1992

Effectiveness of Additive Correction Multigrid in numerical heat transfer analysis when implemented on an Intel IPSC2

James D. Padgett
Portland State University

AN ABSTRACT OF THE THESIS OF James D. Padgett for the Master of Science in Mechanical Engineering presented May 5, 1992.

Title: Effectiveness of Additive Correction Multigrid in Numerical Heat Transfer Analysis when Implemented on an Intel IPSC2.

APPROVED BY THE MEMBERS OF THE THESIS COMMITTEE

[redacted]

Gerald Recktenwald

[redacted]

Herman Migliore

[redacted]

Charles Balogh

[redacted]

Malgorzata Chrzanowska-Jeske

The effectiveness of the Additive Correction Multigrid (ACM) algorithm, a line-by-line Tri-diagonal Matrix Algorithm (TDMA), and simple Gauss-Seidel (GS) iteration in numerical heat transfer analysis is investigated on a conventional single processor computer and on a distributed memory parallel computer. The performance of these methods is studied by solving a two-dimensional, steady heat conduction problem. The execution time

of ACM on a single processor is proportional to the number of unknowns to the 1.5 power. This is in contrast to the execution time of the TDMA for which the execution time is proportional to the number of unknowns to the 2.0 power. The GS , TDMA and ACM algorithms are adapted to a model IPSC2 Intel hypercube which has a 32 processing nodes each with 8 MBytes of local memory. Because GS is a local method, it has almost perfect speed up, but it also converges more slowly than TDMA, The TDMA, on the other hand, is affected by domain decomposition to a greater extent than GS. As the number of processors used to solve the problem is increased, the execution times for GS and TDMA are essentially equal. Solving the model problem with 32 processors on a 192×192 grid resulted in parallel efficiencies of 95%, 80% and 78% for the GS, TDMA, and ACM algorithms, respectively. Though the parallel efficiency of ACM was the lowest of the three, the parallel ACM algorithm required an order of magnitude less time to solve the model than either parallel GS or parallel TDMA without multigrid.

EFFECTIVENESS OF ADDITIVE CORRECTION MULTIGRID IN NUMERICAL HEAT TRANSFER ANALYSIS WHEN IMPLEMENTED ON AN INTEL IPSC2

by

JAMES D. PADGETT

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in
MECHANICAL ENGINEERING

Portland State University
1992

TO THE OFFICE OF GRADUATE STUDIES:

The members of the Committee approve the thesis of James D. Padgett presented May 5, 1992.

[redacted]

Gerald Recktenwald

[redacted]

Herman Migliore

[redacted]

Charles Balogh

[redacted]

Malgorzata Chrzanowska-Jeske

APPROVED:

[redacted]

Graig Spolek, Chair, Department of Mechanical Engineering

[redacted]

C. William Savery, Vice Provost for Graduate Studies and Research

DEDICATION

To my Mom and Dad, who always believed.

ACKNOWLEDGMENTS

I would like to thank Intel Corporation, the Oregon Advanced Computing Institute, and the Tektronix Corporation for access to their computer resources. I would also like to thank my research advisor, Gerry Recktenwald, for always being available for tangential conversations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

$\delta_{l,m}$	Correction to the l,m coarse grid control volume (Eqn 3.13) in additive correction multigrd
φ	General dependent variable
$\varphi_{i,j}$	The value of the dependent variable at the i,j control volume
$\overline{\varphi}_i$	The ith column correction (Eqn 3.8) in block correction algorithm
φ^*	The latest approximation to the solution of the difference equations
$\widehat{\varphi}_{i,j}$	The value of the i,j control volume that has been treated as a constant (Eqn 3.3)
$\varphi^*_{i,j}$	The value of the i,j control volume that has been updated in the same sweep (Eqn 3.2)
Δx	x-direction width of a control volume, Figure 4
Δy	y-direction height of a control volume, Figure 4
δy_e	Distance from the center of a control volume to the center of its eastern neighbor, Figure 4
δy_{e+}	Distance from the center of a control volume to its eastern border, Figure 5
ρ	Fluid density
A	The coefficient matrix of the linear algebra form of the discretized equations
$a^E_{i,j}$, $a^W_{i,j}$, $a^N_{i,j}$, $a^S_{i,j}$	The coefficients that relates the i,j control volume to its neighbors
$\tilde{a}^E_{i,j}$	ACM part of the a^E coefficient. Thus $\tilde{a}^E_{i,j}= 0$ if $\varphi_{i+1,j}$ is inside the coarse grid block

$\hat{a}^E_{i,j}$	ACM part of the a^E coefficient. Thus $\hat{a}^E_{i,j}=0$ if $\varphi_{i+1,j}$ is outside the coarse grid block
$\bar{a}^E_{l,m}$	The a^E coefficient on the coarse level
$a^P_{i,j}$	The discrete coefficient for the i,j point
b	The source term vector of the linear algebra form of the discretized equations
b	Volumetric heat source
$b(x,y)$	Spatially varying distributed heat source
$b_{i,j}$	The value of the discrete heat source at the i,j control volume
c	Specific heat of a solid
e	The error of the linear algebra form of the discretized equations
i	The index in the x-direction, fine grid
j	The index in the y-direction, fine grid
k	Thermal conductivity
k_P	The thermal conductivity of the pth control volume
l	The index in the x-direction, coarse grid
L1	Index of the last x-direction column of grid points in Figure 2. These control volumes have no thickness
m	An arbitrary number of processors
m	The index in the y-direction, coarse grid
m1	Index of the last y-direction of grid points in Figure 2. These control volumes have no thickness
N	The number of unknown control volumes N = (L1 – 2)(m1 – 2)
n	The number of fine grid control volumes captured by one coarse grid control volume
N_s	Number or relaxations between boundary swaps
R	The residual vector of the linear algebra form of the discretized equations
$\bar{R}_{l,m}$	Source term of the coarse level, constructed from the residuals of the fine level in additive correction multigrid

$R_{i,j}$	The residual of the i,j control volume
r	The L_2 norm of the vector residual
T	Temperature
t	Independent time variable
$\mathbf{u}$	Fluid velocity vector

ACRONYMS

ACM	Additive Correction Multigrid
TDMA	TriDiagonal Matrix Algorithm
GS	Gauss Seidel
FAS	Full Approximation Storage
CS	Correction Storage
IPSC2	Intel Personal SuperComputer 2
i386	Intel 80386 processor
i387	Intel 80387 floating point coprocessor
BLIMM	Block Implicit Multigrid Method
SIMPLE	Semi-Implicit Method for Pressure Linked Equations
SIMPLER	SIMPLE Revised

CHAPTER I

INTRODUCTION

The history of computational heat transfer analysis is closely tied to the development of computers. Prior to World War II, numerical methods were in their infancy and most calculations were done by hand [1]. Since that time numerical analysis has grown into its own field of science. Heat transfer, numerical analysis and computer science have become linked together. This research is an investigation of these three sciences. The numerical analysis aspect of this research is the investigation of a numerical method called Additive Correction Multigrid. This method has been implemented on a particular kind of parallel computer called a hypercube. The goal of the research was to determine how well Additive Correction Multigrid on a hypercube was suited to solving the elliptic equations that arise in heat transfer analysis.

This chapter will first discuss the advantages and limitations of computational heat transfer. The second part of this chapter is a discussion of the necessity of supercomputers in some heat transfer problems along with a brief discussion of parallel computers. Finally there is a description of why elliptic problems are important and an identification of the specific contributions of this thesis.

IMPORTANCE OF NUMERICAL ANALYSIS IN HEAT TRANSFER

The best solution to a heat transfer problem would be an analytical solution to the governing equations without any simplifying assumptions. The solution would accurately describes the physical phenomenon. This is not possible because there is only a handful of closed solutions to the governing differential equations of heat transfer.

Another alternative is to perform an experiment. That would require building a scale model, or analyzing the full scale problem. During that experiment the necessary data would have to be unobtrusively gathered. This is often times an expensive alternative because of the cost of building the model. However, sometimes a scale model is a better alternative because scale models have the capability of being more realistic than other alternatives [1]. Analyzing the full scale problem is sometimes preferable if the objective is limited, e.g., finding a single pressure drop. A disadvantage to scale models is that the cost can be prohibitive. Also, matching flow conditions in a model can be quite troublesome.

A third alternative, which is investigated here, is to perform a numerical analysis. A numerical analysis models the problem by reducing the continuous governing equations to set of discrete equations. These nonlinear algebraic equations are then solved on a computer. The biggest advantage of a numerical solution is that it is often a low cost alternative for gaining a quantitative and qualitative solution to a problem.

PRESENT LIMITATIONS OF NUMERICAL ANALYSIS

The introduction of numerical solutions has not removed the need for analytical or experimental solutions. There are some important limitations to the use of numerical methods. When flows become turbulent, it is no longer possible to have a model that is completely free of empiricism [1], and the empirical models introduce additional difficulties. The results must be correlated with actual experimental data to test the validity of assumptions. Complex geometries are also difficult to model numerically. Examples of these would be internal pump flows or air-fuel mixture flow through an engine head. Flows that are anisotropic such as cavitation or reacting flows are also difficult to model.

The convection terms in the Navier Stokes equations are highly nonlinear. It is this nonlinearity that makes it difficult to solve some problems numerically. Nonlinear partial differential equation will also lend themselves to several solutions [17]. There is a required expertise that is needed to decide if the numerical result is physically realistic. There is no single numerical method that can be used to solve all partial differential equations. Each problem must be approached separately. The engineer must have a priori knowledge on how the software operates, on the physics involved and some knowledge about scientific computing.

It should also be mentioned that the difference between these methods has become blurred. Experimental data might be used to find the velocity distribution. This distribution is then used to numerically solve the energy equation to find the heat transfer. All three methods will certainly continue to be used in the future.

PRESENT LIMITATIONS OF ALGORITHMS AND MACHINES

Just a few years ago, computer memory was rare commodity. Programmers spent a large portion of their time reducing the memory needs of their programs. Memory has become cheaper and more available on all platforms. One might think that memory concerns are over. What has happened is with the amount of increased available memory, there has been an increase in the size of problems that can be solved. The amount of data that is generated by these larger models is astounding.

The time to solve these problems scales with problem size as

$$\tau = N^{\gamma}$$

where τ is the solution time, N is the number of grid points and γ is the rate of convergence. The numerical method that has been used in the past by this research group has a rate of convergence of $\gamma = 1.86$. This rate of convergence is not uncommon with numerical solvers. Because of this, problems that require fine spatial resolution (or a large number of time steps) may be impractical to solve numerically with available desktop workstations.

DISTRIBUTED MEMORY COMPUTERS

Many engineering problems can be solved with present workstations. However, there are some problems that require the memory and computational power of a supercomputer. Some examples of these problems are weather, groundwater-pollutant mixing and turbulent mixing in a hypersonic jet engine. Unfortunately supercomputers are economically out of reach of many groups who wish to solve these problems.

Recent advances in the architecture of supercomputers have significantly reduced the price of a particular type of supercomputer. Specifically, there have been advances in the area of parallel supercomputers.

Shared vs. Distributed Memory

There are two types of parallel computers: shared and distributed memory. Both have multiple processors that access memory. The processors of a shared memory computer access the same block of memory. These processors can either access this memory over the same bus or through a switching network. A switching network allows several paths to the same memory. On a distributed memory computer, each processor only has direct access to its own piece of memory. If processor A needs data from processor B, processor

B must stop what it is doing and explicitly send a message to processor A. Likewise processor A will have to stop what it is doing to receive the message.

There is certainly great debate of which paradigm is better. Shared memory computers are easier to program but there is a limit on how many processors they can efficiently have. If the shared memory machine is using a bus based system, this bus becomes a bottleneck that limits the speed at which processors can access memory [19]. Switching networks attempt to avoid this memory access bottleneck by providing many different paths to memory while maintaining a single address space. The disadvantage with switching networks is that multiple memory references can attempt to access the same memory path. Again, this causes bottlenecks that limit the speed of memory access. Because of these inherent limits, typical shared memory systems are limited to the tens of processors.

Distributed memory computers are more difficult to program because data must be explicitly sent between processors. However, distributed memory computers can have number of processors in the tens of thousands. Distributed memory computers were chosen for this research because of the possibility to scale to this many processors in the future. The computer that was used in this research is the Intel IPSC2. The IPSC2 is a distributed memory computer based on the i386 microprocessor and the i387 coprocessor. The machine used in this research had 32 processors, each with 8Mbytes of RAM.

Hypercubes

Each processor in the IPSC2 is part of a larger entity called a node. Each node contains the processor, 8Mbytes of memory, and the electronics that are necessary to perform message communication. The message passing electronics are called a Direct Connect module. These modules control message sending, receiving, and routing [9].

The IPSC2 is called a hypercube. In a hypercube each node is not directly connected to every other node. Instead, each node is connected to nodes in which the binary representation of their address differ by one bit. As an example, consider the eight node hypercube in Figure 1. It has nodes numbered 0 to 7, or in binary form 000 to 111. In this arrangement, node 010 (2) is connected to 110 (6), 000 (0), and 011 (3). Node 010 is not connected to node 100 (4). If information needs to be transmitted from node 000 to 100 it must be routed through other nodes. The user specifies the address of the processor to which a message is to be sent. The details of the routing are transparent to the user.

The hypercube configuration is quite useful. By not using some of the connections, it can be made into a mesh or a ring as shown in Figure 1.

The physical problem is solved on a distributed memory computer by using a domain decomposition method. In this method, the physical domain is broken up into pieces and each piece is given to a processor. The hypercube configuration that is used is a mesh. The physical domain is broken up into pieces that match the shape of the hypercube mesh. Each domain piece is given to the corresponding node in the hypercube mesh. This ensures that adjacent domain pieces are owned by nodes that are adjacent in the hypercube.

LITERATURE REVIEW

There has been a lot of work done using multigrid methods and hypercube computers. It is necessary to mention some of the work used as a starting point for this thesis.

Parallel CFD

Braaten [2] implemented the SIMPLER algorithm of Patankar on the Intel IPSC1, solving two dimensional viscous fluid flow. Braaten found that with a straightforward domain decomposition scheme, the speed up per iteration approached 100% as the grid size was increased. However, the parallel efficiency dropped off as the number of processors increased because of a poorer solution to the pressure correction equation. Braaten corrected this parallel efficiency drop off by implementing a global block correction

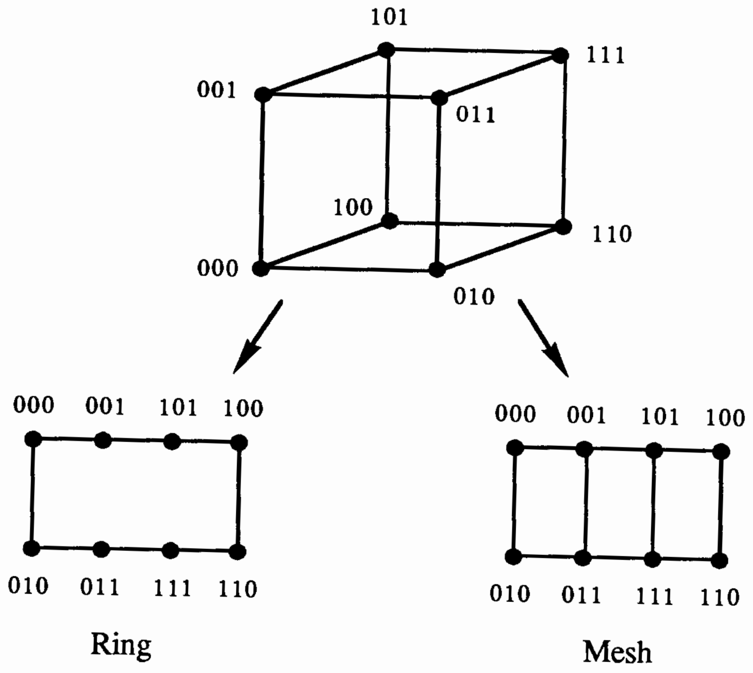


Figure 1. The hypercube configuration for eight nodes (top). The hypercube can also be configured as a ring (bottom left), or a mesh (bottom right).

procedure. The addition of global block correction gave a convergence rate equivalent to the serial algorithm.

Multigrid Methods

Brandt [3] was the first to describe multigrid methods. As the name implies, multigrid methods find the solution to a fine grid problem with the aid of multiple grid levels. These multiple grid levels are of increasing coarseness. These coarser levels are usually constructed so that the coarse grid points are coincident with a regular subset of the fine grids. Brandt developed two types of multigrid, Correction Storage (CS) and Full Approximation Storage (FAS). The coarser grids of CS hold only the corrections to the next finest level. The coarser grids of FAS hold the correction plus the values of the finer grid. Brandt states that FAS is better for nonlinear problems. The operation of going from a fine level to a coarse level is called restriction. The operation of going from a coarse level to a fine level is called interpolation. Brandt's multigrid method has generalized interpolation and restriction operations.

