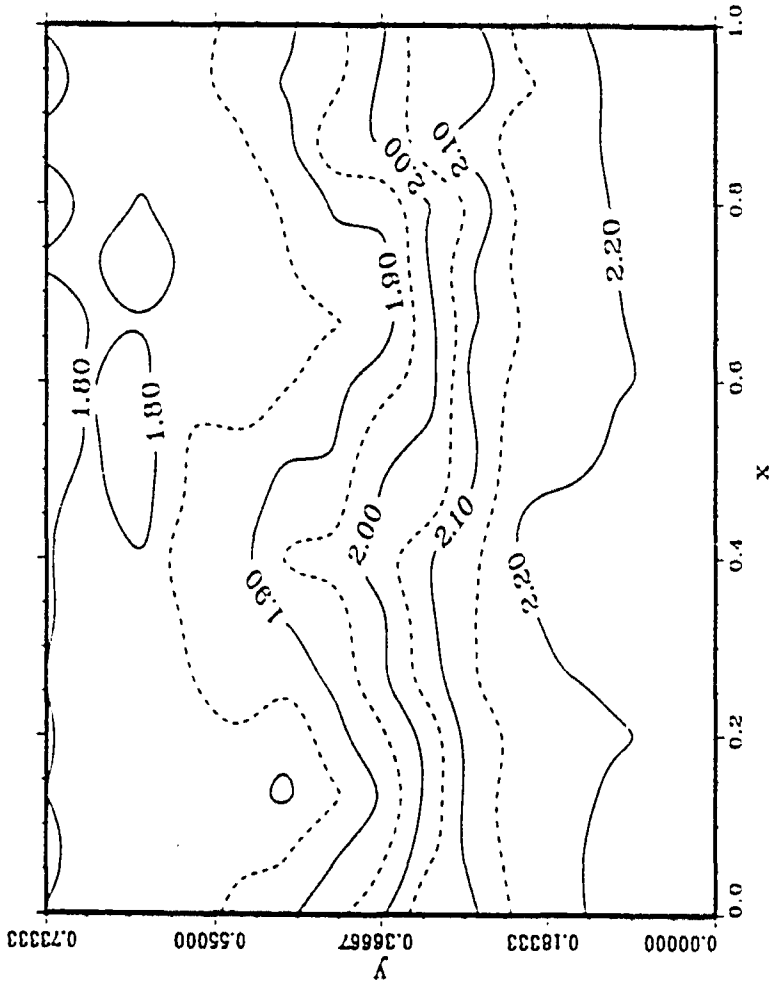


Figure A.8: The geopotential at the end of 5 days.

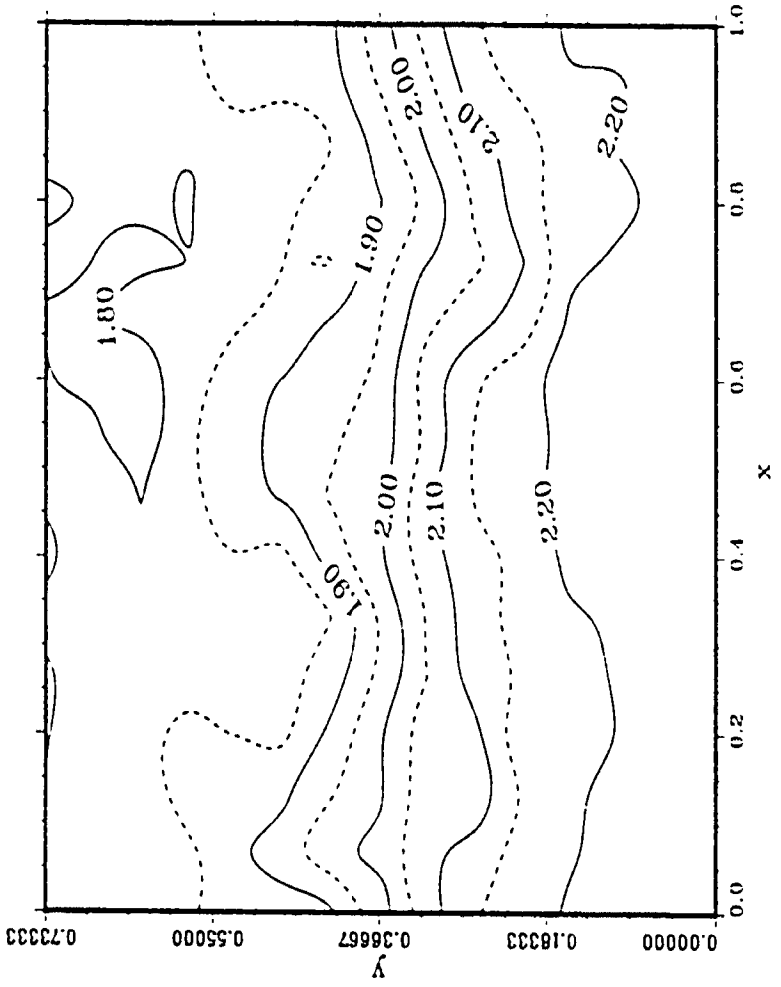


Figure A.9: The geopotential at the end of 6 days.

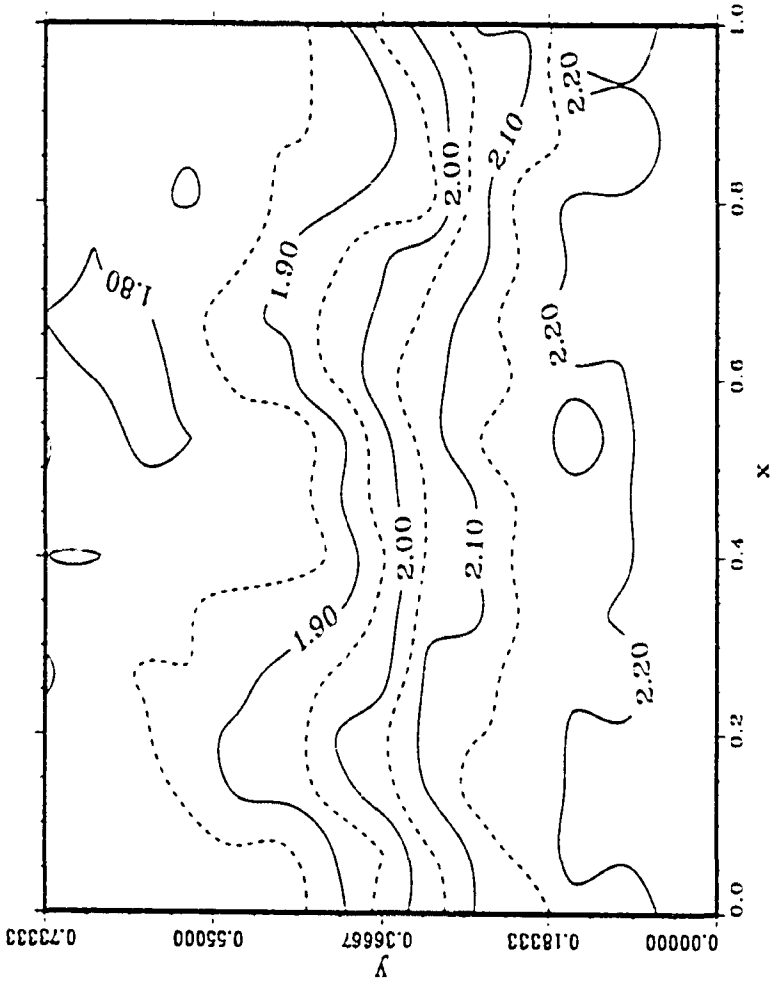


Figure A.10: The geopotential at the end of 7 days.