Jespersen [13] gives a good introduction to multigrid methods. Jespersen implements a CS scheme to solve Poisson's equation with uniform grids. This scheme uses a uniform weighting interpolation and restriction operations. Jespersen analyzed the effect of grid cycling on computational efficiency. He mentions that multigrid methods can be expanded to finite element analysis but that is not developed.

Miller and Schmidt [16] paper is a further exploration of Brandt's work. They found that weighted injection schemes did little to aid convergence and nothing to aid accuracy. This work is unconventional in that the relaxation procedure is done after the interpolation operations.

Hutchinson and Raithby [12] extend the Additive Correction method that was derived by Settari and Aziz [21]. Hutchinson and Raithby develop an additive multigrid scheme based on the finite difference grid description of Patankar [17]. They analyze the effectiveness of additive correction multigrid on the solution of anisotropic problems. Our research is the application of this additive correction multigrid method on distributed memory computers, specifically the IPSC2.

Vanka [22] developed the Block Implicit Multigrid Method (BLIMM). BLIMM is a FAS method that is designed to solve the Navier Stokes Equations. BLIMM solves the coupled momentum-continuity equations as opposed to the conventional segregated solution strategy that relies on pressure correction techniques [17]. BLIMM has shown great promise on serial machines. The next area of our research is to implement BLIMM on distributed memory computers.

Multigrid Methods on Distributed Memory Computers

Multigrid methods have been implemented on distributed memory computers. Chan and Tuminaro [8] implement a CS scheme on the Intel Ipsc1 and the Caltech Mark II Hypercube. They used a higher order finite difference scheme for the solution of hyperbolic problems and a parallel preconditioned conjugate gradient method for the solution of elliptic problems. They studied the effect of various cycling schemes on parallel performance.

Briggs et al. [4] developed a CS strategy that is similar to our own except in the domain decomposition. Their method requires some of the processors to 'sleep' at coarser grid levels. They asserted that the performance loss is negligible because of the low computational cost of relaxing a coarse grid. They also concluded that idle processors were even less of concern for large problems where the number of control volumes was much greater than the number of processors.

Hart and McCormick [11] propose the Asynchronous Fast Adaptive Composite (AFAC) grid method as an alternative to the more conventional multigrid algorithms. In this method domain decomposition is not used and different grid levels are concurrently relaxed. Their results showed good performance for most combinations of grid levels and problem sizes. This method was not chosen because it requires sophisticated data and grid management to support the concurrent solving of multiple grids.

ELLIPTIC PROBLEMS

Elliptic problems arise in the solution to the Navier Stokes equations. The pressure field for incompressible flow and the steady heat equation are both elliptic. When the SIMPLE and SIMPLER algorithms of Patankar [17] are used to simulate incompressible flow, the solution to the pressure field must be refined far compared to the other fields. Thus an iterative solver will spend a large portion of its time refining the pressure field.

An important characteristic of elliptic problems is that a small disturbance in one part of the domain has some effect everywhere in the domain [10]. This characteristic makes the solution of elliptic problems inefficient when a domain decomposition method is used [2]. ACM has shown great promise on serial machines when solving elliptic problems. The focus of this research is to develop Additive Correction Multigrid (ACM) on the Intel IPSC2 using a domain decomposition method.

CONTRIBUTIONS OF THIS THESIS

Hutchinson and Raithby [12] applied a multigrid method to the conservative finite difference control volume method of Patankar [17]. This multigrid scheme was based on the Additive Correction method developed by Settari and Aziz [21]. This thesis is a continuation of the work of Hutchinson and Raithby by applying their additive correction multigrid algorithm to the IPSC2. Additive Correction Multigrid has been demonstrated to be an efficient distributed memory algorithm.

This thesis is the first work that has been done by this research group in the area of parallel processing. The first phase of this research group's goals has been accomplished. It has allowed this research group to become familiar with parallel programming and it has created a grid description that will allow future modeling of the convective and unsteady terms of the advection-diffusion equation. The next phase of this research is to implement the BLIMM method of Vanka [22] on the IPSC line of hypercubes.

CHAPTER II

DISCRETIZATION METHOD

INTRODUCTION

This chapter has three parts. The first part is a description of the partial differential equations that govern heat transfer. In the second part, a generalized grid description of a continuous domain is presented. This grid description allows computer modeling of Cartesian, orthogonal and non-orthogonal coordinate systems. This grid description is general enough to allow the modeling of many engineering problems. The third part of this chapter develops the control volume finite difference method. This method allows the heat equation to be modeled within the grid description previously presented.

DIFFERENTIAL EQUATIONS

Advection Diffusion Equation

The equations that govern heat and mass transfer have the general form of the advection-diffusion equation [17]

$$\frac{\partial}{\partial t}(\rho\varphi) + \nabla\cdot(\rho\mathbf{u}\varphi) = \nabla\cdot(\Gamma_\varphi \nabla \varphi) + b \tag{2.1}$$

where t is the independent time, ρ is the fluid density, φ is the dependent variable, $\mathbf{u}$ is the velocity vector, Γ_φ is the diffusion coefficient, and b_φ is the source term. This equation consists of four terms: the unsteady, the convective, the diffusive and the source term respectively. This research is concerned with the solution of elliptic problems. It is the diffusive and the source term that are elliptic. Thus the PDEs that contain the advective and unsteady portion of this equation are not analyzed in this research. For a description of these, the reader is directed to White [23].

Heat Equation

The equation that governs conductive heat transfer in a solid is the heat equation. The convective term vanishes, the dependent variable is temperature and the diffusive coefficient is the material conductivity

$$\rho c \frac{\partial T}{\partial t} = \nabla \cdot (k \, \nabla T) \; + b \tag{2.2}$$

where T is the temperature, c is the material heat capacity, k is the conductivity, and b is the distributed heat source.

For steady heat conduction with uniform conductivity, Equation 2.2 reduces to Poison's equation.

$$k \, \nabla^2 T + b = 0 \tag{2.3}$$

All of the model problems that are investigated in this thesis are governed by Equation 2.3.

DISCRETIZATION

When solving a heat transfer problem, one must first perform an analysis on the problem, hopefully reducing the governing PDE through relevant and accurate assumptions. If at this point a satisfactory analytical solution cannot be made, a numerical solution can be used.

A numerical solution is not an exact solution to the PDE. A finite difference approximation reduces the continuous PDE to a discrete set of algebraic equations. The solution to the discrete algebraic equations does approach the solution of the PDE as the number of grid points approaches infinity if the method is consistent [6]. Thus, it makes sense to solve a relatively fine grid to maintain the characteristics of the PDEs. For very fine grids it becomes unrealistic to solve this set of discrete equations with a direct method. An iterative method is more practical to use.

Discretization Approximations

A numerical solution requires looking at the domain as a discrete set of control volumes rather than a continuum. The following assumptions are made about individual control volumes [17]:

- The values of the scalar properties in the center of the control volume are the values of that property throughout the control volumes
- Any distributed heat source in the control volume is assumed to be uniformly distributed throughout the control volume
- The scalar properties can be discontinuous at control volume boundaries
- Fluxes are uniformly distributed over the face of a control volume, and are normal to that control volume face
- Fluxes between control volumes are conserved
- Sum of fluxes into a control volume must equal the amount of mass that is amassing there

Finite Difference Grid

Figure 2 is a discretized domain in 2-D Cartesian coordinates. We define the i indices to go in the x direction and the j indices to go in the y direction. The black dots in Figure 2 are the centers for each control volume. Notice that there are dots on the boundaries. These are boundary control volumes that have no thickness. In all problems the solver assumes that the problem has Dirichlet boundary conditions. That might appear to be a tremendous limitation but Neumann and Mixed boundary conditions can be modeled by assuming Dirichlet conditions. The procedure for implementing Neumann and Mixed boundary conditions is described in Patankar [17].

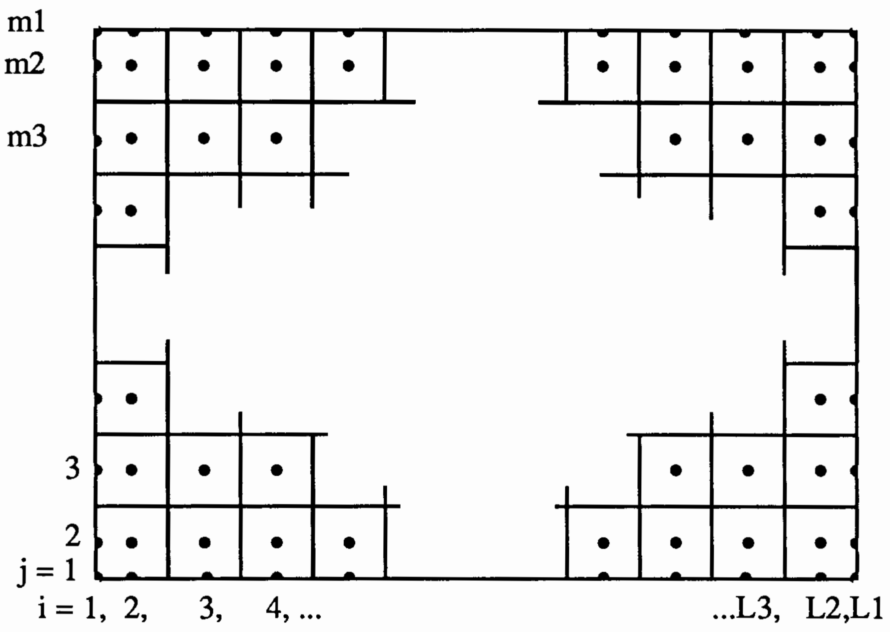


Figure 2. Two dimensional rectangular discretized domain, in Cartesian coordinates.

Define N to be the number of control volumes for which the value is not known. These would be the internal control volumes in Figure 2, because the boundary control volumes are assumed to have Dirichlet conditions imposed.

$$N = (m1 - 2)(L1 - 2) \tag{2.4}$$

This implementation is not restricted to Cartesian coordinate systems. The above grid variables can be put into orthogonal coordinate systems such as axisymmetric and polar as shown in Figure 3. Non-orthogonal coordinate systems can also be modeled. The only restrictions are that shared control volume boundaries be the same size for the two control volumes that share them and that the grid is structured.

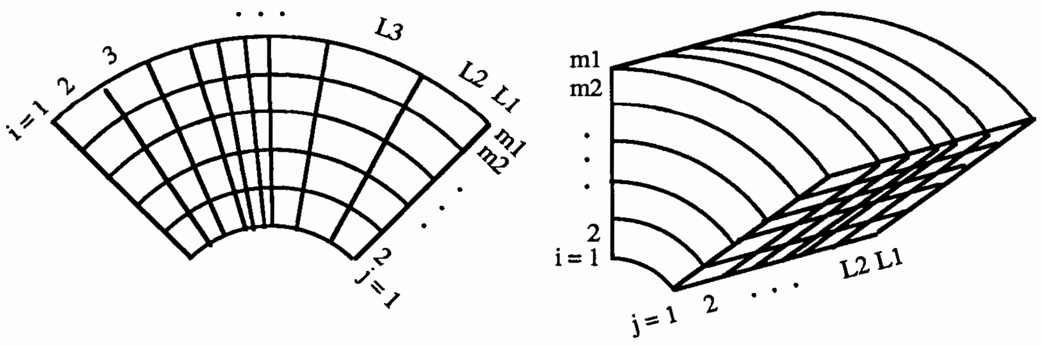


Figure 3. Two dimensional Polar and Axisymmetric discretized grid.

CONTROL VOLUME FINITE DIFFERENCE METHOD

The method that is chosen for discretizing the domain is the conservative control volume finite differencing. This method is chosen because it is a conservative method and thus well suited for the heat equation. Its drawback is that it is difficult to implement on complex geometries. Finite element analysis is better suited for complex geometries. The finite element method is not investigated in this research.

This method is general enough that it also can model the convective and unsteady terms of Equation 2.1. Since this research is only concerned with the diffusive and source term, the discretization of the convective and unsteady terms is not included. The reader is directed to Patankar [17] or Anderson et al [1] for the description of how to discretize the convective and unsteady terms.

Central Differencing

As an example, look at the PDE for steady 2-dimensional heat conduction. The conductivity and the distributed heat source vary spatially only. The variable φ is used instead of T for generality.

$$\frac{\partial}{\partial x}\left(k\frac{\partial \varphi}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\frac{\partial \varphi}{\partial y}\right) + b(x,y) = 0 \qquad (2.5)$$

here the *b* is the distributed heat source. What is needed now is to represent equation 2.5 with a set of algebraic equations.

A single control volume in the grid that was described in Figure 2 is shown in Figure 4.

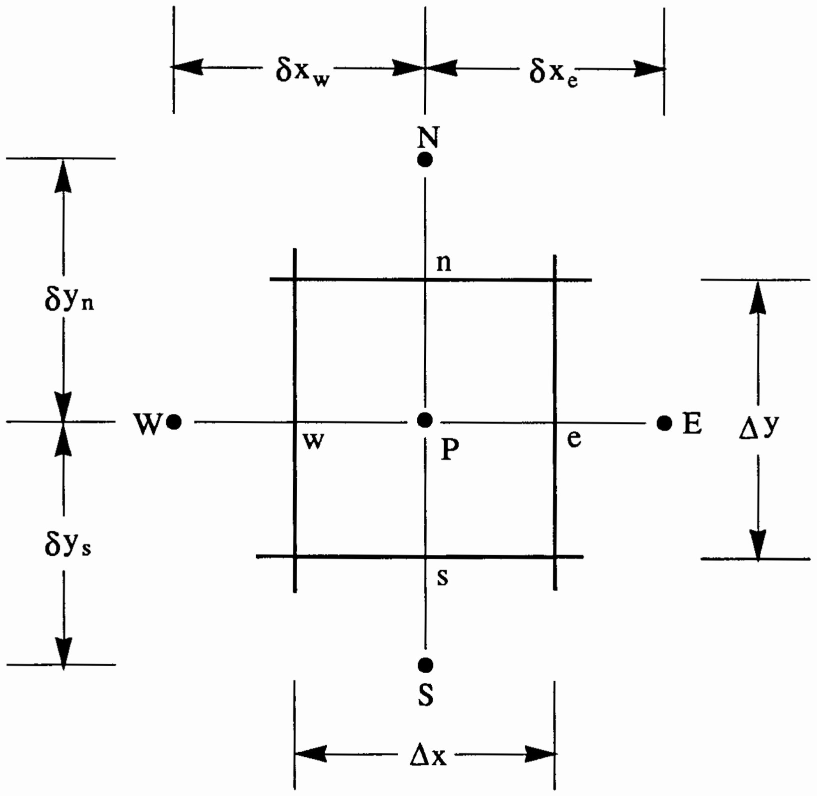


Figure 4. A single control volume from Figure 1.

The upper case letters E, W, N and S are the centers of the control volumes that are east, west, north and south of the center control volume, respectively. The lower case letters refer to quantities that are defined at control volume boundaries. In the case of 2-D heat conduction the only relevant values at the boundaries are the heat fluxes. If convective transport is present, the fluid velocities are also defined at the boundaries [17].

The grid is not necessarily uniform. A uniform grid would be one in which all control volumes have the same width and the same height. The conductivity and source term may vary throughout the domain. However, the conductivity and the source term in a single control volume are uniform throughout the control volume.

Integrate Equation 2.5 over a single control volume.

$$\int_V \left[\frac{\partial}{\partial x}\left(k\frac{\partial \varphi}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\frac{\partial \varphi}{\partial y}\right)\right] \mathrm{d}\mathcal{V} + \int_V b\, \mathrm{d}\mathcal{V} = 0 \tag{2.6}$$

where $\mathrm{d}\mathcal{V} = \mathrm{dx\ dy}$, and $V = \Delta\mathrm{x}\Delta\mathrm{y}$. But since V is the volume per unit depth of the control volume and the function φ is defined to be continuous inside that control volume, the integral can be broken up and the order of integration can be changed to get

$$\int_{s}^{n}\int_{w}^{e} \frac{\partial}{\partial x}\left(k\frac{\partial \varphi}{\partial x}\right)\mathrm{dxdy} + \int_{w}^{e}\int_{s}^{n} \frac{\partial}{\partial y}\left(k\frac{\partial \varphi}{\partial x}\right)\mathrm{dydx} + \int_{s}^{n}\int_{w}^{e} b\mathrm{dydx} = 0 \tag{2.7}$$

Since the source term is defined to be uniformly distributed throughout the control volume its integral becomes $b\Delta\mathrm{x}\Delta\mathrm{y}$. Now take the first term in this equation and evaluate the integral

$$\begin{aligned} \int_{s}^{n}\int_{w}^{e} \frac{\partial}{\partial x} k\left(\frac{\partial \varphi}{\partial x}\right)\mathrm{dxdy} &= \int_{s}^{n}\left(\int_{w}^{e} \frac{\partial}{\partial x}\left(k\frac{\partial \varphi}{\partial x}\right)\mathrm{dx}\right)\mathrm{dy} \\ &= \int_{s}^{n}\left[\left(k\frac{\partial \varphi}{\partial x}\right)_{e} - \left(k\frac{\partial \varphi}{\partial x}\right)_{w}\right]\mathrm{dy} \end{aligned} \tag{2.8}$$

and by the assumption that the fluxes are uniformly distributed over the control volume faces, Equation 2.8 is integrated to get

$$= \left[\left(k \frac{\partial \varphi}{\partial x} \right)_e - \left(k \frac{\partial \varphi}{\partial x} \right)_w \right] \Delta y \tag{2.9}$$

Similarly the second term in 2.8 can be integrated to get

$$= \left[\left(k \frac{\partial \varphi}{\partial y} \right)_n - \left(k \frac{\partial \varphi}{\partial y} \right)_s \right] \Delta x \tag{2.10}$$

Each term in these equations can be discretized to a five point stencil using a Taylor series expansion. This expansion will give an approximation that is accurate to $O(\Delta x^2)$ or $O(\Delta y^2)$ depending on the term. The details of the Taylor series expansion can be found in [17][6]. The Series expansions lead to

$$\left(k_e \frac{\partial \varphi}{\partial x} \right)_e = \frac{k_e(\varphi_P - \varphi_E)}{\delta x_e} \tag{2.11}$$

$$\left(k_w \frac{\partial \varphi}{\partial x} \right)_w = \frac{k_w(\varphi_P - \varphi_W)}{\delta x_w} \tag{2.12}$$

$$\left(k_n \frac{\partial \varphi}{\partial y} \right)_n = \frac{k_n(\varphi_N - \varphi_P)}{\delta y_n} \tag{2.13}$$

$$\left(k_s \frac{\partial \varphi}{\partial y} \right)_s = \frac{k_s(\varphi_S - \varphi_P)}{\delta y_s} \tag{2.14}$$

where k_e is an equivalent conductivity between the center control volume and the east control volume. Insert Equations 2.11 to 2.14 into 2.9 and 2.10 and assemble to obtain

$$a^P \varphi_P = a^E \varphi_E + a^W \varphi_W + a^N \varphi_N + a^S \varphi_S + b \tag{2.15}$$

where

$$a^E = \frac{k_e \Delta \mathrm{x}}{\delta \mathrm{x}_e}$$
$$a^W = \frac{k_w \Delta \mathrm{x}}{\delta \mathrm{x}_w}$$
$$a^N = \frac{k_n \Delta \mathrm{y}}{\delta \mathrm{y}_n}$$
$$a^S = \frac{k_s \Delta \mathrm{y}}{\delta \mathrm{y}_s}$$
$$a^P = a^E + a^W + a^N + a^S$$

This is the finite difference approximation for the single control volume in Figure 4. Obviously there is a similar equation for each control volume in the domain. To keep track of which control volume is being referred to, use a two dimensional array that is the same size as the domain. Each coefficient in equations 2.15 is stored in its own array. In addition, the source term and the φ field have their own arrays. When an arbitrary i,j control volume in Figure 2 is being referenced, the finite difference equation is

$$a^P_{i,j}\varphi_{i,j} = a^W_{i,j}\varphi_{i\text{-}1,j} + a^E_{i,j}\varphi_{i+1,j} + a^S_{i,j}\varphi_{i,j\text{-}1} + a^N_{i,j}\varphi_{i,j+1} + b_{i,j} \tag{2.16}$$

Interface Conductivity

Up to this point there has been no mention about adjacent control volumes with different conductivities. In Figure 5 the a^E coefficient spans both the center control volume and the east control volume. If the east and center control volumes have different conductivities, the definition of k_e needs some consideration. A simple arithmetic mean of

k_E and k_P will lead to erroneous results if the conductivities are orders of magnitude apart [17].

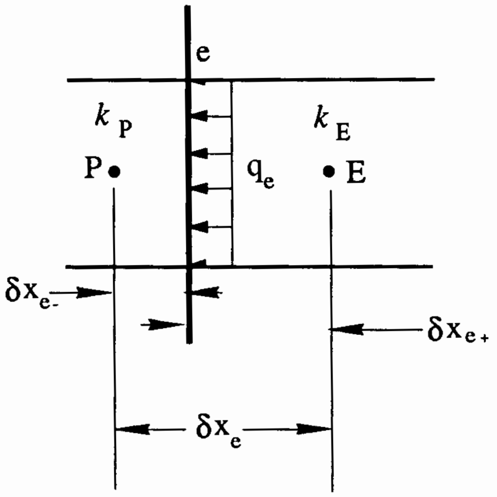


Figure 5. Interface between two control volumes of different conductivity.

What is important is that the flux q_e is correct. That is, the value we choose to use for k_e must reflect the physics that is involved between these two conductivities. The system in Figure 5 can be viewed as two thermal resistances in a series. k_e is just the conductivity part of the equivalent thermal resistance. The equivalent thermal resistance is the sum of the two adjacent thermal resistances

$$\frac{\delta_e}{k_e\,\Delta y} = \frac{\delta_{e+}}{k_E\,\Delta y} + \frac{\delta_{e-}}{k_P\,\Delta y}$$

or

$$k_e = \frac{\delta_e\, k_E\, k_P}{k_E\,\delta_{e-} + k_P\,\delta_{e+}} \tag{2.17}$$

If the interface is midway between the two control volumes the equation for the interface conductivity becomes

$$k_e = \frac{2k_E k_P}{k_E + k_P} \tag{2.18}$$

Equation 2.18 is known as the harmonic mean of k_P and k_E.

CHAPTER III

ADDITIVE CORRECTION MULTIGRID

INTRODUCTION

This Chapter first describes a simple method for solving the difference equations called Gauss Seidel. This method has been in use for a long time and is often considered antiquated. A second method is presented to solve multiple dimensional problems using the tridiagonal matrix algorithm as shown by Patankar [17]. This is the numerical solver that has been used up to this point by this research group. The tridiagonal matrix solver has been aided with a simple multigrid scheme called Block Correction. Block Correction is also described in this chapter.

The additive correction multigrid method that was developed by Hutchinson and Raithby is presented [12]. This method is implemented with the grid cycling scheme that was presented by McCormick [5].

GAUSS SEIDEL

In the previous chapter, the continuous domain was broken up into discrete control volumes. Recall that each control volume has an algebraic equation that links it to its immediate neighbors. Now the challenge is to develop a method to solve these resulting algebraic equations. A simple method, which is called Jacobi, is to take Equation 2.16 and solve for $\varphi_{i,j}$.

$$\varphi_{i,j} = (a_{i,j}^{W}\varphi_{i-1,j} + a_{i,j}^{E}\varphi_{i+1,j} + a_{i,j}^{S}\varphi_{i,j-1} + a_{i,j}^{N}\varphi_{i,j+1} + b_{i,j})/a_{i,j}^{P} \tag{3.1}$$

Equation 3.1 can be applied to every control volume in the domain. A loop might start at the south west corner of Figure 2 and move to the north east corner, applying Equation 3.1 to every control volume. The updated values for $\varphi_{i,j}$ are stored in an intermediate array. After the method has completed an entire sweep, one sweep going from the south west corner to the north east corner, the values of in the intermediate array are put into the φ array. Note that every new value for $\varphi_{i,j}$ depends only on old values of its neighbors.

Holding this intermediate array for the updated φ field is clumsy and requires extra memory. Another way to solve this problem is to eliminate the need for the update array. Instead, change the φ field as the method proceeds. This method, which is called Gauss Seidel (GS), changes $\varphi_{i,j}$ based on values of its neighbors just like Jacobi. In the GS method, the south and west neighbors have already been updated in the same sweep. Equation 3.2 is the finite difference equation for GS

$$\varphi_{i,j} = (a^W_{i,j}\varphi^*_{i-1,j} + a^E_{i,j}\varphi_{i+1,j} + a^S_{i,j}\varphi^*_{i,j-1} + a^N_{i,j}\varphi_{i,j+1} + b_{i,j})/a^P_{i,j} \qquad (3.2)$$

In Equation 3.2, the φ^* term represents a value that has been previously updated in the same sweep. These two methods are beautiful in their simplicity, but they are ineffective at solving large problems. This is because GS and Jacobi are very good at reducing high frequency, local errors but very poor at reducing low frequency, global errors [5]. They are ineffective because information only passes between two control volumes for each sweep. So as the problem becomes large, it takes more sweeps to propagate information through the domain. GS is a little better because updated values for $\varphi_{i,j}$ are based on values that have been updated in the same sweep. Thus information has the possibility to travel more than one control volume per sweep. Because information travels a little faster with GS, GS converges a little more rapidly that Jacobi [6]. The application of one GS sweep is called a relaxation.

TDMA

GS and Jacobi are local methods because each control volume update is based only on its nearest neighbors. A control volume in one part of the domain is not directly affected by things going on elsewhere in the domain. What is needed is a method that updates control volumes based on all or part of the other control volumes in the domain. A method that behaves this way is referred to as a global method. The global method that is used is the Tridiagonal Matrix Algorithm (TDMA) [17]. TDMA is also known as the Line Gauss Seidel (LGS) method, when applied to two and three dimensional problems.

Take the discretized domain from Figure 2 and look at the ith column, as shown in Figure 6. Because of our finite difference approximations, the values of any control volume in that column depend on the values in the two control volumes on either side and the values of the control volumes to the north and south. What can be done is to artificially treat the value of the control volumes on the east and west of the ith column as if they were known. With this assumption, any control volume in the center column depends only on the values of the control volumes to the north and to the south, i.e., by assuming that the values on the i+1 and i–1 columns are known, the field is reduced to a one dimensional problem for the ith column.

Since the values of the control volumes in the adjacent columns are known, they can be moved to the right hand side of the equation. This becomes the new source term. The finite difference equation becomes

$$a^P_{i,j}\,\varphi_{i,j} - a^S_{i,j}\,\varphi_{i,j-1} - an^N_{i,j}\,\varphi_{i,j+1} = \mathrm{b}_{i,j} + a^W_{i,j}\,\widehat{\varphi}_{i-1,j} + a^E_{i,j}\,\widehat{\varphi}_{i+1,j} \qquad (3.3)$$

where $\widehat{\varphi}_{i,j}$ is an adjacent column that is assumed to be known. Apply this equation to every control volume in the ith column. The result is a set of linear equations. This set of

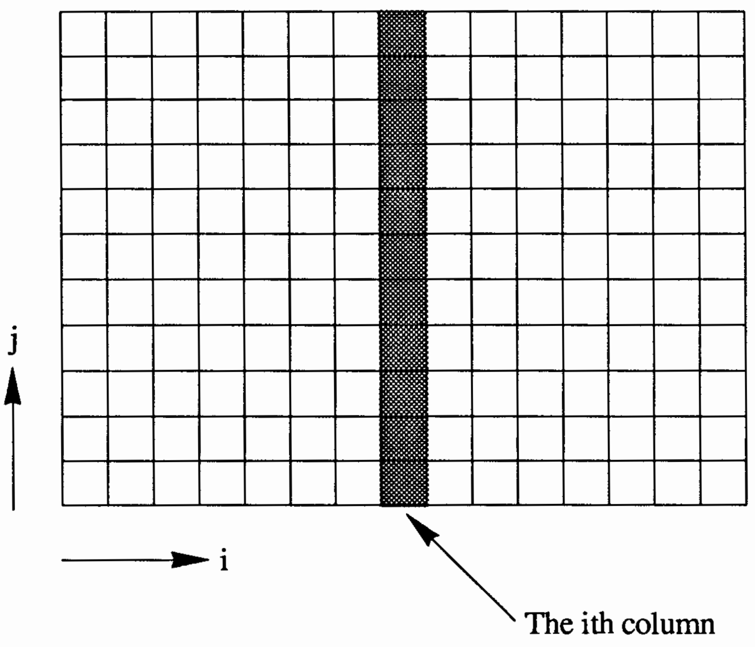


Figure 6. The ith column of the discretized domain.

equations can be put into the more familiar form **Ax** = **b**, where **A** is a tridiagonal matrix that looks like Figure 7.

Fortunately, tridiagonal matrices are very simple and straight forward to solve [6] [18]. The reader should note that the tridiagonal matrix solver is a direct method to solving the system in Figure 7. After the ith column has been solved, boundary information has traveled immediately throughout that column. This is in contrast to GS in which information moves only one control volume every sweep through the entire domain. GS and TDMA are also referred to as point and line solvers respectively. GS is a specific implementation of TDMA. Instead of updating an entire column for TDMA, half a column can be updated, or a forth, or even a single control volume. When we apply Equation 3.3 to a single control volume, the result is GS.

$$\begin{bmatrix} -a^P_{i,2} & a^S_{i,3} & 0 & \cdot & 0 & 0 & 0 \\ a^N_{i,2} & -a^P_{i,3} & a^S_{i,4} & \cdot & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & a^N_{i,m4} & -a^P_{i,m3} & a^S_{i,m2} \\ 0 & 0 & 0 & \cdot & 0 & a^N_{i,m3} & -a^P_{i,m2} \end{bmatrix} \begin{Bmatrix} \varphi_{i,2} \\ \varphi_{i,3} \\ \cdot \\ \cdot \\ \cdot \\ \varphi_{i,m3} \\ \varphi_{i,m2} \end{Bmatrix}$$

$$= \begin{Bmatrix} -b_{i,2} - a^W_{i,2}\,\widehat{\varphi}_{i-1,2} - a^E_{i,2}\,\widehat{\varphi}_{i+1,2} - a^N_{i,1}\,\widehat{\varphi}_{i,1} \\ -b_{i,3} - a^W_{i,3}\,\widehat{\varphi}_{i-1,3} - a^E_{i,3}\,\widehat{\varphi}_{i+1,3} \\ \cdot \\ \cdot \\ \cdot \\ -b_{i,m3} - a^W_{i,m3}\,\widehat{\varphi}_{i-1,m3} - a^E_{i,m3}\,\widehat{\varphi}_{i+1,m3} \\ -b_{i,m2} - a^W_{i,m2}\,\widehat{\varphi}_{i-1,m2} - a^E_{i,m2}\,\widehat{\varphi}_{i+1,m2} - a^S_{i,m1}\,\widehat{\varphi}_{i,m1} \end{Bmatrix}$$

Figure 7. Matrix Form of Equation 3.3 Applied to a single ith column of the discretized domain in Figure 6.

The way TDMA progresses is to start at the i = 2 column. That column is updated by using Equation 3.3 and the tridiagonal matrix solver. The TDMA then updates the next column. This progression continues until the L2 column is updated. The columns are then updated in reverse order. Columns i = 1 and i = L1 are not updated because the values there are assumed to be known. A similar progression is performed for the rows, going from j = 2 to j = m2 and back again. One single application of all of these progressions are referred to as one TDMA relaxation.

ADDITIVE CORRECTION

The coefficients of Equation 2.16 are held in separate arrays to minimize memory requirements. All of those coefficients could be held in a single array, and the system of equations would be in the form

$$\mathbf{A}\varphi = \mathbf{b} \tag{3.4}$$

where **A** is a pentadiagonal matrix. In this system the a^E, a^W, etc coefficients are held in the **A** matrix. The φ field is held in the φ vector and the distributed source term is held in the **b** vector.

This system should not be confused with the system in Figure 7, which is the system that was derived from applying a TDMA relaxation to an arbitrary ith column. It is possible to look at this system of equations in either 2.8 or 3.4 forms. Both forms are algebraically identical. The difference equations are sometimes easier to manipulate if they are in the form of Equation 3.4.

Residual and Error

TDMA and GS are iterative methods. This means that one of these algorithms is continually applied to the last best guess to the domain. The application of one of these methods will create a new best guess that is hopefully closer to actual solution. This iteration process continues until a satisfactory solution is achieved. To qualitatively evaluate what a satisfactory solution is, the residual and error must first be defined.