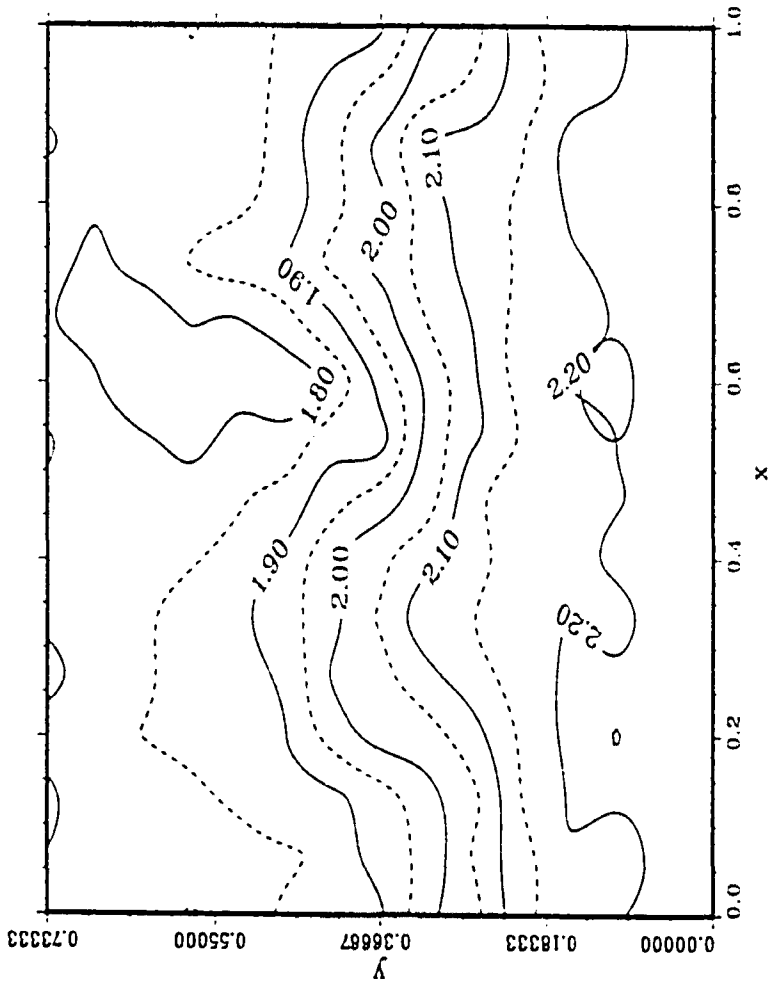


Figure A.11: The geopotential at the end of 8 days.

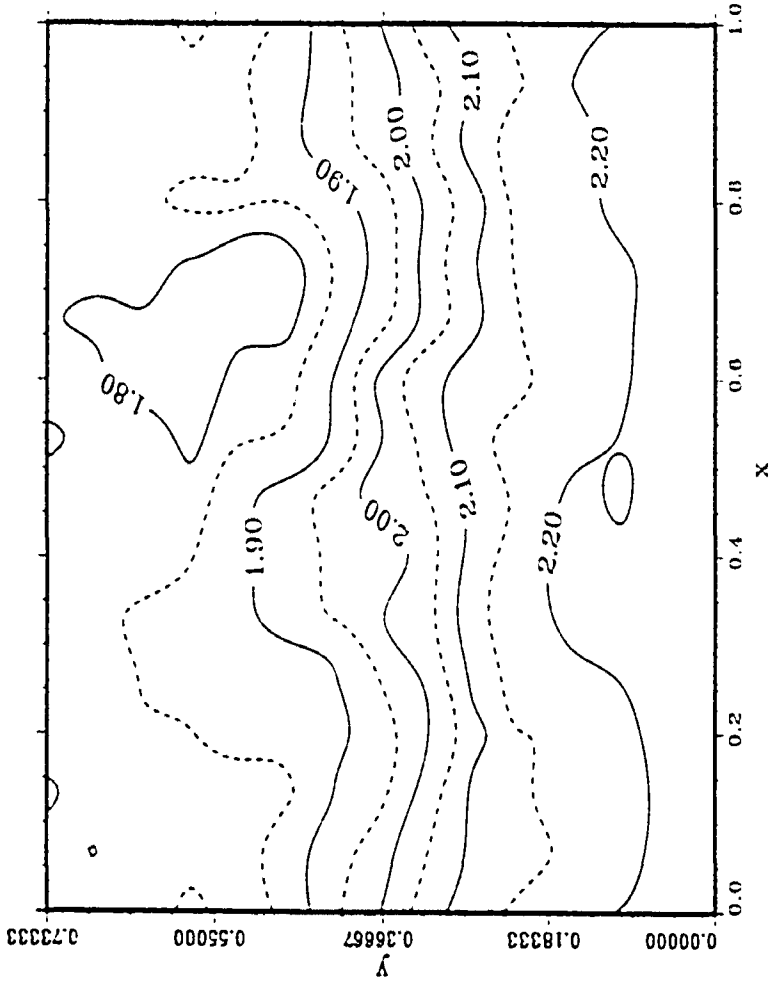


Figure A.12: The geopotential at the end of 9 days.

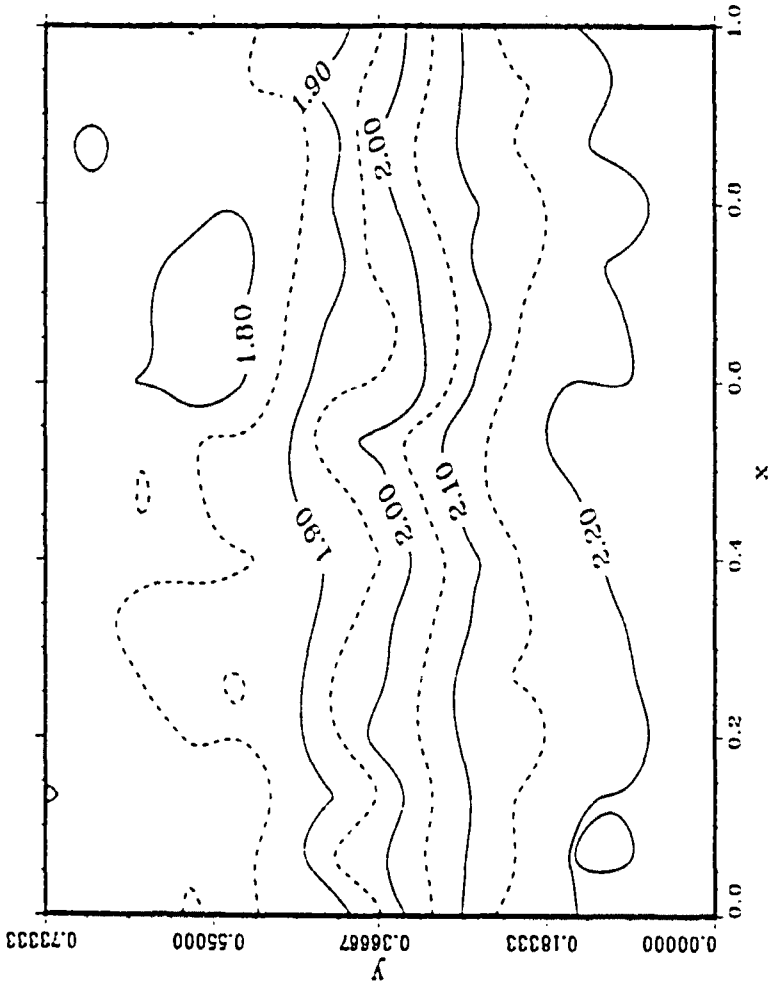


Figure A.13: The geopotential at the end of 10 days.