Define the actual solution to the difference equations to be φ. The field φ is generally never known. All that is known is the last approximation to φ which is designated φ^*.

Once an original guess to φ* is made, TDMA or GS is continually applied to the domain. Hopefully, each time, φ* gets closer to φ. If φ* is put into Equation 3.4, there will be "something" left over. This "something" is defined as the residual vector.

$$\mathbf{R} = \mathbf{b} - \mathbf{A}\varphi^* \tag{3.5}$$

The residual vector represents how far φ* is away from φ. A scalar measure of the size of the residual vector can be found by taking the L_2 norm of **R**

$$r = \| \mathbf{R} \|_2 = \sum_{i=2}^{L2} \sum_{j=2}^{m2} \left(\mathrm{R}_{i,j}^2 \right)^{\frac{1}{2}} \tag{3.6}$$

where

$$R_{i,j} = a_{i,j}^W \varphi_{i-1,j}^* + a_{i,j}^E \varphi_{i+1,j}^* + a_{i,j}^S \varphi_{i,j-1}^* + a_{i,j}^N \varphi_{i,j+1}^* + b_{i,j} - a_{i,j}^P \varphi_{i,j}^*$$

The scalar residual will hence be referred to as simply the residual.

Define the error to be the difference between φ* and φ.

$$\mathbf{e} = \varphi - \varphi^* \tag{3.7}$$

The value of the error is not generally known because it depends on the exact solution to the difference equations. It is useful, however, in the development of additive correction and additive correction multigrid.

Additive Correction

Take Equation 3.7 and solve for φ and substitute into Equation 3.5 to get

$$\mathbf{A}(\mathbf{e} + \varphi^*) = \mathbf{b}$$

which can be rearranged to get

$$\mathbf{A}\mathbf{e} = \mathbf{b} - \mathbf{A}\varphi^*$$

substituting the right hand side with Equation 3.5 to get

$$\mathbf{Ae} = \mathbf{R} \tag{3.8}$$

Equation 3.8 states that the distributed heat source **b** can be replaced by the residual vector that was calculated based on φ^*. If this equation is solved, the result is the error to φ^*, not the solution φ. The error is also be referred to as the correction to φ. Since the correction is now known, it can be added to φ^* to get a better approximation to φ.

BLOCK CORRECTION

What has been described previously are methods to solve the two dimensional problem. It is possible to approximate the two dimensional problem as a one dimensional problem. The reason for doing this is that smaller problems (one dimensional in this case) are computational less burdensome to solve. The idea is to advance the solution while only paying a smaller computational price. The Block Correction Method (BC) advances the solution of the two dimensional problem by solving a one dimensional problem [21].

BC is a multigrid method. As the name implies, multigrid uses multiple grids to advance the solution of the φ field. In the BC method, two grids were used. The first grid is the grid that was developed to solve the φ field. The second grid is a 1-D grid that is used to calculate the correction to the φ field, see Figure 8.

This second grid is constructed from the first grid. As shown in Figure 8, the x direction width of the control volumes in the correction grid (second grid) is the same as the width of the control volumes in the first grid. The y direction width of the control volumes in the correction grids are as long as the y dimension of the domain. A similar correction grid can be constructed for the row corrections.

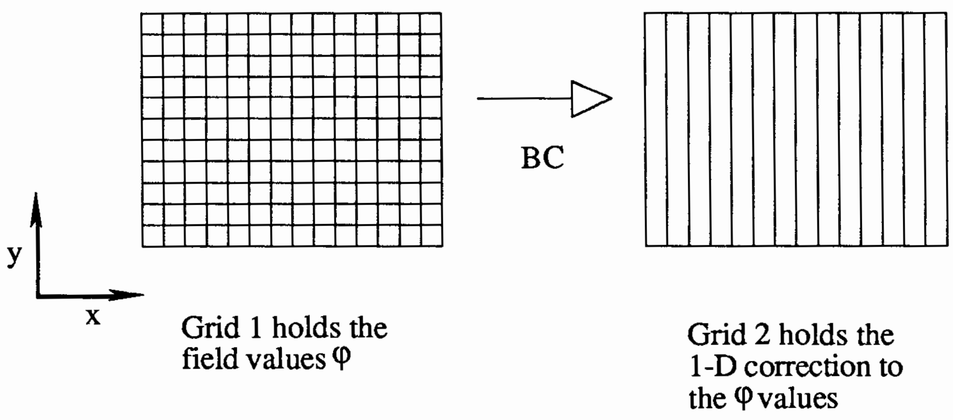

BC
y
x
Grid 1 holds the field values φ
Grid 2 holds the 1-D correction to the φ values

Figure 8. Multiple grids used in the Block Correction Method.

As with the TDMA method, take a single column in the domain in Figure 6. Assume that every control volume in that column can be expressed as the last estimate for that control volume plus some correction for that entire column.

$$\varphi_{i,j} = \varphi^*_{i,j} + \overline{\varphi_i} \tag{3.9}$$

where $\varphi^*_{i,j}$ is the last estimate of $\varphi_{i,j}$ and $\overline{\varphi_i}$ is the correction for the entire ith column. Take Equation 2.16

$$a^P_{i,j}\varphi_{i,j} = a^W_{i,j}\varphi_{i-1,j} + a^E_{i,j}\varphi_{i+1,j} + a^S_{i,j}\varphi_{i,j-1} + a^N_{i,j}\varphi_{i,j+1} + b_{i,j}$$

and sum up over all the control volumes in the ith column. Then insert Equation 3.9 and get

$$\begin{aligned}\sum_{j=2}^{m2} a^P_{i,j}(\varphi^*_{i,j}+\bar{\varphi}_i) &= \sum_{j=2}^{m2} a^W_{i,j}(\varphi^*_{i-1,j}+\bar{\varphi}_{i-1}) \\ &+ \sum_{j=2}^{m2} a^E_{i,j}(\varphi^*_{i+1,j}+\bar{\varphi}_{i+1}) \\ &+ \sum_{j=2}^{m2} a^S_{i,j}(\varphi^*_{i,j-1}+\bar{\varphi}_i) \\ &+ \sum_{j=2}^{m2} a^N_{i,j}(\varphi^*_{i,j+1}+\bar{\varphi}_i) + \sum_{j=2}^{m2} b_{i,j}\end{aligned} \tag{3.10}$$

If terms are collected, Equation 3.10 can be rewritten as

$$\begin{aligned}\bar{\varphi}_i\left[\sum_{j=2}^{m2}(a^P_{i,j}) - \sum_{j=2}^{m2}(a^N_{i,j}) - \sum_{j=2}^{m2}(a^S_{i,j})\right] &= \bar{\varphi}_{i+1}\sum_{j=2}^{m2}(a^E_{i,j}) \\ &+ \bar{\varphi}_{i-1}\sum_{j=2}^{m2}(a^W_{i,j}) + \sum_{j=2}^{m2}(R_{i,j})\end{aligned} \tag{3.11}$$

where $R_{i,j}$ is the residual for i,j control volume and is defined in equation 3.6. Equation 3.11 is the equation for the ith column correction. There is a similar equation for each column in the domain. All these equations can be put into the **Ax** = **b** form, and again, **A** is a tridiagonal matrix. This system can be solved directly for all the $\bar{\varphi}_i$ using the same tridiagonal matrix solver that is used in the TDMA algorithm.

The 2-D problem for φ has been reduced to a 1-D problem for the correction for each column in the domain. A correction for all of the columns can be found directly. Each correction can be added to the last estimate for the entire column to get a better approximation to the solution. An equation similar to 3.11 can be developed for all the rows in the domain. One BC sweep is defined as one i direction sweep (updating the columns) and one j direction sweep (updating all the rows).

ADDITIVE CORRECTION MULTIGRID

Introduction

BC constructed correction grids in an inflexible way. In particular, one correction grid control volume was constructed from an entire row or column of φ grid control volumes. The derivation of Additive Correction Multigrid (ACM) is similar to BC except in the construction of the correction grid and the number of correction levels.

Additive Correction Multigrid

First construct a coarse grid from the fine grids of Figure 2. Capture at most *n* fine grid control volumes (in Figure 9, *n* = 4). The indices of the fine grid are i and j and go in the x and y direction respectively. Let the indices of the coarse grid be l and m, respectively. Figure 9 shows the coarse grid, bold lines, superimposed on the fine grid, fine lines.

The equations for the coarse grids are developed from the fine grid equations. What will be done is to sum up all the fine grid control volumes that lie inside a single coarse grid control volume. To facilitate this, break up the fine grid coefficients into two pieces. For example, the east coefficient can be rewritten as $a^E_{i,j} = \tilde{a}^E_{i,j} + \hat{a}^E_{i,j}$ where $\hat{a}^E_{i,j} = 0$ if $\varphi_{i+1,j}$ is outside the coarse grid block and $\tilde{a}^E_{i,j} = 0$ if $\varphi_{i+1,j}$ is inside the coarse grid block. This is possible because control volume finite element is a conservative method. Assume that *n* fine grid control volumes are captured inside a single coarse grid. Substitute the new definition for the coefficients into the difference Equation 2.16 and get

$$\begin{aligned} a^P_{i,j}\varphi_{i,j} - \tilde{a}^E_{i,j}\varphi_{i+1,j} - \tilde{a}^W_{i,j}\varphi_{i-1,j} - \tilde{a}^N_{i,j}\varphi_{i,j+1} - \tilde{a}^S_{i,j}\varphi_{i,j-1} \\ = \hat{a}^E_{i,j}\varphi_{i+1,j} + \hat{a}^W_{i,j}\varphi_{i-1,j} + \hat{a}^N_{i,j}\varphi_{i,j+1} + \hat{a}^S_{i,j}\varphi_{i,j-1} + b_{i,j} \end{aligned} \tag{3.12}$$

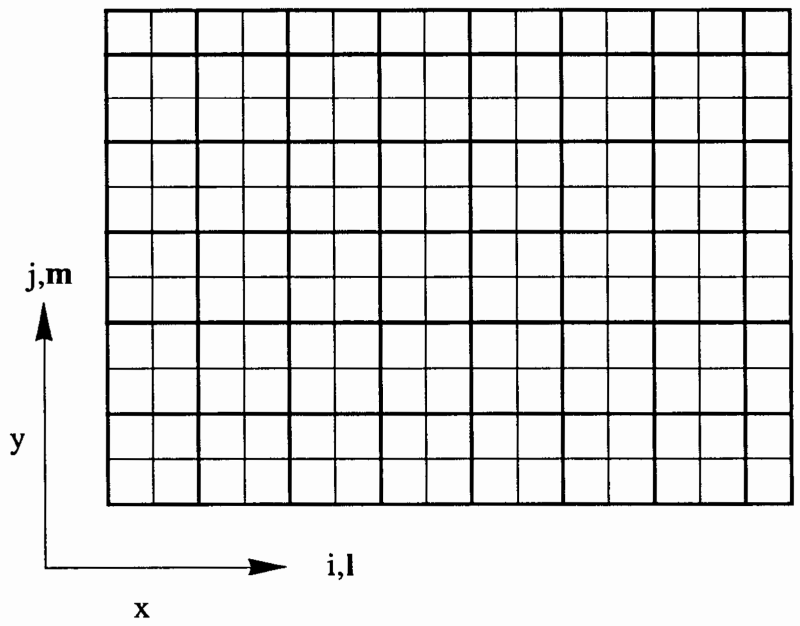


Figure 9. Coarse correction grid superimposed on the fine grid φ field. The bold lines denote coarse grid entities.

If the *n* fine grid control volumes are summed up inside the single coarse grid control volume, the difference equation becomes

$$\sum_n D_{i,j}\varphi_{i,j} = \sum_n \left(\hat{a}^E_{i,j}\varphi_{i+1,j} + \hat{a}^W_{i,j}\varphi_{i-1,j} + \hat{a}^N_{i,j}\varphi_{i,j+1} + \hat{a}^S_{i,j}\varphi_{i,j-1} + b_{i,j}\right)$$

where (3.13)

$$D_{i,j} = a^P_{i,j} - \tilde{a}^E_{i,j} - \tilde{a}^W_{i,j} - \tilde{a}^N_{i,j} - \tilde{a}^S_{i,j}$$

As in the other solver techniques, the values for $\varphi_{i,j}$ are never known. Instead define $\varphi^*_{i,j}$ to be the present best guess to the solution and $\delta_{l,m}$ to be the correction to the entire coarse grid control volume. That is, $\delta_{l,m}$ is the correction for all *n* control volumes that have been captured by the l,m coarse level control volume. They are related by

$$\varphi_{i,j} = \varphi^*_{i,j} + \delta_{l,m} \tag{3.14}$$

which can be inserted into Equations 3.12 to derive the difference equation for the coarse grid control volumes.

$$\bar{a}^P_{l,m}\delta_{l,m} = \bar{a}^W_{l,m}\delta_{l-1,m} + \bar{a}^E_{l,m}\delta_{l+1,m} + \bar{a}^S_{l,m}\delta_{l,m-1} + \bar{a}^N_{l,m}\delta_{l,m+1} + \bar{R}_{l,m} \tag{3.15}$$

$$\bar{a}^P_{l,m} = \sum_n D_{i,j} \qquad \bar{a}^E_{l,m} = \sum_n \hat{a}^E_{i,j} \qquad \bar{a}^W_{l,m} = \sum_n \hat{a}^W_{i,j} \qquad \cdots \tag{3.16}$$

$$\begin{aligned} \bar{R}_{l,m} = \sum_n \big(b_{i,j} &+ \hat{a}^E_{i,j}\varphi^*_{i+1,j} + \hat{a}^W_{i,j}\varphi^*_{i-1,j} \\ &+ \hat{a}^N_{i,j}\varphi^*_{i,j+1} + \hat{a}^S_{i,j}\varphi^*_{i,j-1} - D_{i,j}\varphi^*_{i,j}\big) \end{aligned} \tag{3.17}$$

Equation 3.15 can be solved to obtain a correction to Equation 2.16 using GS or TDMA. If the fine grid has N control volumes, the coarse grid will have approximately N/4 control volumes. So the work to compute a correction to the fine grid takes approximately one quarter of the computational effort of solving the fine grid.

If the coarse grid is still to large, another grid can be constructed on top of it. ACM can be recursive, see Figure 10. Coarse levels are created until there is only one control volume in either the x or y direction. The generated grids are numbered from 1 (finest) to M_L (coarsest).

Brandt [3] and Briggs [5] both give rigorous description of why multigrid works. They both start by looking at the error. The error is the difference between the last estimate to the field and the exact solution. This error can be broken up with a Fourier analysis to a series of periodic functions with variable wavelengths. These functions are called modes. The wavelengths of error modes vary from the size of the grid spacing to the size of the domain. GS is slow to converge because it is only efficient at solving the error modes with wavelengths that are of the order of the grid spacing. Multigrid schemes work because these multiple grids allow efficient relaxing of all the modes of the problem.

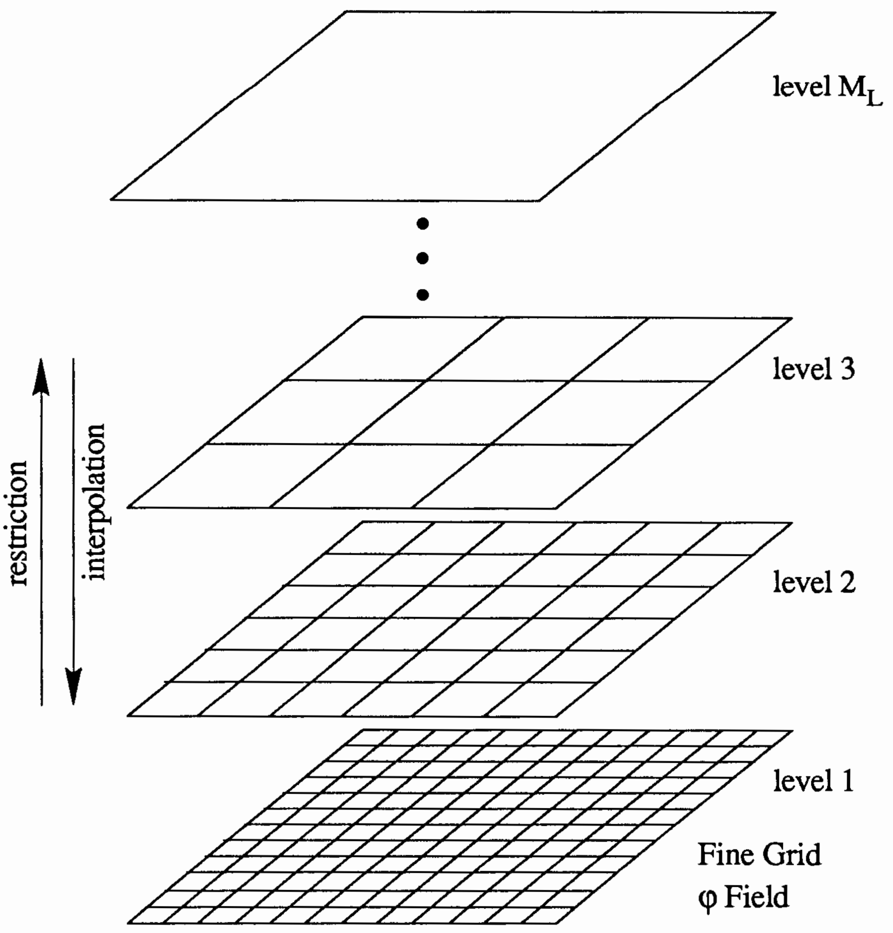


Figure 10. Recursive additive correction grids. Grid levels are labeled from 1 (finest) to M_L (coarsest). The action of going from a finer level to a coarser level is called restriction. The action of going from a coarser level to a finer level is called interpolation.

ACM Implementation

The restriction operation can be broken into two parts: restricting the coefficients and restricting the residual. Equations 3.16 and 3.17 are used to restrict the coefficients and the residual respectively. The action of going from a coarser to a finer level is called interpolation and is done via Equation 3.14. A relaxation operation is defined as a single TDMA sweep. GS is not used to relax the field because it converges slower than TDMA.

A Simple ACM cycle

Imagine a simple ACM progression that contains only two levels. Level one would hold the temperature field. The second level would hold the correction to the temperature field. A single cycle would be organized as follows:

```
Begin Simple ACM cycle:
        restrict coefficients to level 2

        restrict residual to level 2

        relax level 2 $S_c$ times

        interpolate correction to level 1

        relax level 1 $S_f$ times
end Simple ACM cycle
```

Level two could be relaxed to machine accuracy. At first that might seem to be a reasonable thing to do since the computational effort on the second grid will be about a forth compared to level one. However, what is generally done is to only advance the solution on level two by a small amount. Miller and Schmidt [16] found that multigrid schemes were the most computationally efficient if the residual at each level was reduced by approximately one half. They also found that the optimal residual reduction amount was problem dependent.

V and F cycles

McCormick [15] defines two methods for cycling through the domain called F and V cycles. These cycles use all the grids. A V cycle progresses as

```
Begin V cycle(N)
        for n = 1 to N – 1
                restrict residual to level n+1
                relax level n+1 $S_c$ times
        end for

        for n = N – 1 to 1
                interpolate correction to level n
        end for
        relax level 1 $S_f$ times
end V cycle
```

Multiple V cycles can be applied until the residual, Equation 3.6, is reduced by a user specified amount ε. Call this progression an ACM V cycle

```
Begin ACM V cycles:
        $r_0$ = scalar residual of level 1
        $r_{new}$ = 10 * $r_0$
        I = 1

        for n = 1 to $M_L$
                restrict coefficients to level n + 1
        end for

        while ( $r_{new}/r_0$ ) > ε
                V cycle ($M_L$)
                I = I + 1
                $r_{new}$ = scalar residual of level 1
        end while

end ACM V cycles
```

There is another cycling scheme that is defined by McCormick [15] is the F cycle.

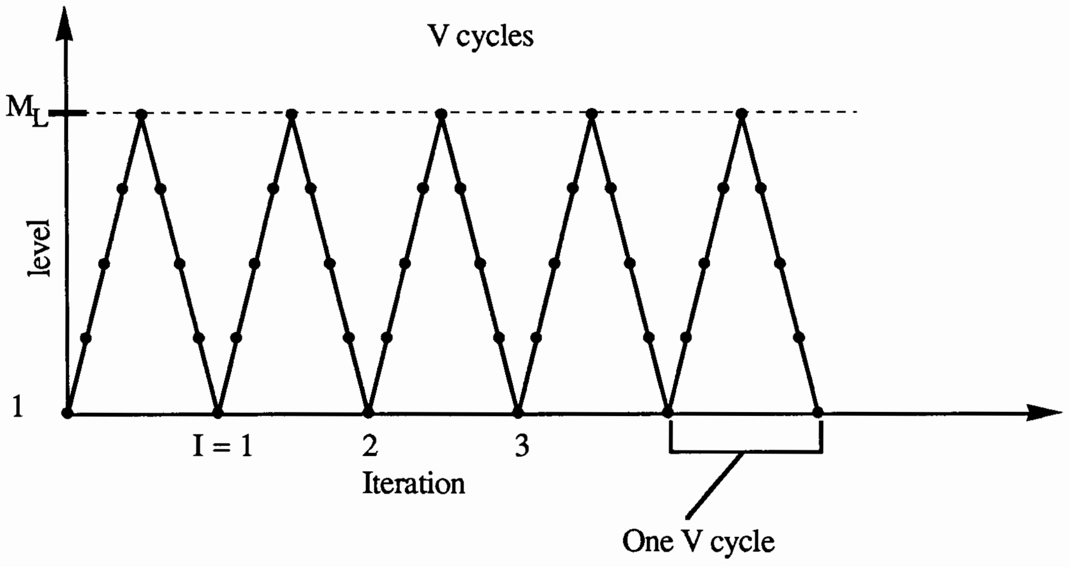
```
Begin F cycle:
        $r_0$ = scalar residual of level 1
        $r_{new}$ = 10 * $r_0$
        I = 1

        for n = 1 to $M_L$
                restrict coefficients to level n + 1
        end for

        while ( $r_{new}/r_0$ ) > ε
                for m = 2 to $M_L$
                        V cycle (m)
                end for
                I = I + 1
                $r_{new}$ = scalar residual of level 1
        end while
end F cycle
```

Both of F and V cycles are shown in Figure 11.

It should be noted that each grid level is relaxed by doing a predetermined number of sweeps. The fine level is relaxed S_f times and all other levels are relaxed S_c times. What would be ideal is to reduce the residual on all levels by an optimal amount [16]. An optimal amount is not known but can be approximated [16]. To reduce the residual by the optimal



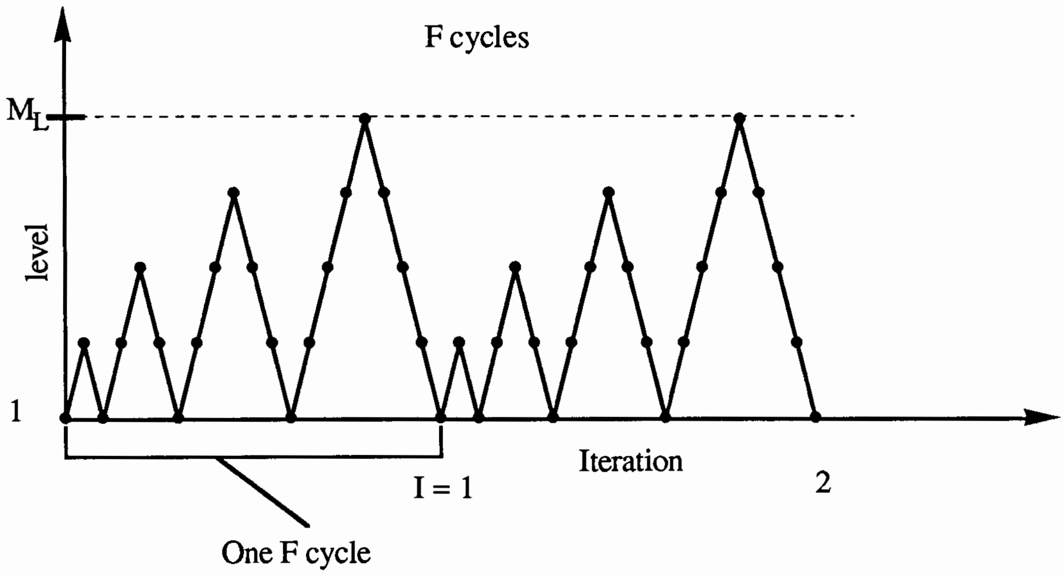


Figure 11. Two common multigrid cycles, F and V cycles. In the above Figures, I is the iteration number.

amount requires calculating the residual at each level, each time that level is relaxed. This is not done because calculating the residual requires a lot of work compared to the effort of relaxing a small amount. So what is gained by having an optimal procedure is lost by monitoring the residual.

CHAPTER IV

SERIAL RESULTS

INTRODUCTION

As discussed in Chapter I elliptic problems are of particular importance in numerical heat transfer analysis. Because of their importance, the bulk of this research is directed at their solution. The first strategy of this research was to implement ACM on serial machines. Serial machines are more familiar than parallel computers and the environment is better for code development. All the results discussed in this Chapter were obtained on a Tektronix XD/88 workstation. After the code was successfully implemented on a serial machine, it was moved to an Intel model IPSC2 hypercube. The second strategy, implementation of ACM on parallel machines, is discussed in Chapter V.

This chapter has three parts. The first part is the description of model problems that are used for comparison of serial and parallel results. The second part is a description on how the code was verified for physical reality. The final part is an analysis of how anisotropic problems affect ACM, TDMA and BC performance.

MODEL PROBLEMS

Model Problem #1

One way to indicate the consistency of ACM is to arbitrarily create an algebraic φ field. This φ field is arbitrary because it does not necessarily correspond to any physical process although it is an exact solution to the Poisson equation. This field can be put into the Poisson equation to derive the distributed source term. This source term is used in the finite difference equations. The φ field that was chosen is

$$\varphi(x,y) = x(1 - x^2)\, y(1 - y)$$
$$\text{defined on} \tag{4.1}$$
$$\Omega = \{(x,y) : 0 \le x,y \le 1\}$$

which is called model problem #1.

This field is chosen because of its elliptic behavior and its convenient Dirichlet boundary conditions. If this equation is put into the heat equation, the result is

$$\nabla^2 \varphi = -[2(x - x^3) + 6x(y - y^2)]$$
$$\text{with boundary conditions} \tag{4.2}$$
$$\varphi = 0 \text{ on } \partial\Omega$$

which gives the source term.

Model Problem #2

The focus of this research is the effectiveness of ACM solving elliptic problems on distributed memory parallel computers. To gage this effectiveness on several platforms, a model problem has been chosen. This problem is steady 2-D heat conduction in a square region as shown in Figure 12. This problem is referred to as model problem #2. The units are arbitrary. This problem was chosen because of its elliptic nature and its convenient Dirichlet Boundary conditions.

Model Problem #3

The problem that Kelkar [14] used to test the efficiency of BC and TDMA on anisotropic fields is heat conduction in a flat media with square embedded heat generating rods. The conduction is two dimensional with Dirichlet boundary conditions. This problem is referred to as model problem #3 and is shown in Figure 13. Figure 14 is an enlargement of the computational domain. On the computational domain, the east, west

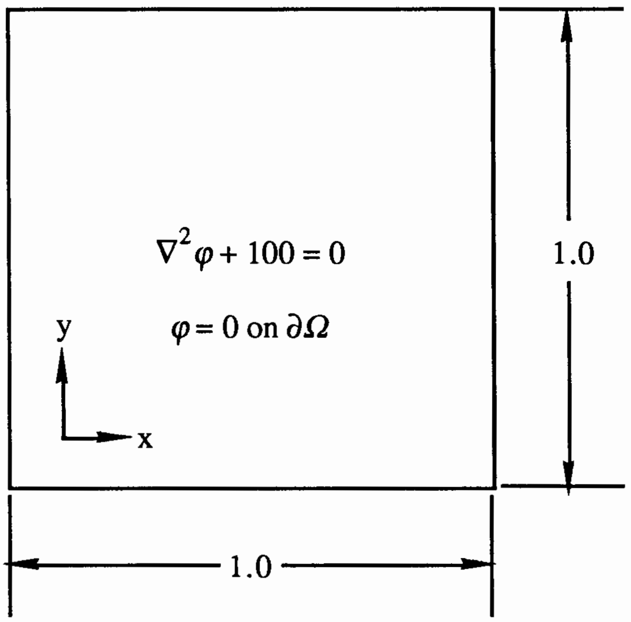

$\nabla^2\varphi + 100 = 0$
$\varphi = 0$ on $\partial\Omega$
y
x
1.0
1.0

Figure 12. Model Elliptic problem #2.

and south boundaries are adiabatic due to symmetry. Kelkar defined the ratio of the conductivities to be

$$c = \frac{k_2}{k_1}$$

where k_1 and k_2 are the thermal conductivities of the two regions in Figure 14.

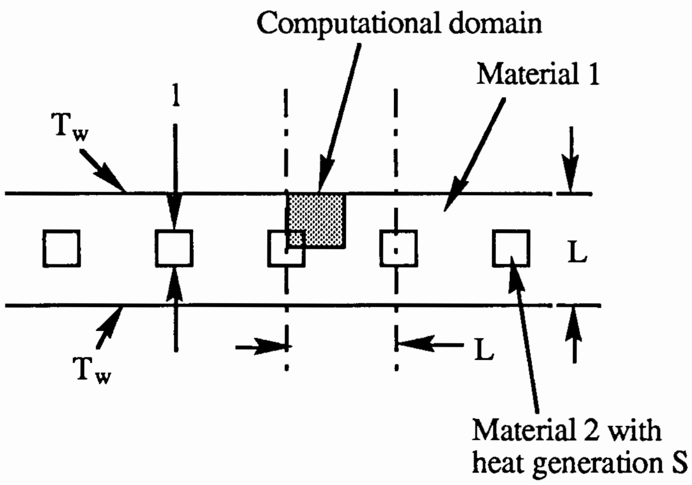


Figure 13. Physical description of model problem #3.

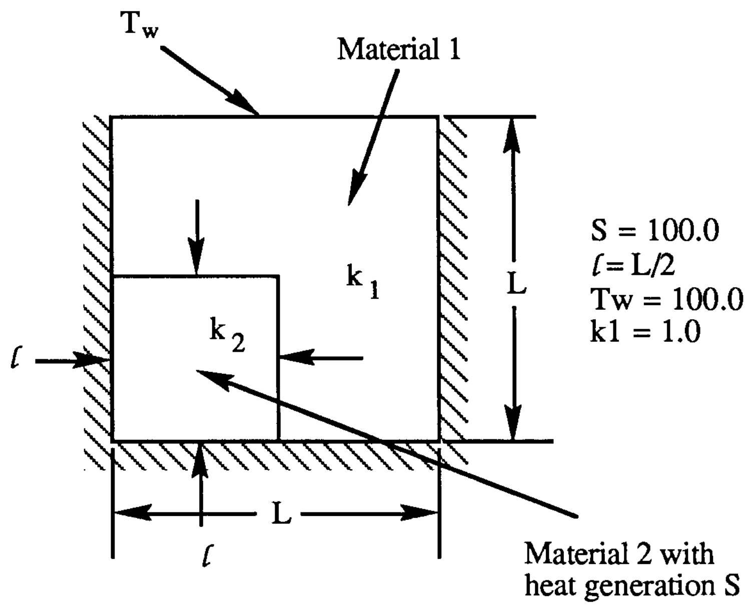


Figure 14. The computational domain of model problem #3.

VERIFICATION OF CODE

The accuracy of the ACM code was verified with a global energy balance and by numerically solving model problem # 1 and comparing the result with the analytical solution. An energy balance is done by summing up the heat transfer from all non-adiabatic boundaries. This amount must equal the amount that is input as a distributed heat source. The energy must balance because control volume finite difference is a conservative method [17]. ACM has shown itself to have an exact energy balance for model problem #1 and all other problems that were solved.

One question to ask, is how well do these difference equations approximate the solution to the PDE. A method is called consistent if the iterated solution approaches the actual solution as the number of control volumes approaches infinity [6]. Rather than formally prove consistency, it is indicated by experiment.

Now the numerical solution that uses Equation 4.2 as a source term will have intermediate field value of φ^* as the numerical solution progresses. The actual analytical solution is defined as φ. The difference between these two fields are reduced to a normalized scalar value, *e*, by taking the L_2 norm of the difference and dividing by the L_2 norm of the exact field.

$$e = \frac{\| \varphi^* - \varphi \|_2}{\| \varphi \|_2} = \left(\frac{\sum_{j=2}^{m2} \sum_{i=2}^{L2} \left(\varphi^*_{i,j} - \varphi_{i,j} \right)^2}{\sum_{j=2}^{m2} \sum_{i=2}^{L2} \left(\varphi_{i,j} \right)^2} \right)^{\frac{1}{2}} \tag{4.3}$$

The scalar value *e* is a normalized value for the difference between the iterated and exact field. This was done for several grid sizes and the results are shown in Figure 15. It can be seen from Figure 15 that the normalized error flattens out after a certain number of iterations. This is the point at which the iterated solution reaches the solution to the discrete finite difference equations to within machine accuracy. As the grid size is made finer, this flat part occurs at a smaller value because the difference equations approach the exact solution. This indicates consistency for the ACM.