REFERENCES

[1] L. Adams. m-Step preconditioned conjugate gradient methods. *SIAM J. Sci. Stat. Comput.*, 6(2):452–463, 1985.

[2] L. Adams. Reordering computations for parallel execution. *Comm. Appl. Numer. Meth.*, 2:263–271, 1986.

[3] L. Adams and R. Voigt. A methodology for exploiting parallelism in the finite element process. In J. Kowalik, editor, *Proceedings of the NATO Workshop on High Speed Computations, NATO ASI Series*, pages 373–392. Springer, Berlin, 1984.

[4] V.I. Agoshkov. Poincaré-Steklov's operators and domain decomposition methods in finite dimensional spaces. In R. Glowinski, G.H. Golub, G.A. Meurant, and J. Périaux, editors, *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 73–112. SIAM, Philadelphia, 1988.

[5] H.H. Ahlberg, E.N. Nielson, and J.L. Walsh. *The Theory of Splines and Their Application, Mathematics in Science and Engineering*. 38. Academic Press, New York, 1967.

[6] G.M. Amdahl. The validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the AFIPS Computing Conference*, Volume 30, pages 483–485, 1967.

[7] G.M. Amdahl. Limits of expectation. *Int. J. Supercomputer Appl.*, 2(1):88–97, 1988.

[8] O. Axelsson. Incomplete block matrix factorization preconditioning methods. The ultimate answer? *J. Comp. Appl. Math.*, 12&13:3–18, 1985.

[9] O. Axelsson. A survey of preconditioned iterative methods for linear systems of algebraic equations. *Bit*, 25:166–187, 1986.

[10] O. Axelsson and V.A. Barker. *Finite Element Solutions of Boundary Value Problems. Theory and Computation.* Academic Press, New York, 1984.

[11] O. Axelsson and G. Lindskog. On the eigenvalue distribution of a class of preconditioning methods. *Numer. Math.*, 48:479–498, 1986.

[12] O. Axelsson and N. Munksgaard. A class of preconditioned conjugate gradient methods for the solution of a mixed finite-element discretization of the biharmonic operator. *Int. J. Numer. Math. Eng.*, 14:1001–1019, 1978.

[13] O. Axelsson and N. Munksgaard. Analysis of incomplete factorizations with fixed storage allocation. In D.J. Evans, editor, *Preconditioning Methods: Analysis and Applications*, pages 219–241. Gordon and Breach, Science Publishers, Inc., 1983.

[14] O. Axelsson and B. Polman. Block preconditioning and domain decomposition methods I. Technical Report 8735, Department of Mathematics, University of Nijmegen, 1987.

[15] R.G. Babb II. *Programming Parallel Processors.* Addison-Wesley Publishing Company, Inc., 1988.

[16] I. Babuška. On the Schwarz algorithm in the theory of differential equations of mathematical physics. *Tchecosl. Math. J.*, 8:328–342, 1958.

[17] B.E. Bank, W.M. Coughran, M.A. Driscoll R.K. Smith, and W. Fichtner. Iterative methods in semiconductor device simulation. *Comput. Phys. Comm.*, 53:201–212, 1989.

[18] S. Barnett. *Matrices, Methods and Applications.* Clarendon Press, Oxford, 1990.

[19] Ph. Berger, D. Comte, and Ch. Fraboul. MIMD Supercomputers for numerical applications. In J.T. Devreese and P.V. Camp, editors, *Supercomputers in Theoretical and Experimental Science*, pages 115–142. Plenum Press, New York, 1985.

[20] P.E. Bjørstad and O.B. Widlund. Iterative methods for the solution of elliptic problems on regions partitioned into substructures. *SIAM J. Numer. Anal.*, 23(6):1097–1120, 1986.

[21] P.E. Bjørstad and O.B. Widlund. To overlap or not to overlap: a note on a domain decomposition method for elliptic problems. *SIAM J. Sci. Stat. Comput.*, 10(5):1053–1061, 1989.

[22] J.H. Bramble, J.E. Pasciak, and A.H. Schatz. The construction of preconditioners for elliptic problems by substructures. *Math. Comp.*, 47:103–134, 1986.

[23] J.H. Bramble, J.E. Pasciak, and A.H. Schatz. An iterative method for elliptic problems on regions partitioned into substructures. *Math. Comp.*, 46:361–369, 1986.

[24] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Math. Comp.*, 31(138):333–390, 1977.

[25] A. Brandt. Guide to multigrid development. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods, Lecture Notes in Mathematics 960*, pages 220–312. Springer-Verlag Berlin Heidelberg, 1982.

[26] S. Brawer. *Introduction to Parallel Programming*. Academic Press, New York, 1989.

[27] W.L. Briggs. *A Multigrid Tutorial*. SIAM, Philadelphia, 1987.

[28] P.N. Brown. A theoretical comparison of the Arnoldi and GMRES algorithms. *SIAM J. Sci. Stat. Comput.*, 12(1):58–78, 1991.

[29] G. Brussino and V. Sonnad. A comparison of direct and preconditioned iterative techniques for sparse, unsymmetric systems of linear equations. *Int. J. Numer. Methods Eng.*, 28:801–815, 1989.

[30] X. Cai, W.D. Gropp, and D.E. Keyes. A comparison of some domain decomposition algorithms for nonsymmetric elliptic problems. In D.E. Keyes, T.F. Chan, G. Meurant, J.S. Scroggs, and R.G. Voigt, editors, *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 224–235. SIAM, Philadelphia, 1992.

[31] X. Cai, W.D. Gropp, and D.E. Keyes. A comparison of some domain decomposition and ILU preconditioned iterative methods for nonsymmetric elliptic problems. *J. Numer. Lin. Alg. Applics.*, to appear.

[32] X. Cai, W.D. Gropp, D.E. Keyes, and M.D. Tidriri. Newton-Krylov-Schwarz methods in CFD. In D.E. Keyes and J. Xu, editors, *Seventh International*

Conference on Domain Decomposition Methods in Scientific and Engineering Computing. AMS series of Contemporary Mathematics, American Mathematical Society, to appear.

[33] X. Cai and O.B. Widlund. Domain decomposition algorithms for indefinite elliptic problems. *SIAM J. Sci. Stat. Comput.*, 13(1):243–258, 1992.

[34] Y. Cai and I.M. Navon. Iterative domain decomposition algorithms: theory and applications. In F.X. Le Dimet, editor, *High Performance Computing in the Geosciences*. Kluwer Academic Publishers B.V., To appear.

[35] Y. Cai and I.M. Navon. Parallel block preconditioning techniques for the numerical simulation of the shallow water flow using finite element methods. *J. Comput. Phys.*, to appear.

[36] Y. Cai and I.M. Navon. Parallel domain decomposed preconditioners for the finite element shallow water flow modeling. In D.E. Keyes and J. Xu, editors, *Seventh International Conference on Domain Decomposition Methods in Scientific and Engineering Computing*. AMS series of Contemporary Mathematics, American Mathematical Society, to appear.

[37] G.F. Carey. Parallelism in finite element modeling. *Comm. Appl. Numer. Meth.*, 2:281-287, 1986.

[38] T.F. Chan. Analysis of preconditioners for domain decomposition. *SIAM J. Numer. Anal.*, 24(2):382–390, 1987.

[39] T.F. Chan. A domain-decomposed fast Poisson solver on a rectangle. *SIAM J. Sci. Stat. Comput.*, 8(1):s14–s26, 1987.