COMPUTATIONAL RESULTS

The execution time of ACM can be measured in work units. A work unit is defined as the time it takes to do a single relaxation on the finest level. Brandt [3] proved that with an optimal implementation of a multigrid scheme, the number of work units to solve a problem is independent of problem size. The execution time of a basic TDMA solver can also be measured in work units. With a TDMA solver, the work units will be the number of TDMA relaxations that were required to solve the problem. The work units of TDMA and ACM solving model problem #2 are compared for various grid sizes in Figure 16.

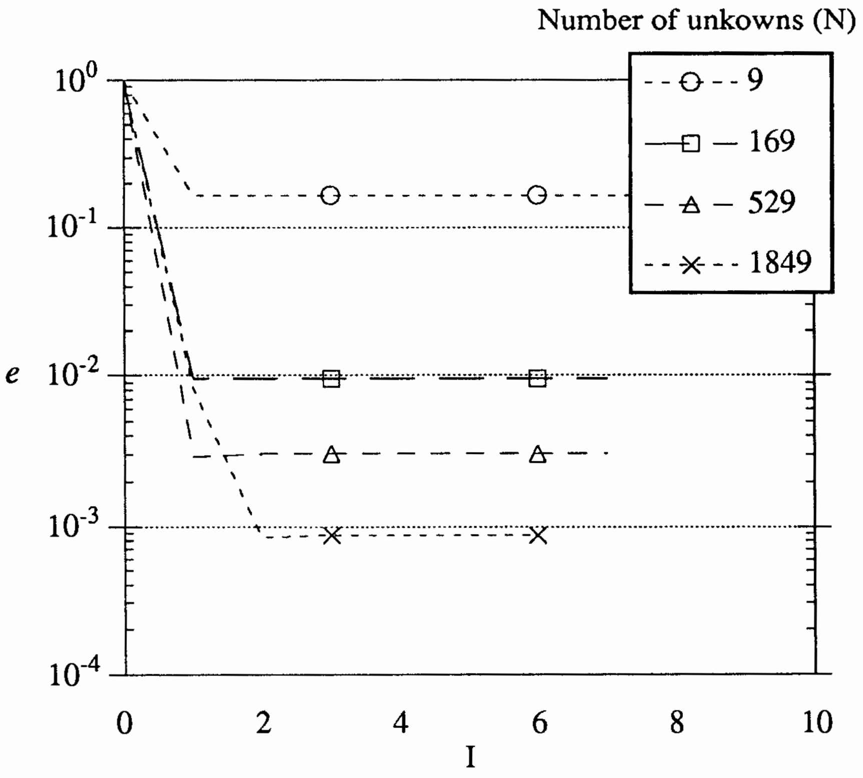


Figure 15. The L_2 Norm of the difference between the numerical and exact solution of Equation 4.2 using ACM, V cycles. I is the number of V cycles performed.

The ACM line should be flat if the number of work units is independent of problem size. However, the number of work units does increase slightly with problem size, indicating that ACM is a sub-optimal form of multigrid. There are two reasons why it is not optimal. The first is that the residual reduction is not monitored on each level. The reduction of the residual at each of these levels is not the optimal amount. The second reason is that the ACM interpolation scheme is not weighted. However, the ACM line is close to being flat compared to the TDMA line. The loss of optimality is fairly negligible when compared with the improvement over basic TDMA.

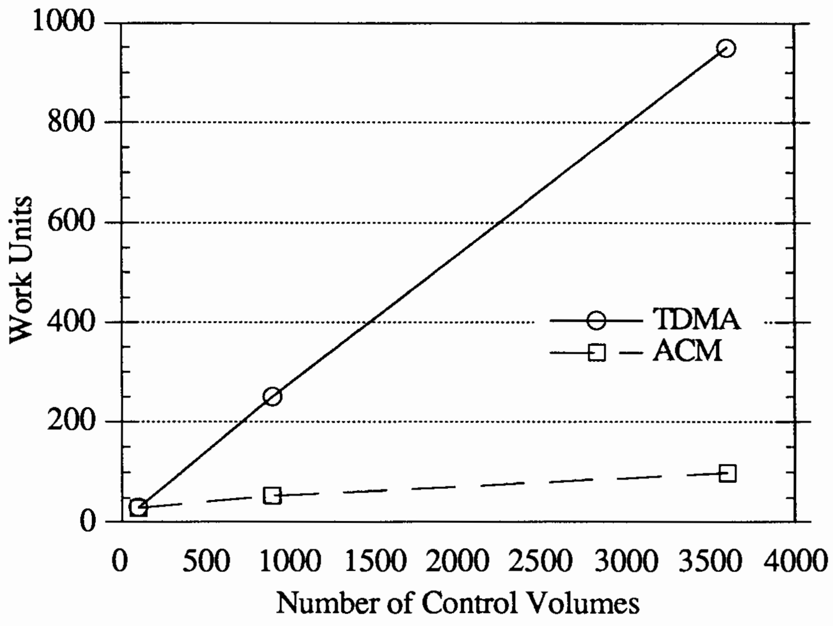


Figure 16. Work Unit convergence rate of ACM and TDMA, solving model problem #2.

The use of work units as a measure of computational efficiency is not as tangible as actual wall clock time. The time to solve the problem and how well that execution time scales with problem size is the most practical measure of computational efficiency. The fact that the number of work units for ACM should be independent of problem size means that the execution time of ACM should be linear in problem size. This is true if the operational count to do one relaxation on the finest grid is proportional to problem size. The time to do one TDMA relaxation is essentially linearly proportional to problem size. Figure 16 is replotted in Figure 17 except with the ordinate measured in execution time.

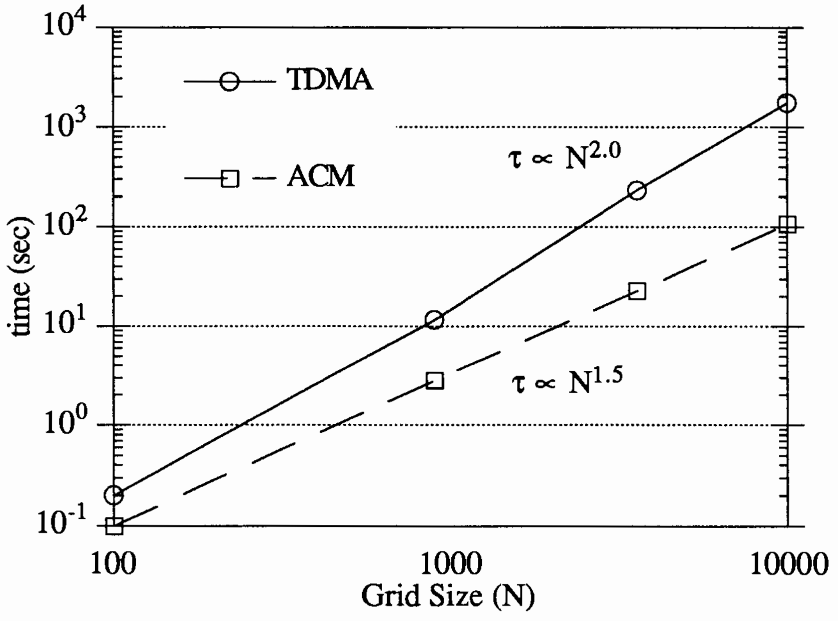


Figure 17. Rate of convergence of ACM versus TDMA solving model problem # 2. The τ in the above figure is the rate at which execution time scales with problem size.

Figure 17 shows that the execution time of ACM is proportional to N with an exponent of 1.5. If the ACM was optimal, this exponent should be 1. The TDMA scales as N squared. The decrease of an exponent of 0.5 might not seem impressive until it is noticed that the execution time of ACM is more that an order of magnitude lower that the TDMA for N = 10,000. Thus, ACM is very efficient compared to TDMA for large N.

The next issue to investigate is the effect of ACM cycling scheme on execution time. F and V cycling schemes were used to solve model problem #2. The execution times are shown in Figure 18.

Figure 18 indicates that there is not a large difference between using V and F cycles. V cycles are favored over the use F cycles because one or two F cycles will drive the solution to machine error. This is actually unwanted for nonlinear problems. With nonlinear problems, the finite difference coefficients are a function of the φ field. A nonlinear problem is solved by iterating the φ field a small amount and then recalculating the coefficients. After the coefficients are recalculated, the field is advanced again. It turns out to be inefficient to advance the solution very far between coefficient updates. Simply, F

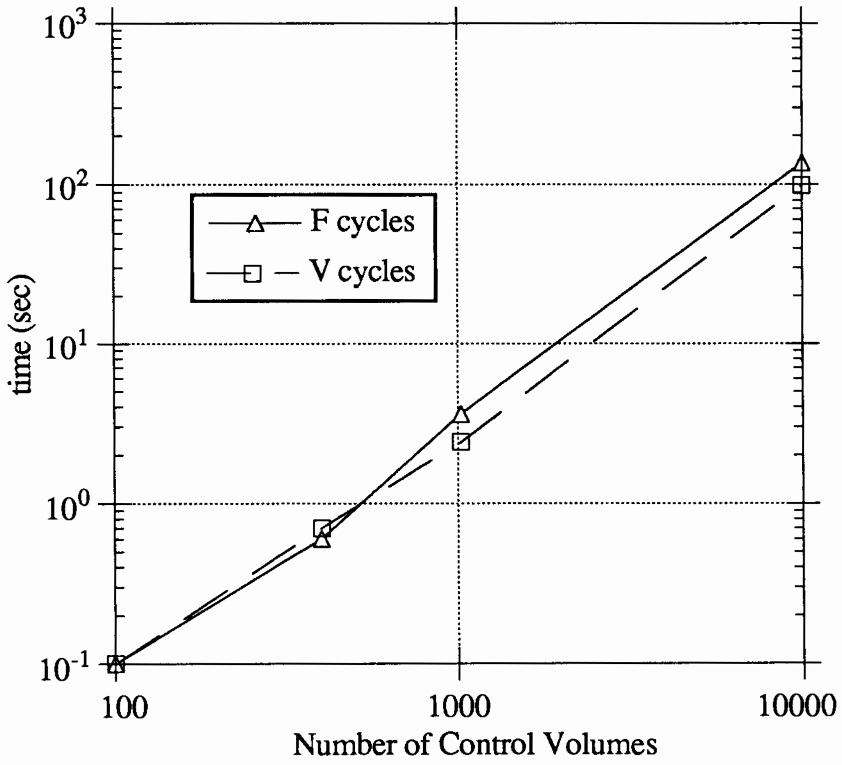


Figure 18. Rate of convergence of ACM using V and F cycles solving model problem 2.

cycles do too much work per iteration for nonlinear problems.

Anisotropic Problems

Heat conduction in materials with large differences in thermal conductivities occur quite often in engineering practice. The convergence of TDMA deteriorates rapidly with an increase in the differences in conductivity. This property of TDMA is investigated by Kelkar [14]. Kelkar also describes how the application of BC significantly increases the convergence rate of TDMA. Since BC is an additive correction method, it would be expected that ACM should perform well on anisotropic problems.

Two values of c were used, c = 1.0 and c = 1.0 x 10^5, for model problem #3. As can be seen from Figure 19, when c = 1.0 (the material is isotropic) TDMA and ACM converge. The convergence of TDMA is aided with the addition of BC. The ACM still drives to the solution more rapidly than TDMA or TDMA with BC.

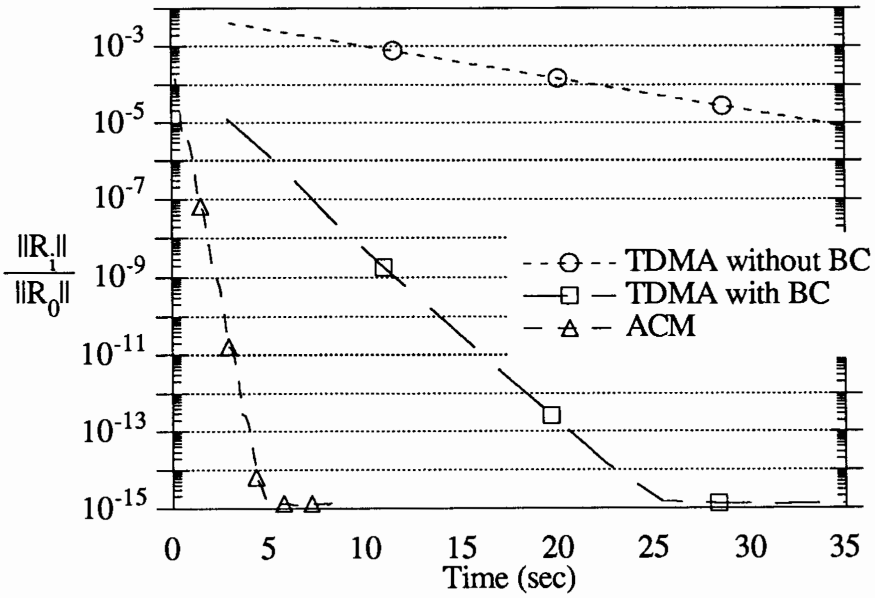


Figure 19. Convergence of model problem #3 when c = 1.0 for ACM, TDMA and TDMA with BC. Solved on a 31 x 31 grid.

However, as shown in Figure 19, the TDMA algorithm stalls when c increases by orders of magnitude. When a BC is added to every fifth TDMA relaxation, the solution converges. Still, the ACM drives to the solution faster. ACM solved both problems in the same number of iterations. This is also true for the TDMA with BC.

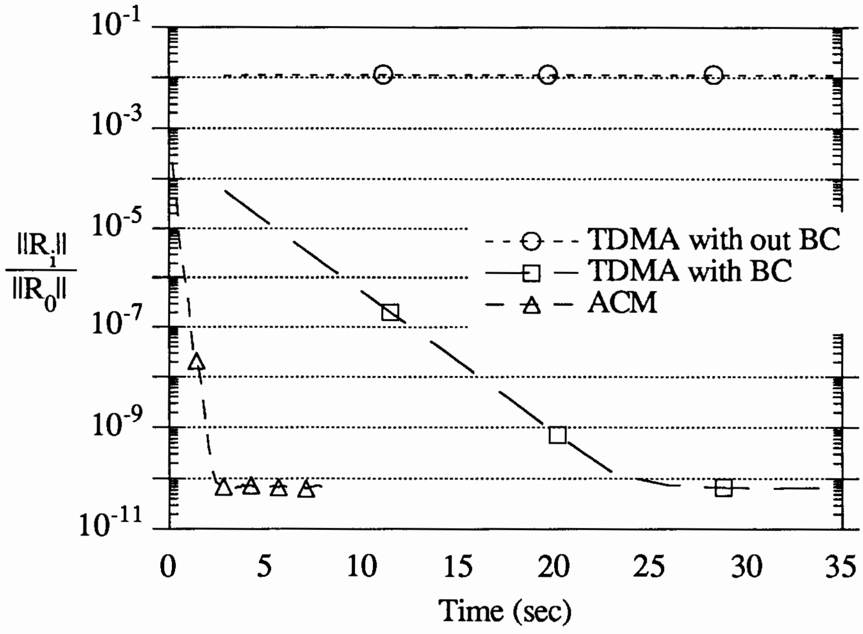


Figure 20. Convergence of model problem #3 when $c = 1.0 \times 10^5$ for ACM, TDMA and TDMA with BC. Solved on a 31 x 31 grid.

A grid size of 31 x 31 was intentionally chosen so that some of the coarse grid control volumes would be constructed from both materials. It might be expected that these coarse grid control volumes would return a correction that would not accurately represent the gradient at the material interface. This may in fact be the case, but the relaxer on the finest level is able to fix that gradient. If the fine grid field is close to the solution, then the residual that gets restricted will be small. Thus the correction that will be interpolated will be small. It appears that ACM (and BC) will not return a correction that will divert the field away from the solution.