[40] T.F. Chan. Boundary probe domain decomposition preconditioners for fourth order problems. In T.F. Chan, R. Glowinski, J. Périaux, and O.B. Widlund, editors, *Second International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 160–167. SIAM, Philadelphia, 1989.

[41] T.F. Chan. Fourier analysis of relaxed incomplete factorization preconditioners. *SIAM J. Sci. Stat. Comput.*, 12(3):668–680, 1991.

[42] T.F. Chan and D. Goovaerts. On the relationship between overlapping and nonoverlapping domain decomposition methods. *SIAM J. Matrix Anal. Appl.*, 13(2):663–670, 1992.

[43] T.F. Chan and D.E. Keyes. Interface preconditioner for domain-decomposed convection-diffusion operators. In T.F. Chan, R. Glowinski, J. Périaux, and O.B. Widlund, editors, *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 245–262. SIAM, Philadelphia, 1990.

[44] T.F. Chan and T.P. Mathew. The interface probing technique in domain decomposition. *SIAM J. Matrix Anal. Appl.*, 13(1):212–238, 1992.

[45] T.F. Chan and D. Resasco. A survey of preconditioners for domain decomposition. Technical Report 414, Computer Science Dept., Yale Univ., 1985.

[46] T.F. Chan and D.C. Resasco. A framework for the analysis and construction of domain decomposition preconditioners. In R. Glowinski, G.H. Golub, G.A. Meurant, and J. Périaux, editors, *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 217–230. SIAM, Philadelphia, 1988.

[47] R. Chandra. *Conjugate gradient methods for partial differential equations.* PhD thesis, Computer Science Dept., Yale Univ., New Haven, CT, 1978.

[48] S.C. Chen, D.J. Kuck, and A.H. Sameh. Practical parallel band triangular system solvers. *ACM Trans. Math. Softw.*, 4:270–277, 1978.

[49] A.T. Chronopoulos. A fast squared Lanczos method for nonsymmetric linear systems. Technical Report UMSI 91/310, Univ. of Minnesota Supercomputer Institute, 1991.

[50] R.W. Clough. The finite element in plane stress analysis. In *Proceedings of the Second A.S.C.E. Conference on Electronic Computation*, pages 345–378, 1960.

[51] R.W. Clough. The finite element method after twenty-five years: a personal view. *Computers and Structures*, 12:361–370, 1980.

[52] P. Concus, G.H. Golub, and D.P. O'Leary. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. In J.R. Bunch and D.J. Rose, editors, *Sparse Matrix Computations*, pages 309–332. Academic Press, New York, 1976.

[53] R. Cottle. Manifestations of the Schur complement. *Linear Algebra Appl.*, 8:189–211, 1974.

[54] R. Courant. Variational methods for the solution of problems of equilibrium and vibration. *Bull. Am. Math. Soc.*, 49:1–23, 1943.

[55] CRAY Research, Inc., Distribution Center, 2360 Pilot Knob Road, Mendota Heights, MN 55120. *UNICOS Performance Utilities Reference Manual, SR-2040 6.0*, 1987.

[56] CRAY Research, Inc., Distribution Center, 2360 Pilot Knob Road, Mendota Heights, MN 55120. *CRAY Y-MP^{TM}, CRAY X-MP EA^{TM}, and CRAY X-MP^{TM} multitasking programmer's manual, SR-0222 F-01*, 1989.

[57] CRAY Research, Inc., Distribution Center, 2360 Pilot Knob Road, Mendota Heights, MN 55120. *CF77^{TM} Compiling Systems, Volume 4: Parallel Processing Guide, SG-3074 4.0*, 1990.

[58] M.J.P. Cullen and K.W. Morton. Analysis of evolutionary error in finite-element and other methods. *J. Comput. Phys.*, 34:245–267, 1980.

[59] A.R. Curtis, M.J. Powell, and J.K. Reid. On the estimation of sparse Jacobian matrices. *J. Inst. Math. Appl.*, 13:117–120, 1974.

[60] J.W. Daniel. The conjugate gradient method for linear and nonlinear operator equations. *SIAM J. Numer. Anal.*, 4:10–26, 1967.

[61] B.W. Davies. Supercomputing — a forward look. In R.G. Evans and S. Wilson, editors, *Supercomputational Science*, pages 333–343. Plenum Press, New York, 1990.

[62] Q.V. Dinh, R. Glowinski, J. Périaux, and G. Terrasson. On the coupling of viscous and inviscid models for incompressible fluid flows via domain decomposition. In R. Glowinski, G.H. Golub, G.A. Meurant, and J. Périaux, editors, *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 350–369. SIAM, Philadelphia, 1988.

[63] D. Dodson and J. Lewis. Issues relating to extension of the Basic Linear Algebra Subprograms. *ACM SIGNUM Newsletter*, 20(1):2–18, 1985.

[64] J.J. Dongarra, J. DuCroz, I.S. Duff, and S. Hammarling. A set of level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.*, 16:1–17, 1990.

[65] J.J. Dongarra, J. DuCroz, S. Hammarling, and R. Hanson. An extended set of Fortran Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.*, 14:1–17, 1988.

[66] J.J. Dongarra, I.S. Duff, D.C. Sorensen, and H.A. Van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers*. SIAM, Philadelphia, 1991.

[67] J.J. Dongarra and D.C. Sorensen. A portable environment for developing parallel FORTRAN programs. *Parallel Computing*, 5:175–186, 1987.

[68] J.J. Dongarra, D.C. Sorensen, K. Connolly, and J. Patterson. Programming methodology and performance issues for advanced computer architectures. *Parallel Computing*, 8:41–58, 1988.

[69] M. Dryja. A capacitance matrix method for Dirichlet problem on polygon regions. *Numer. Math.*, 39:51–64, 1982.

[70] M. Dryja. A finite element-capacitance method for elliptic problems on regions partitioned into subregions. *Numer. Math.*, 44:153–168, 1984.

[71] M. Dryja. An additive Schwarz algorithm for two- and three-finite element elliptic problems. In T.F. Chan, R. Glowinski, J. Périaux, and O.B. Widlund, editors, *Second International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 168–172. SIAM, Philadelphia, 1989.

[72] M. Dryja and O.B. Widlund. An additive variant of the Schwarz alternating method for the case of many subregions. Technical Report 339, Courant Institute, NYU, 1987.

[73] M. Dryja and O.B. Widlund. Some domain decomposition algorithms for elliptic problems. In D.R. Kincaid and L.J. Hayes, editors, *Iterative Methods for Large Linear Systems*, pages 273–291. Academic Press, Inc., 1990.

[74] M. Dryja and O.B. Widlund. Towards a unified theory of domain decomposition algorithms for elliptic problems. In T.F. Chan, R. Glowinski, J. Périaux, and O.B. Widlund, editors, *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 3–21. SIAM, Philadelphia, 1990.

[75] M. Dryja and O.B. Widlund. Additive Schwarz methods for elliptic finite element problems in three dimensions. In D.E. Keyes, T.F. Chan, G. Meurant, J.S. Scroggs, and R.G. Voigt, editors, *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 3–18. SIAM, Philadelphia, 1992.

[76] P. Dubois and G. Rodrigue. An analysis of the recursive doubling algorithm. In D. Kuck, D. Lawrie, and A. Sameh, editors, *High Speed Computer and Algorithm Organization*, pages 299–305. Academic Press, New York, 1977.

[77] I.S. Duff. Research directions in sparse matrix computations. In G.H. Golub, editor, *Studies in Numerical Analysis, Volume 24 of Studies in Mathematics*, pages 83–139. Mathematical Association of America, 1984.

[78] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, London, 1986.

[79] S.C. Eisenstat. Personal communication, 1985.

[80] H.C. Elman. *Iterative methods for large sparse nonsymmetric systems of linear equations.* PhD thesis, Computer Science Dept., Yale Univ., New Haven, CT, 1982.

[81] D.J. Evans. *Preconditioning Methods: Analysis and Applications.* Gordon and Breach, Science Publishers, Inc., 1983.

[82] R. G. Evans. Supercomputing on conventional architectures. In R.G. Evans and S. Wilson, editors, *Supercomputational Science*, pages 3-12. Plenum Press, New York, 1990.

[83] R. G. Evans and S. Wilson. Supercomputing with novel architectures. In R.G. Evans and S. Wilson, editors, *Supercomputational Science*, pages 13-23. Plenum Press, New York, 1990.

[84] C. Farhat. Parallel computational strategies for large space and aerospace flexible structures: algorithms, implementations and performance. In P. Melli and C.A. Brebbia, editors, *Supercomputing in Engineering Structures*, pages 109-132. Springer-Verlag, Berlin, 1989.

[85] C. Farhat and L. Crivelli. A general approach to nonlinear FE computations on shared-memory multiprocessors. *Comput. Methods. Appl. Mech. & Eng.*, 72:153-171, 1989.

[86] C.A.J. Fletcher. *Computational Galerkin Methods.* Springer-Verlag, New York, Berlin, Heidelberg, Tokyo, 1984.

[87] C.A.J. Fletcher. *Computational Techniques for Fluid Dynamics 1.* Springer-Verlag Berlin Heidelberg, 1988.

[88] R. Fletcher. Conjugate gradient methods for indefinite systems. In G.A. Watson, editor, *Proceedings of the Dundee Biennial Conference on Numerical Analysis*, pages 73–89. Springer-Verlag, 1975.

[89] M. Flynn. Very high speed computing systems. *Proc. IEEE*, 54:1901–1909, 1966.

[90] M. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Comput.*, C-21:948–960, 1972.

[91] D. Funaro, A. Quarteroni, and P. Zanolli. An iterative procedure with interface relaxation for domain decomposition methods. *SIAM J. Numer. Anal.*, 25:1213–1236, 1988.

[92] Y.C. Fung. *Foundations of Solid Mechanics*. Prentice-Hall, Inc., 1965.

[93] T. Furnike. Computerized multiple level substructuring analysis. *Computers and Structures*, 2:1063–1073, 1972.

[94] K.A. Gallivan, R.J. Plemmons, and A.H. Sameh. Parallel algorithms for dense linear algebra computations. In *Parallel Algorithms for Matrix Computations*, pages 1–82. SIAM, Philadelphia, 1990.

[95] W. Gentleman. Some complexity results for matrix computations on parallel processors. *J. ACM.*, 25:112–115, 1978.

[96] R. Glowinski, W. Kinton, and M.F. Wheeler. Acceleration of domain decomposition algorithms for mixed finite elements by multi-level methods. In T.F. Chan, R. Glowinski, J. Périaux, and O.B. Widlund, editors, *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 263–289. SIAM, Philadelphia, 1990.

[97] R. Glowinski, J. Périaux, and G. Terrasson. On the coupling of viscous and inviscid models for compressible fluid flows via domain decomposition. In T.F. Chan, R. Glowinski, J. Périaux, and O.B. Widlund, editors, *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 64–97. SIAM, Philadelphia, 1990.

[98] R. Glowinski and P.Le Tallec. Augmented Lagrangian interpretation of the nonoverlapping Schwarz alternating method. In T.F. Chan, R. Glowinski, J. Périaux, and O.B. Widlund, editors, *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 224–231. SIAM, Philadelphia, 1990.

[99] G.H. Golub and D. Mayers. The use of preconditioning over irregular regions. In R. Glowinski and J.L. Lions, editors, *Computing Methods in Applied Science and Engineering VI*, pages 3–14. North-Holland, 1984.

[100] G.H. Golub and C.F. Van Loan. *Matrix Computations, second edition*. The Johns Hopkins University Press, 1989.

[101] M.A. Gomes-Ruggiero, J.M Martínez, and A.C. Moretti. Comparing algorithms for solving sparse nonlinear systems of equations. *SIAM J. Sci. Stat. Comput.*, 13(2):459–483, 1992.

[102] A. Grammeltvedt. A survey of finite difference scheme for the primitive equations for a barotropic fluid. *Mon. Wea. Rev.*, 97(5):384–404, 1969.

[103] W.D. Gropp and D.E. Keyes. Complexity of parallel implementation of domain decomposition techniques for elliptic partial differential equations. *SIAM J. Sci. Stat. Comput.*, 9(2):312–326, 1988.

[104] W.D. Gropp and D.E. Keyes. Domain decomposition methods in computational fluid dynamics. *Int. J. Num. Met. Fluid.*, 14:147–165, 1992.

[105] W.D. Gropp and D.E. Keyes. Domain decomposition with local mesh refinement. *SIAM J. Sci. Stat. Comput.*, 13(4):967–993, 1992.

[106] J.L. Gustafson. Reevaluating Amdahl's law. *Comm. ACM*, 31:532–533, 1988.

[107] J.L. Gustafson, G.R. Montry, and R.E. Benner. Development of parallel methods for a 1024-processor hypercube. *SIAM J. Sci. Stat. Comput.*, 9:609–638, 1988.

[108] I. Gustafsson. A class of first order factorization methods. Technical Report 77.04 R, Department of Computer Science, Chalmers University of Technology, Göteborg, Sweden, 1977.

[109] I. Gustafsson. A class of first order factorization methods. *BIT*, 18:142–156, 1978.

[110] I. Gustafsson. Modified incomplete Cholesky (MIC) methods. In D.J. Evans, editor, *Preconditioning Methods: Analysis and Applications*, pages 256–294. Gordon and Breach, Science Publishers, Inc., 1983.

[111] F.G. Gustavson. Some basic techniques for solving sparse systems of equations. In D.J. Rose and R.A. Willoughby, editors, *Sparse Matrices and Their Applications*, pages 41–52. Plenum Press, New York, 1972.

[112] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*. Springer-Verlag, 1994.

[113] L.A. Hageman and D.M. Young. *Applied Iterative Methods*. Academic Press, New York, 1981.

[114] W.W. Hager. *Applied Numerical Linear Algebra*. Prentice Hall, 1988.

[115] M.T. Heath, E. Ng, and B.W. Peyton. Parallel algorithms for sparse linear systems. In *Parallel Algorithms for Matrix Computations*, pages 83–124. SIAM, Philadelphia, 1990.

[116] D. Heller. Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems. *SIAM J. Numer. Anal.*, 13:484–496, 1978.

[117] D. Heller. A survey of parallel algorithms in numerical linear algebra. *SIAM Rev.*, 20:740–777, 1978.

[118] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.*, 49:409–436, 1954.

[119] R.W. Hockney. The potential calculation and some applications. *Meth. Comput. Phys.*, 9:135–211, 1970.

[120] R.W. Hockney and C.R. Jesshope. *Parallel Computers*. Adam Hilger Ltd, Bristol, 1981.

[121] D. Houghton, A. Kasahara, and W. Washington. Long-term integration of the barotropic equations by the Lax-Wendroff method. *Mon. Wea. Rev.*, 94(3):141–150, 1966.

[122] K.H. Huebner and E.A. Thornton. *The Finite Element Method for Engineers, Second Edition*. John Wiley & Sons, Inc., 1978.

[123] T.J.R Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1987.

[124] K. Hwang and F. A. Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill Book Company, New York, 1984.