CHAPTER V

PARALLEL RESULTS

INTRODUCTION

This chapter has four parts. The first part is a grid description that defines how the finite difference grid is solved on the IPSC2 with domain decomposition. The second part is a description of the parallel implementation of TDMA and GS, along with computational results. The third part is a description of the parallel implementation of ACM, along with computational results. The final part is a discussion of possible areas of future research. All results in this chapter are obtained by solving model problem #2.

The goal of the research is to solve Equation 2.16 over a physical domain using domain decomposition. Domain decomposition requires that data is distributed among the processor nodes used to solve the problem. In this chapter the terms global and local will be used to distinguish between the entire data set and the data resident in the memory of a single node, respectively. The domain that is distributed on a single node will be referred as that node's subdomain.

GRID DESCRIPTION

Globally, the grid is the same as that described in Chapter II. The domain decomposition occurs along control volume boundaries as shown in Figure 21. The light lines denote control volume boundaries and the bold lines are the node's subdomain boundaries. In Figure 21, node 0 owns the control volumes inside its subdomain, and no other.

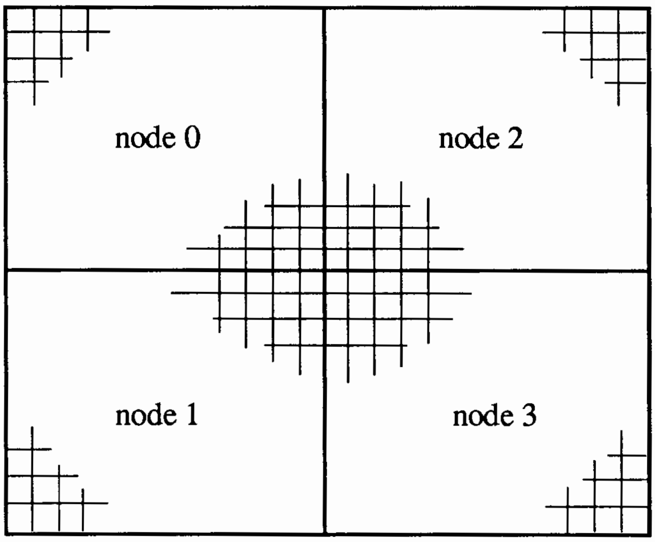


Figure 21. Domain decomposition of the Cartesian Grid that was presented in Chapter III.

Handling Boundaries

The solvers that were presented in Chapter III assume that boundary control volumes are known. These solvers never update the values of the boundaries, instead boundary updates are done between relaxations. See Patankar [17] for a description of handling more complex boundary condition types.

In Chapter III, all the boundaries were "real". In a domain decomposition approach, there are now two types of boundaries, real and shared. Both boundary conditions are treated as Dirichlet. The difference is in the construction of the coefficients. In Figure 21, node 0 has a real west boundary and a shared east boundary. Figure 22 is a more detailed view of those boundaries. The distance from an adjacent control volume to the boundary control volume is different if the boundary is real or shared. This distance is longer for a shared boundary because that distance (δx_S) extends past node 0's boundary into node 1's domain. Thus this longer distance will be used to construct the coefficients on that boundary.

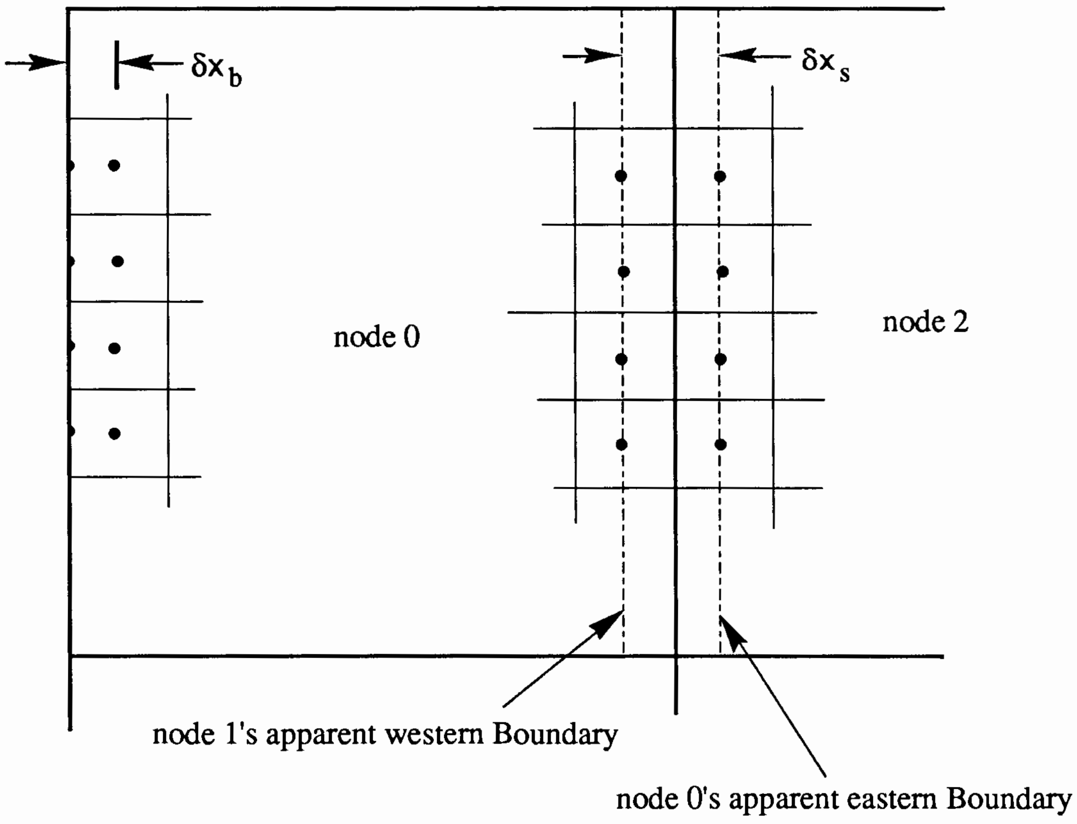

δx_b
δx_s
node 0
node 2
node 1's apparent western Boundary
node 0's apparent eastern Boundary

Figure 22. Node 0's east and west boundaries.

Node 0 owns the control volumes on its western boundary. The western boundary control volumes are handled as “real” boundaries. Node 0’s eastern boundary control volumes are the first column of control volumes in the grid owned by node 2. Node 2 is able to change the value of the control volumes that node 0 thinks are its eastern boundary. After node 2 refines the value of node 0’s boundary values, node 2 will send them to node 0. Likewise node 0 holds the control volumes that node 2 thinks is its western boundary control volumes. Node 0 updates these boundary values and sends them to node 2. Thus, it is a node’s neighbors that update shared boundaries.

Gray Coding

The hypercube can be configured as a mesh as seen in Figure 1. The domain decomposition can be done in such a way that the adjacent domain pieces are owned by nodes that are adjacent in the hypercube. This is called gray coding [7]. This is done because message passing is only necessary between pieces of the domain that are adjacent. If the hypercube is laid out in a gray code fashion, then messages will only be sent between nodes that are nearest neighbors on the hypercube. No messages will have to be routed through intermediate nodes.

PARALLEL IMPLEMENTATIONS OF GS AND TDMA

Swapping Boundaries

Every node that has a shared boundary must send updated boundary information to the node that also shares that boundary. The action of updating boundary information is called a boundary swap.

As the number of nodes increase the swapping of boundaries must be carefully orchestrated. A red-black scheme has been adopted. Nodes that are mapped along the same vertical line are either red or not red. Nodes that are mapped along the same horizontal lines are either black or not black. The domain that is broken among eight nodes looks like Figure 23.

The squares in Figure 23 are subdomains and these should not be confused with domain control volumes. These subdomains are the node's pieces of the global domain. Each node can be both black or red. The scheme progresses as follows: first the east and west shared boundaries are swapped.

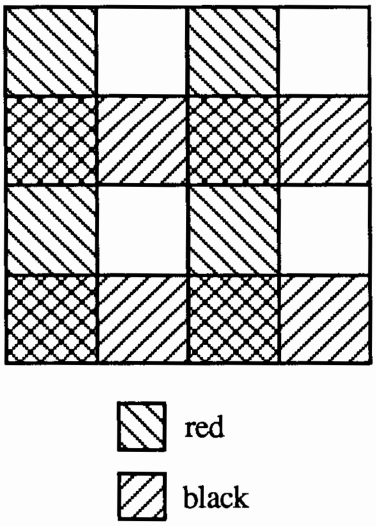


Figure 23. The node mapping for the red-black scheme for eight nodes.

red send east	–	not red receive from west
not red send west	–	red receive from east
red send west	–	not red receive from east
not red send east	–	red receive from west

Then a similar swapping goes on in the north south direction.

black send north	–	not black receive from south
not black send south	–	black receive from north
black send south	–	not black receive from north
not black send north	–	black receive from south

After a boundary swap is completed, each processor with a shared boundary will have new values for that boundary.

Parallel Performance

Performance of a parallel algorithm is measured by speed up and parallel efficiency [19]. Speed up is defined as

$$Su\,(m) = \frac{\tau_1}{\tau_m} \tag{5.1}$$

where τ_1 is the time to solve the problem on one node and τ_n is the time to solve the problem on *n* nodes. Parallel efficiency is defined as

$$\eta(m) = \frac{\tau_1}{m\tau_m} \text{ x } 100 \tag{5.2}$$

Execution time is also important in quantifying the performance of a parallel algorithm.

In general, for a fixed number of finite difference control volumes, as the number of nodes increases, the parallel efficiency decreases. This loss of efficiency is because as the number of nodes increase, the size of each node's subdomain decreases. Each node spends a larger portion of its time in communication rather than in computation.

GS and TDMA

Parallel TDMA and GS versions are similar in implementation. After the hypercube has been allocated, each node is given its domain information. This domain information is the size of the domain, the node IDs of its neighbors and an initial guess for the field. Each node allocates the necessary memory to hold its piece of the domain. Then the execution of GS proceeds as

```
Begin GS:

        set $r_0$ equal the global value of the residual
        $r_{new} = 10*r_0$

        while $(r_{new}/r_0) > \varepsilon$
                for n = 1 to $N_s$
                        relax once with GS
                end for

                swap boundaries
                set $r_{new}$ equal the global value of the residual
        end while

end GS
```

N_S is the number of relaxations that are performed between boundary swaps. The TDMA implementation is analogous except that a TDMA relaxation is performed rather than a GS.

After each boundary swap, each node calculates the value of the residual in its domain. The nodes then globally sum up these residuals. It is important that the global residual is calculated after the boundary swap. To illustrate why the order is important, look at Figure 22. Node 0's eastern boundary is the first column in node 2's domain. Both node 0 and node 2 have a copy of the values in that column. Node 0 cannot change those values but node 2 can. Node 2 changes those values during a relaxation operation. The only time both nodes have the same version of those values is immediately after a boundary swap. For the global residual to be correct, each node must calculate the residual after a boundary swap.

Swaps versus Domain Relaxing

Between boundary swaps, a certain number of relaxations are performed. The optimal number of relaxations between boundary swaps is investigated here. Figure 24 is the speedup of the parallel version of GS, solving model problem #2. The different curves represent various values of N_S The top dotted line represents the line of perfect speed-up which is never attained.

Figure 24 shows that it is important to do a minimal amount of work between boundary swaps. What happens is that the values for the shared boundaries are erroneous until the solution is finally obtained. If a number of relaxations are performed between boundary swaps, a lot of computational work is wasted refining the field based on bad shared boundary values.

Computation versus Communication

Figure 24 shows another important aspect of parallel programming. The speedup for this problem was poor. With 32 processors the best speed up is 12.4. That gives a parallel

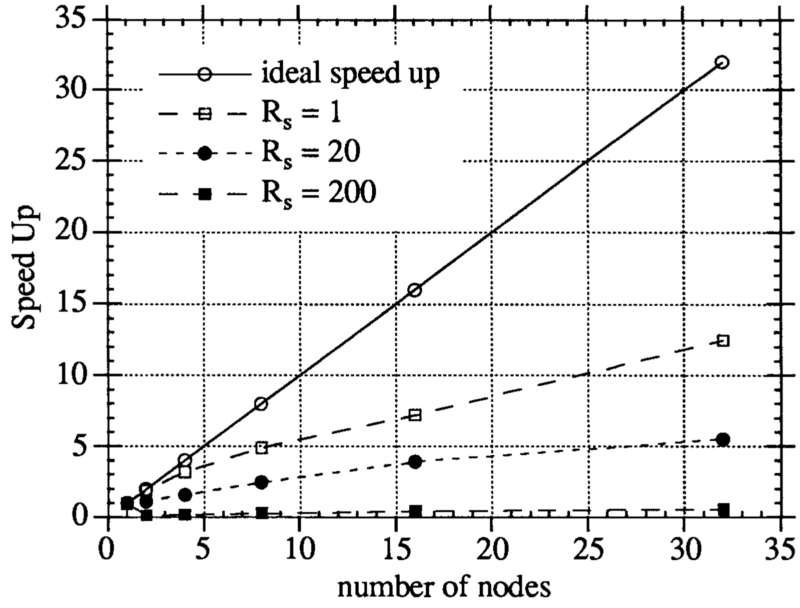


Figure 24. Parallel GS on a 48 x 48 grid solved on the IPSC2. The problem being solved is model problem #1.

efficiency of 38.8 percent. This poor performance is because the problem is too small for the number of processors used. The time to communicate is quite large compared to the time to execute processor instructions. The time to communicate is proportional to the length of the domain. The time to relax the grid is proportional to the area of the node's domain. By solving a larger problem the ratio of area to length is increased. Thus, the ratio of computation time to communication time will decrease. The number of processors used should be based on the size of the problem. The domain was broken up so that each piece of the domain is as square as possible. This will maximize the ratio of domain area to domain length. Despite the loss of parallel efficiency the 32 node solution with $R_s = 1$ took the least amount of wall clock time.

Model problem #2 was solved again on the IPSC2, but this time on a 192 x 192 grid and with TDMA. There was one relaxation per boundary swap. Figure 25 shows that for large problems, GS parallelizes almost perfectly. This is because GS is a local method. Each control volume is updated by its neighbors only. GS is almost immune to the effect of domain decomposition. The TDMA has poorer speedup. The loss of parallel efficiency for the TDMA is because it is a global method. A TDMA relaxation updates an entire column or row. If it is done on one node, this column or row spans the entire domain. One TDMA relaxation will cause information that reflects both boundaries to travel through the domain. By implementing domain decomposition, this boundary to boundary information transfer is severed. The TDMA relaxations are now done on columns or rows that touch shared boundaries. The TDMA refinements are less accurate because shared boundaries may not be accurate. Thus domain decomposition affects the parallel efficiency of global methods to a greater extent.

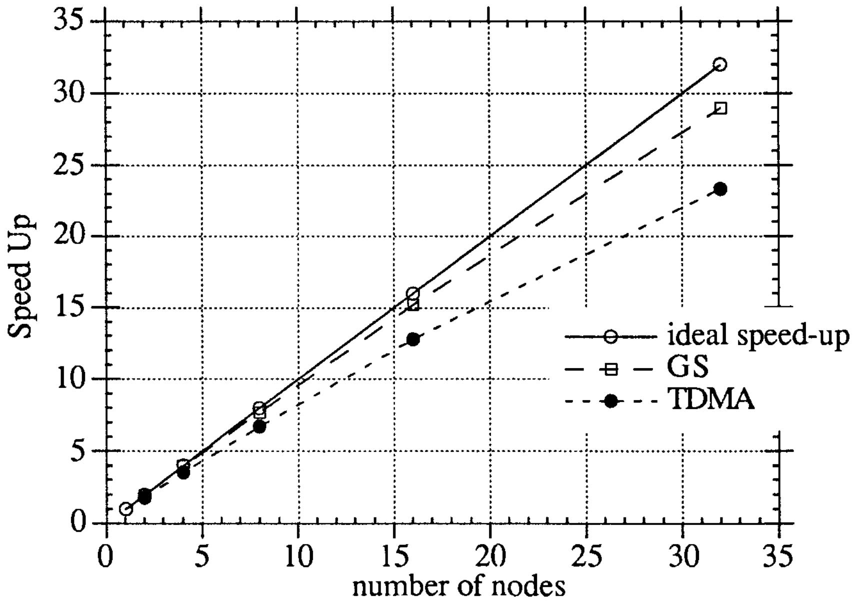


Figure 25. Model problem #2 solved with parallel GS and TDMA on a 192 x 192 grid. These results are from a 32 node IPSC2.

Table I is a summary of the execution times for the runs that were made in Figure 25. Both TDMA and GS took a very long to solve the problem. On one node, the TDMA was 20 percent faster than GS. As the number of nodes increased, the domain decomposition took its effect on execution time of the TDMA. At 32 processors, the execution times were approximately equal for TDMA and GS.