[125] IMSL, Inc. *IMSL User's Manual, Math/Library$^{T.M.}$, FORTRAN Subroutines for Mathematical Applications*, 1987.

[126] K.C. Jea and D.M. Young. Generalized conjugate gradient acceleration of nonsymmetric iterative methods. *Lin. Alg. Appl.*, 34:159–194, 1980.

[127] D.C. Jespersen. Multigrid methods for partial differential equations. In G.H. Golub, editor, *Studies in Numerical Analysis, Volume 24 of Studies in Mathematics*, pages 270–318. Mathematical Association of America, 1984.

[128] D.S. Kershaw. The incomplete Choleski-conjugate gradient method for the iterative solution of systems of linear equations. *J. Comput. Phys.*, 26:43–65, 1978.

[129] D.E. Keyes. Domain decomposition methods for the parallel computation of reacting flows. *Comput. Phys. Comm.*, 53:181–200, 1989.

[130] D.E. Keyes. Domain decomposition: a bridge between nature and parallel computers. Technical Report 92-44, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, Virginia, 1992.

[131] D.E. Keyes and W.D. Gropp. A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation. *SIAM J. Sci. Stat. Comput.*, 8(2):s166–s202, 1987.

[132] D.E. Keyes and W.D. Gropp. Domain decomposition for nonsymmetric systems of equations: examples from computational fluid dynamics. In T.F. Chan, R. Glowinski, J. Périaux, and O.B. Widlund, editors, *Second International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 321–339. SIAM, Philadelphia, 1989.

[133] D.E. Keyes and W.D. Gropp. Domain decomposition techniques for the parallel solution of nonsymmetric systems of elliptic boundary value problems. *Applied Numer. Math.*, 6:281–301, 1989/90.

[134] D.R. Kincaid and L.J. Hayes. *Iterative Methods for Large Linear Systems*. Academic Press, Inc., 1990.

[135] P. Kogge. *The Architecture of Pipelined Computers*. McGraw-Hill Book Company, New York, 1981.

[136] D.A. Kopriva. Computation of hyperbolic equations on complicated domains with patched and overset Chebyshev grids. *SIAM J. Sci. Stat. Comput.*, 10(1):120–132, 1989.

[137] D.A. Kopriva. Domain decomposition with both spectral and finite difference methods for the accurate computation of flows with shocks. *Applied Numer. Math.*, 6:141–151, 1989.

[138] D.A. Kopriva. Spectral solution of inviscid supersonic flows over wedges and axisymmetric cones. *Computers and Fluids*, 21(2):247–266, 1992.

[139] W.M. Lai, D. Rubin, and E. Krempl. *Introduction to Continuum Mechanics*. Pergamon Press, 1978.

[140] J. Lambiotte and R. Voigt. The solution of tridiagonal linear systems on the CDC STAR-100 computer. *ACM Trans. Math. Softw.*, 1:308–329, 1975.

[141] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw.*, 5(3):308–323, 1979.

[142] C. Lazou. *Supercomputers and their Use*. Clarendon Press, Oxford, 1988.

[143] J.M. Levesque and J.W. Williamson. *A Guidebook to Fortran on Supercomputers*. Academic Press, Inc., 1989.

[144] P.L. Lions. On the Schwarz alternating method I. In R. Glowinski, G.H. Golub, G.A. Meurant, and J. Périaux, editors, *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 1–42. SIAM, Philadelphia, 1988.

[145] P.L. Lions. On the Schwarz alternating method II: stochastic interpretation and order properties. In T.F. Chan, R. Glowinski, J. Périaux, and O.B. Widlund, editors, *Second International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 47–70. SIAM, Philadelphia, 1989.

[146] P.L. Lions. On the Schwarz alternating method III: a variant for nonoverlapping subdomains. In T.F. Chan, R. Glowinski, J. Périaux, and O.B. Widlund, editors, *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 202–223. SIAM, Philadelphia, 1990.

[147] J.W.H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. and Applics.*, 11:134–172, 1990.

[148] R.D. Lowry. The parallel processing revolution. In J.R. Kirkland and J.H. Poore, editors, *Supercomputers, A Key to U.S. Scientific, Technological, and Industrial Preeminence*, pages 77-83. Praeger Publishers, 1987.

[149] J. Mandel. Two-level domain decomposition preconditioning for the p-version finite element method in three dimensions. *Int. J. Numer. Methods Eng.*, 29:1095-1108, 1990.

[150] L.D. Marini and A. Quarteroni. A relaxation procedure for domain decomposition methods using finite elements. *Numer. Math.*, 55:575-598, 1989.

[151] A.M. Matsokin and S.V. Nepomnyashchikh. A Schwarz alternating method in a subspace. *Soviet Mathematics*, 29(10):78-84, 1985.

[152] S.F. McCormick. *Multilevel Adaptive Methods for Partial Differential Equations*. SIAM, Philadelphia, 1989.

[153] M.R. Mehrabi and R.A. Brown. Finite-element/Newton method for solution of nonlinear problems in transport processes using domain decomposition and nested dissection on MIMD parallel computers. In D.E. Keyes and J. Xu, editors, *Seventh International Conference on Domain Decomposition Methods in Scientific and Engineering Computing*. AMS series of Contemporary Mathematics, American Mathematical Society, to appear.

[154] J.A. Meijerink and H.A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. of Comp.*, 31(137):148-162, 1977.

[155] J.A. Meijerink and H.A. van der Vorst. Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems. *J. Comput. Phys.*, 44:134–155, 1981.

[156] L.F. Menabrea. Notions sur la machine analytique de M. Charles Babbage. Bibliothèque Universelle de Genève, Série 3, Tome 41, 1842.

[157] G. Meurant. Domain decomposition methods for partial differential equations on parallel computers. *Int. J. Supercomputer Appl.*, 2(4):5–12, 1988.

[158] G. Meurant. Domain decomposition versus block preconditioning. In R. Glowinski, G.H. Golub, G.A. Meurant, and J. Périaux, editors, *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 231–249. SIAM, Philadelphia, 1988.

[159] G. Meurant. Incomplete domain decomposition preconditioners for nonsymmetric problems. In T.F. Chan, R. Glowinski, J. Périaux, and O.B. Widlund, editors, *Second International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 219–225. SIAM, Philadelphia, 1989.

[160] S.G. Miklin. On the Schwarz algorithm. *DAM USSR*, 77(4):569–571, 1951.

[161] K. Miller. Numerical analogs to the Schwarz alternating procedure. *Numer. Math.*, 7:91–103, 1965.

[162] W.L. Miranker. A survey of parallelism in numerical analysis. *SIAM Rev.*, 13(4):524–547, 1971.

[163] K. Morgan, E. Oñate, J. Periaux, J. Peraire, and O.C. Zienkiewicz, editors. *Finite Elements in Fluids: New trends and applications, Proceedings of VIII*

International Conference on Finite Elements in Fluids, Part I and II. Centro Internacional de Métodos Numéricos en Ingeniería, Barcelona, Spain, 1993.

[164] W.E. Nagel and F. Szelényi. Multitasking on CRAY and IBM multiprocessors: concepts and experiences. In J.-L. Delhaye and E. Gelenbe, editors, *High Performance Computing*, pages 133–142. Elsevier Science Publishers B.V., 1989.

[165] R. Natarajan. Finite element applications on a shared-memory multiprocessor: algorithms and experimental results. *J. Comput. Phys.*, 94:352–381, 1991.

[166] I.M. Navon. A survey of finite-element methods in quasi-linear fluid flow problems. Technical Report WISK Report 240, National Research Institute for Mathematical Sciences, Pretoria, South Africa, 1977.

[167] I.M. Navon. Finite-element simulations of the shallow-water equations model on a limited area domain. *Appl. Math. Modelling*, 3:337–348, 1979.

[168] I.M. Navon. *Numerical methods for the solution of the shallow-water equations in meteorology.* PhD thesis, University of the Witwatersrand, Johannesburg, South Africa, 1979.

[169] I.M. Navon. A Numerov-Galerkin technique applied to a finite-element shallow-water equations model with enforced conservation of integral invariants and selective lumping. *J. Comput. Phys.*, 52:313–339, 1983.

[170] I.M. Navon. FEUDX: A two-stage, high-accuracy finite-element FORTRAN program for solving shallow-water equations. *Computers and Geosciences*, 13:255–285, 1987.

[171] I.M. Navon. PENT: A Periodic Cyclic Pentadiagonal System Solver. *Communications in Numerical Methods*, 3:63–69, 1987.

[172] I.M. Navon. A review of finite-element methods for solving the shallow-water equations. In B.D. Schrefler and O.C. Zienkiewicz, editors, *Computer Modeling in Ocean Engineering*, pages 273–279. A.A. Balkema Publishers, 1988.

[173] I.M. Navon and Y. Cai. Domain decomposition of a finite-element model of the shallow water equations. In *Proceedings of the 11th Symposium on Finite Element Methods in South Africa*, pages 1–38. FRD/UCT center for research in computational and applied mechanics, Cape Town, South Africa, 1992. A key-note paper.

[174] I.M. Navon and Y. Cai. Domain decomposition and parallel processing of a finite element model of the shallow water equations. *Comput. Methods Appl. Mech. & Eng.*, 106(1-2):179 -212, 1993.

[175] I.M. Navon and U. Muller. FESW-a finite-element Fortran IV program for solving the shallow-water equations. *Advances in Engineering Software*, 1:77–86, 1979.

[176] I.M. Navon, P.K.H. Phua, and M. Ramamurthy. Vectorization of conjugate-gradient methods for large-scale minimization in meteorology. *Journal of Optimization, Theory and Applications*, 66(1):71 -93, 1990.

[177] I.M. Navon and H.A. Riphagen. An implicit compact fourth-order algorithm for solving the shallow-water equations in conservative law-form. *Mon. Wea. Rev.*, 107:1107–1127, 1979.

[178] I.M. Navon, X. L. Zou, J. Derber, and J. Sela. Variational data assimilation with an adiabatic version of the NMC spectral model. *Mon. Wea. Rev.*, 120:1433 1446, 1992.

[179] B. Neta. Analysis of finite-elements and finite differences for shallow water equations: a review. *Math. Comput. Simul.*, 34:141–161, 1992.

[180] B. Neta and R. Thanakij. Finite element approximation of the shallow water equations on the MasPar. In M.N. Dhaubhadel, M. S. Engelman, and W. G. Habashi, editors, *Advances in Finite Element Analysis in Fluid Dynamics, FED-Vol 171*, pages 47–52. Amer. Soc. Mech. Eng., N.Y., 1993.

[181] B. Neta and R.T. Williams. Stability and phase speed for various finite-element formulations of the advection equation. *Computers and Fluids*, 14:393–410, 1986.

[182] K. Neves and J. Kowalik. Supercomputing: key issues and challenges. In J.S. Kowalik, editor, *Supercomputing*, pages 3–39. Springer-Verlag Berlin Heidelberg, 1989.

[183] A.K. Noor. New computing systems and their impact on structural analysis and design. In P. Melli and C.A. Brebbia, editors, *Supercomputing in Engineering Structures*, pages 1–42. Springer-Verlag, Berlin, 1989.

[184] T.C. Oppe, W.D. Joubert, and D.R. Kincaid. NSPCG user's guide, A package for solving large sparse linear systems by various iterative methods, version 1.0. Center for Numerical Analysis, The University of Texas at Austin, 1988.

[185] J.M. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems*. Plenum Press, New York, 1988.

[186] J.M. Ortega and W.C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, New York, 1970.

[187] J.M. Ortega and C. Romine. The ijk forms of factorization II. Parallel systems. *Parallel Computing*, 7(2):149–162, 1988.

[188] J.M. Ortega and R.G. Voigt. Solution of partial differential equations on vector and parallel computers. *SIAM Rev.*, 27:147–240, 1985.

[189] J.M. Ortega, R.G. Voigt, and C.H. Romine. A bibliography on parallel and vector numerical algorithms. In *Parallel Algorithms for Matrix Computations*, pages 125–197. SIAM, Philadelphia, 1990.

[190] A.T. Patera. A spectral element method for fluid dynamics: laminar flow in a channel expansion. *J. Comput. Phys.*, 54:468–488, 1984.

[191] W. Poole and R. Voigt. Numerical algorithms for parallel and vector computers: an annotated bibliography. *Comp. Rev.*, 15:379–388, 1974.

[192] D.A. Poplawski. Parallel computer architectures. *Applied. Math. and Comput.*, 20(1 and 2):41–51, 1986.

[193] J.S. Przemieniecki. Matrix structural analysis of substructures. *AIAA J.*, 1:138–147, 1963.

[194] A. Quarteroni. Domain decomposition and parallel processing for the numerical solution of partial differential equations. *Surv. Math. Ind.*, 1:75–118, 1991.

[195] A. Quarteroni. Mathematical aspects of domain decomposition methods. In *Proceedings of the First European Congress of Mathematics*, Paris, France, to appear.

[196] A. Quarteroni, F. Pasquarelli, and A. Valli. Heterogeneous domain decomposition: principles, algorithms, applications. Technical Report UMSI 91/183, University of Minnesota Supercomputer Institute, 1991.

[197] A. Quarteroni and A. Valli. Theory and application of Steklov-Poincaré operators for boundary value: the heterogeneous operators case. Technical Report preprint 90-36, Army High Performance Computing Research Center, University of Minnesota, 1990.

[198] J.J. Quirk and U.R. Hanebutte. A parallel adaptive mesh refinement algorithm. Technical Report 93-63, ICASE, 1993.

[199] G. Radicati, di Brozolo, and Y. Robert. Parallel conjugate gradient-like algorithms for solving sparse nonsymmetric linear systems on a vector multiprocessor. *Parallel Computing*, 11:223–239, 1989.

[200] G. Radicati, Y. Robert, and S. Succi. Iterative algorithms for the solution of nonsymmetric systems in the modelling of weak plasma turbulence. *J. Comput. Phys.*, 80:489–497, 1989.

[201] M.M. Rai. A conservative treatment of zonal boundaries for Euler equation calculations. *J. Comput. Phys.*, 62:472–503, 1986.

[202] M.F. Rubinstein. Combined analysis by substructures and recursion. *ASCE J. of the Structural Division*, 93(ST2):231–235, 1967.

[203] Y. Saad. Krylov subspace methods for solving unsymmetric linear systems. *Math. Comp.*, 37:105–126, 1981.

[204] Y. Saad. On the design of parallel numerical methods in message passing and shared memory environments. In A. Lichnewsky and C. Saguez, editors,

Supercomputing, State-of-the-Art, pages 253–274. Elsevier Science Publishers B.V., 1987.

[205] Y. Saad. Krylov subspace methods on supercomputers. *SIAM J. Sci. Stat. Comput.*, 10(6):1200–1232, 1989.

[206] Y. Saad and M.H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869, 1986.

[207] U. Schendel. *Introduction to Numerical Methods for Parallel Computers*. Ellis Horwood Ltd., United Kingdom, 1984.

[208] B.K. Schwarz and B. Wendroff. The relative efficiency of finite-difference and finite-element methods, I. hyperbolic problems and splines. *SIAM J. Numer. Anal.*, 11:979–993, 1974.

[209] H.A. Schwarz. Über einige Abbildungsaufgaben. *Ges. Math. Abh.*, 11:65–83, 1869.

[210] T. Schwederski, D.G. Meyer, and H.J. Siegel. Parallel processing. In V.M. Milutinović, editor, *Computer Architecture: Concepts and Systems*, pages 178–224. Elsevier Science Publishing Co., Inc., New York, 1988.

[211] C. Seitz and J. Matisoo. Engineering limits on computer performance. *Physics Today*, 37(5):38–45, 1984.

[212] J.R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, 1994.

[213] G.L.G. Sleijpen and D.R. Fokkema. BICGSTAB(L) for linear equations involving unsymmetric matrices with complex spectrum. Technical Report 772, Dept. of Math., Univ. of Utrecht, 1993.

[214] B.F. Smith. An optimal domain decomposition preconditioner for the finite element solution of linear elasticity problems. *SIAM J. Sci. Stat. Comput.*, 13(1):364–378, 1992.

[215] S.L. Sobolev. Schwarz algorithm in the theory of elasticity. *DAM USSR*, 4(6):235–238, 1936.

[216] P. Sonneveld. CGS, A fast Lanczos-solver for nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 10(1):36–52, 1989.

[217] J. Steppler, I.M. Navon, and H.-I. Lu. Finite-element schemes for extended integrations of atmospheric models. *J. Comput. Phys.*, 89(1):95–124, 1990.

[218] H.S. Stone. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *J. ACM*, 20:27–38, 1973.

[219] H.S. Stone. Problems of parallel computation. In J.F. Traub, editor, *Complexity of Sequential and Parallel Numerical Algorithms*, pages 1–16. Academic Press, New York and London, 1973.

[220] K. Stüben and U. Trottenberg. Multigrid methods: fundamental algorithms, model problem analysis and applications. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods, Lecture Notes in Mathematics 960*, pages 1–176. Springer-Verlag Berlin Heidelberg, 1982.

[221] P.N. Swarztrauber. Fast Possion Solvers. In G.H. Golub, editor, *Studies in Numerical Analysis, Volume 24 of Studies in Mathematics*, pages 319–370. Mathematical Association of America, 1984.

[222] P.N. Swarztrauber and R.A. Sweet. Efficient FORTRAN subroutines for the solution of separable elliptic partial differential equations. *ACM Trans. Math. Software*, 5:352–364, 1977.

[223] C. Taylor and J.M. Davis. Tidal propagation and dispersion in estuaries. In J.T. Oden, O.C. Zienkiewicz, R.H. Gallagher, and C. Taylor, editors, *Finite Elements in Fluids, Chapter 5*, pages 95–118. John Wiley & Sons, Inc., 1975.

[224] R. Temam. Survey of the status of finite element methods for partial differential equations. In D.L. Dwoyer, M.Y. Hussaini, and R.G. Voigt, editors, *Finite Elements, Theory and Application*, pages 1–33. Springer-Verlag, New York, 1988.

[225] C. Temperton. Algorithms for the solution of cyclic tridiagonal systems. *J. Comput. Phys.*, 19:317–323, 1975.

[226] M.J. Turner, R.W. Clough, H.C. Martin, and L.J. Topp. Stiffness and deflection analysis of complex structures. *J. Aero. Sci.*, 23:805–823, 1956.

[227] H.A. van der Vorst. Bi-CGSTAB: a more smoothly converging variant of CG-S for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13(3):631-644, 1992.

[228] R.S. Varga. Factorizations and normalized iterative methods. In R.E. Langer, editor, *Boundary Problems in Differential Equations*, pages 121-142. The University of Wisconsin Press, Madison, Wisconsin, 1960.

[229] R.S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, N.J., 1962.

[230] D.U. Von-Rosenberg. *Methods for the Numerical Solution of Partial Differential Equations*. American Elsevier, New York, 1969.

[231] C. Vuik. Solution of the discretized incompressible Navier-Stokes equations with the GMRES method. *Int. J. Numer. Methods Fluids*, 16:507–523, 1993.

[232] H.F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM J. Sci. Stat. Comput.*, 9(1):152–163, 1988.

[233] H.H. Wang. A practical method for tridiagonal equations. *ACM Trans. Math. Softw.*, 7:170–183, 1981.

[234] W.H. Ware. The ultimate computer. *IEEE Spectrum*, 9(3):84–91, 1972.

[235] O.B. Widlund. Domain decomposition algorithms and the bicentennial of the French revolution. In T.F. Chan, R. Glowinski, J. Périaux, and O.B. Widlund, editors, *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages xv–xx. SIAM, Philadelphia, 1990.

[236] E.L. Wilson. Finite element analysis on computers with multiple processors. In P. Melli and C.A. Brebbia, editors, *Supercomputing in Engineering Structures*, pages 43–54. Springer-Verlag, Berlin, 1989.

[237] S. Wilson. Numerical recipes for supercomputers. In R.G. Evans and S. Wilson, editors, *Supercomputational Science*, pages 81–107. Plenum Press, New York, 1990.

[238] J. Xu. Iterative methods by space decomposition and subspace correction. *SIAM Rev.*, 34(4):581-613, 1992.

[239] D.M. Young. *Iterative Solution of Large Linear Systems.* Academic Press, New York, 1971.

[240] V. Zakharov. Parallelism and array processing. *IEEE Trans. Comput.*, C-33:45-78, 1984.

[241] P. Zave and W.C. Rheinboldt. Design of an adaptive, parallel finite-element system. *ACM Trans. Math. Softw.*, 5(1):1-17, 1979.

[242] K. Zhu, I.M. Navon, and X. Zou. Variational data assimilation with a variable resolution finite-element shallow-water equations model. *Mon. Wea. Rev.*, 122:946-965, 1994.

[243] K. Zhu, I.M. Navon, and X. Zou. LADFEUDX-A FORTRAN program for variational data assimilation with a finite-element shallow-water equations model. *Computers and Geosciences*, to appear.

[244] O.C Zienkiewicz. Why finite elements. In J.T. Oden, O.C. Zienkiewicz, R.H. Gallagher, and C. Taylor, editors, *Finite Elements in Fluids, Chapter 1*, pages 1-23. John Wiley & Sons, Inc., 1975.

[245] O.C. Zienkiewicz. *The Finite Element Method, third edition.* McGraw-Hill Book Company (UK) Limited, 1977.

[246] O.C. Zienkiewicz and K. Morgan. *Finite Elements and Approximation.* John Wiley & Sons, Inc., 1983.

BIOGRAPHICAL SKETCH

Yihong Cai was born on August 22, 1961 in Shanghai, People's Republic of China. He graduated from Fudan University, Shanghai, China, with a Bachelor's degree in Applied Mathematics and Mechanics in 1983. Then he worked as an assistant professor at department of mechanical engineering in Central South University, Hunan, China, for nine months. He entered a Master's program in solid mechanics at the department of mechanical engineering in Shanghai Maritime Institute, Shanghai, China, in July 1984 and received his Master's degree with honor in Mechanical Engineering in 1987. Thereafter, upon invitation, he served as an assistant professor at the same department for nine months. In August 1988, he came to the United States of America and began his Ph.D. study at the department of mathematics in the Florida State University. He started his intensive research with Prof. Michael Navon in domain decomposition methods and parallel computing techniques in the Spring of 1991 and obtained his Ph.D. degree in Applied and Computational Mathematics in the Summer of 1994. He will continue his research as a postdoctoral associate at Massachusetts Institute of Technology (MIT) in the areas of domain decomposition and parallel processing software development with applications to engineering problems.