On serial machines it makes sense to use TDMA over GS because for single processors, TDMA is much more efficient. Table I shows that for model problem #1, the benefit of TDMA is lost due to domain decomposition. So for larger number of processors, it would be worth investigating GS as a relaxer.

Another issue that was not investigated in this research is the application of these methods on a hypercube with cache memory and vector processing, specifically the i860 which is the next generation of Intel hypercube. The GS can be written in a one dimensional data structure so it should vectorize well and take advantage of cache memory [20]. GS might offer significant performance over TDMA for these machines.

TABLE I

EXECUTION TIME FOR PARALLEL GS AND TDMA, SOLVING MODEL PROBLEM #2 ON THE IPSC2 WITH A 192 X 192 GRID

	GS		TDMA	
Num. Nodes	Time (hrs)	Su	Time (hrs)	Su
1	25.7	1	20.4	1
2	12.9	2	11.2	1.8
4	6.2	4	5.8	3.6
8	3.3	7.7	3.1	6.7
16	1.7	15.2	1.6	12.8
32	0.89	29.0	.88	23.3

PARALLEL IMPLEMENTATIONS OF ACM

Parallel Grid Definition of ACM

The grid definition for ACM is the same as for the serial implementation. The fine grid is distributed among the nodes like GS and TDMA. Each processor constructs all higher grid levels from the finest grid level. In other words, each node owns the coarse level grids that are constructed from its fine level control volumes as depicted in Figure 26. The boundaries of the all coarse grids occur at the same place in the domain as the fine grids. Coarse grids are constructed until there is only one control volume in the x or y direction, in each subdomain. TDMA is used as the relaxer.

Each node was given the same number of fine grid control volumes. This was done so that the nodes would construct the same number of coarse grid levels. Giving each node the same size problem insured load balancing among the nodes.

The parallel version of ACM progresses like its serial counterpart. It can either perform F or V cycles with a predefined number of relaxations at each level. A predefined number of relaxations insured that no processor would finish first and be idle. No relaxing is done after an interpolation except at the finest level.

What constitutes a relaxation operation is different from the serial ACM. It was found that the best convergence from GS and TDMA occurs when one relaxation is done between boundary swaps. Because of this, one relaxation operation now refers to the application of a single TDMA relaxation operation and a boundary swap.

There are two values that affect the ACM execution. These values are the number of relaxations done on the fine level and the number of relaxations that are done on all the other levels. The optimal number of relaxations depend on the number of nodes. This is because there is an optimal amount for which the residual should be reduced on every level. As the problem is solved on a larger number of nodes, it takes a greater number of relaxations to reduce the residual by this amount.

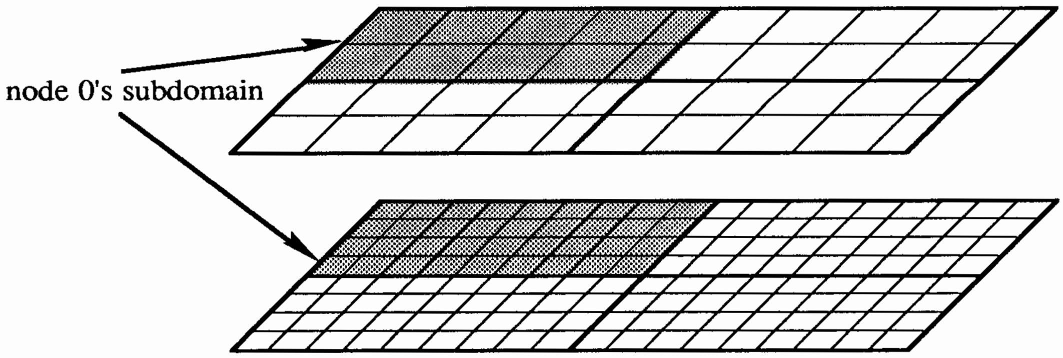


Figure 26. Grid description of parallel ACM. Each node owns all coarse level control volumes that are coincident on its fine grid control volumes.

The definition of speedup is based on the fastest time for a single processors. To compute the speedup of ACM, it is necessary to make several runs for each number of nodes. The number of relaxations between boundary swaps on each grid is varied for these runs. The best execution time for each set of nodes can then be compared.

The speed up for ACM running V cycles on the 192 x 192 problem are shown in Figure 27. It can be seen that the speedup are about the same as the TDMA. Table II has the wall clock time for ACM, TDMA and GS all solving the 192 x 192 problem. The execution time of ACM is more than an order of magnitude lower than the TDMA or GS.

TABLE II

EXECUTION TIME FOR PARALLEL GS, ACM AND TDMA, SOLVING MODEL PROBLEM #2 ON THE IPSC2 ON A 192 X 192 GRID

Num. Nodes	Execution Time (hrs)		
	GS	TDMA	ACM
1	25.7	20.4	1.9
2	12.9	11.2	1.0
4	6.2	5.8	.62
8	3.3	3.1	.30
16	1.7	1.6	.15
32	0.89	.88	.092

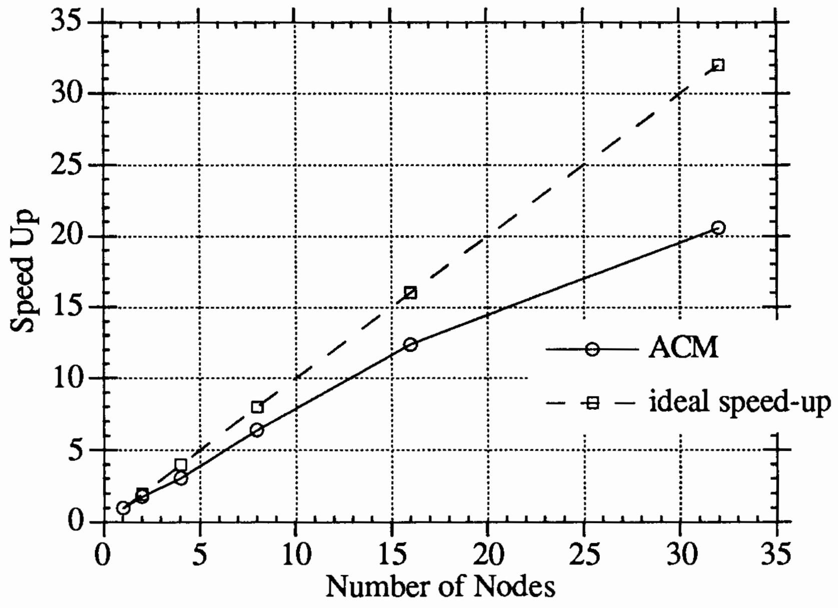


Figure 27. Parallel ACM on a 192 x 192 grid solved on the IPSC2. The problem being solved is model problem #1.

Global Levels

To allow information to travel through the domain more quickly, a global level was added. This level is constructed from all the coarsest levels of all the nodes. Each node takes a turn globally broadcasting its field information to all the other nodes. Each node receives this information and constructs a global problem. This new level is a global version of the correction that was on each node's coarsest level. Each node relaxes this global problem the same number of times. Then each node takes its portion of the correction and interpolates it to the next finest level.

The hope was that having every processor solve a global level would help negate some of the loss of parallel efficiency due to domain decomposition. The number of control volumes in this coarse level is low, of the order of the number of processors.

Implementing this global level did decrease the number of iterations required to solve the problem, however it also increased the required execution time.

This method did not work because of communication overhead. The time required to communicate a message is very long compared to the time required to execute an instruction. The problem with this algorithm is that each processor had to stop execution and take turns sending its subdomain to all the other nodes. This overhead outweighed the benefits of a global solution. Using this algorithm on larger grids did not help. The reason is that as the problem size increased, the global level occurred at a higher level. Thus, the correction made at the coarsest level did not greatly affect the refinement on the finest level.

Adjusting the transition level between the domain decomposition levels and the global level did not help either. A lower global level meant that each node had to broadcast a larger field. This increased communication cost, and again outweighed the better correction that was gained.

AREAS OF FURTHER RESEARCH

The global level approach mentioned above did decrease the number of iterations required to solve the problem. This global level injected a correction that reflected the global error. The idea was correct but the implementation was poor. What is needed is an efficient communication scheme that allows each processor to gain the global information. Another approach would be to create a global TDMA algorithm. This global algorithm would solve the global problem without each node having to have the entire field in memory. This method would certainly entail idle processors and global communication. Thus, a global TDMA relaxation would have to be judiciously used to refine the solution.

Another area for future research is to solve hyperbolic and parabolic problems. This would be done by adding the unsteady and advective terms of the advection-diffusion equation. A grid description is described by Patankar [17]. This research group plans to do this in the immediate future.

In practice, it is unreasonable to expect the user to make multiple runs to see which is the optimal number of relaxations for the particular number of nodes to be used. On the serial code it was acceptable to execute the code by performing a predefined amount of relaxing at each level. The number of relaxations that gave good performance did not change much with problem size. However, the convergence of parallel ACM is strongly affected by the number of relaxations performed. One way to solve this problem would be to use an adaptive technique. The program would monitor residual reduction on the first cycle. The program would find the number of relaxations that were required to reduce the residual at each level by a user specified amount. Subsequent cycles would use these without further residual monitoring. This methods would allow larger grids to be solved without consideration to the amount of work to be done on each level.

CHAPTER VI

SUMMARY

SERIAL RESULTS

This research analyzed the performance of ACM on elliptic problems. A model problem was devised so that cross platform performance could be evaluated. This model problem was two dimensional heat conduction with a distributed heat source. The governing partial differential equation is Poisson's equation.

Brandt [3] introduced multigrid methods. ACM is a subset of these methods. Brandt showed that an optimal implementation of multigrid would give an execution time (measured in work units) that is independent of problem size. ACM gives a slight increase in work units as the problem size increases. This is due to the suboptimal implementation of ACM. The suboptimal implementation is because there is a predetermined amount of relaxing done on each level and the interpolation scheme is not weighted.

F and V cycle schemes were investigated in this research. For the model problem, V cycles gave slightly better performance of F cycles. V cycles are recommended for nonlinear problems because F cycles drive the residual to machine error in a few iterations. For nonlinear problems, the finest field need only be relaxed a small amount between coefficient updates.

The execution time of ACM was compared to TDMA. It was found that the execution time of TDMA scaled as the number of unknowns to a power of 2.0. The ACM scaled with an exponent of 1.5. This decrease of 0.5 translate into the execution time of ACM being more than an order of magnitude lower than the TDMA for large problems.

TDMA fails to solve anisotropic problems were the material conductivities differ by orders of magnitude. Kelkar [14] found that the application of BC fixed that problem. The physical problem that Kelkar modeled was reproduced. It was found that ACM was also able to solve this problem. This is expected because BC is a multigrid method. The relative execution time between ACM and TDMA with BC was unchanged when the problem became anisotropic.

PARALLEL RESULTS

A domain decomposition strategy was chosen to solve the discrete problem on a distributed memory parallel computer. In a domain decomposition method, the discrete domain is broken up into pieces and each piece is given to a processor. The grid is broken up in such a way that a node's shared boundary control volumes are internal control volumes in its neighbor's domain. This means that it is a node's neighbor who updates the value of these control volumes. After the neighbor has updated these control volumes, their value is sent to the node. The process of doing this is called a boundary swap.

The global residual is found by having each control volume find the residual in its piece of the domain. A global summation operation is then performed to find the total global residual. Two processors will each have a version of the value of the boundary control volumes. The only time these two versions are the same is immediately after a boundary swap. Thus, the global residual calculation must be performed after a boundary swap to be correct.

TMDA and GS was adapted to hypercubes. Each processor does a certain number of TDMA or GS relaxations on the domain. A boundary swap is then performed. It was found that for the best performance, a minimal amount of relaxations should be performed between boundary swaps. For both TDMA and GS, this minimal amount was one relaxation. The combination of one TDMA or GS relaxation and a boundary swap will hence be referred to a relaxation operation.

It was found that the GS parallelizes almost perfectly. That is, GS has almost perfect speed up. This is because GS is a local method. A control volume's update only depends on neighbor control volumes. Thus, GS is unaffected by domain decomposition. Unfortunately, it is because GS is a local method, that it is slow to converge on large problems.

TDMA is a global method. TDMA updates entire columns of control volumes. On a single processor computer these columns span the entire domain and boundary information travels immediately through the entire column. When a domain decomposition strategy is adopted on a parallel computer, this boundary to boundary communication is severed. Boundary to boundary information must now travel from one processor to another during a boundary swap. Thus global methods are affected by domain decomposition.

The time to communicate messages between processors is significant compared to the time of computation. It is important to minimize the ratio of communication to computation. This ratio is minimized in two ways. First, each processor's piece of the domain should be as square as possible. Second, only large problems should be solved on these machines. The second method may seem to be a severe restriction. But the reader should be reminded that it is only large problems that require the computational power of a supercomputer.

ACM is a global method and thus does not give perfect speed up. There is a loss of parallel efficiency that is due to the domain decomposition. The speed up of ACM is comparable to TDMA but ACM gives execution times that are more that an order of magnitude lower that TDMA. Because the speed up of ACM is good, it appears that it will also run well on a larger number of processors.

In an attempt to minimize the loss of parallel efficiency due to domain decomposition, a global level was added. This global level is constructed at the coarsest level. It is a global version of the coarsest level. Each processor solved the same problem concurrently. Each processor then interpolated the correction back down to lower levels. This method achieved the desired affect of giving a number of iterations that was almost independent of the number of processors. However, the cost of communication was high because of poor implementation. This method holds promise for future research.

It was found on serial machines that continually monitoring the residual reduction is very inefficient because of the computational cost relative to relaxing. It was assumed that this would be the case for the parallel ACM. The current implementation of ACM does a predefined number of relaxations on each grid level. Doing a predetermined amount of work was not a large concern of serial machines. This was because the execution time was almost independent of problem size for a constant amount of relaxing on each level. However, the execution time of the parallel ACM is strongly affected by the number of processors for the same amount of relaxing on each level. The current implementation of ACM is weak because of this. One way to fix this problem would be to use an adaptive scheme that monitors the residual reduction on each level on the first ACM cycle. The number of relaxations required to reduce the residual by optimal amount would be found on all levels. This number of relaxations would be used with out further residual monitoring.

ACM has been found to be an efficient method for the solving of elliptic problems on distributed memory machines. Future distributed memory machine can be expected to be more computationally powerful and to have much faster communication rates. The software that allows one to take advantage of these hardware advances is lagging behind. It is hoped that this research has somewhat filled that void.

WORKS CONSULTED

1. Anderson, D.A., J.C. Tannehill and R.H. Pletcher, *Computational Fluid Mechanics and Heat Transfer*, 1984, Hemisphere, Washington.

2. Braaten, M.E., "Solution of viscous fluid flows on a distributed memory concurrent computer", *Int. J. Num. Methods Fluids*, vol. 10: 889-995, 1990.

3. Brandt, A., "Multi-level adaptive solutions to boundary-value problems", *Math. Comput.*, vol. 13(138): 333-390, 1977.

4. Briggs, B., L. Hart, S. McCormick and D. Quinlan, "Multigrid methods on a hypercube", in *Multigrid Methods: Theory, Applications and Supercomputing*, S.F. McCormick (ed.), 1988, Marcel Dekker, Inc, NY.

5. Briggs, W. and S. McCormick, "Introduction", in *Multigrid Methods*, S.F. McCormick (ed.), 1987, SIAM, Philadelphia.

6. Burden, R.L. and J.D. Faires, *Numerical Analysis*, 1985, Prindle, Weber & Schmidt, Boston.

7. Chan, T.F. and Y. Saad, "Multigrid algorithms on the Hypercube multiprocessor", *IEEE Trans. Computers*, vol. C-35(11): 969-977, 1986.

8. Chan, T.F. and R.S. Tuminaro, 1986, "Implementation of multigrid algorithms on hypercubes", NASA Ames Research Center, Report No. 86.30,

9. Intel Corporation, "iPSC/2 User's Guide", Beaverton, Oregon, 1988.

10. Fletcher, C.A.J., *Computational Fluid Dynamics, volume I*, Springer Series in Computational Physics, J. Armand, et al. (ed.) 1988, Springer-Verlag, Berlin.

11. Hart, L. and S. McCormick, "Asynchronous multilevel adaptive methods for solving partial differential equations on multiprocessors: Basic ideas", *Parallel Computing*, vol. 12: 131-144, 1989.

12. Hutchinson, B.R. and G.D. Raithby, "A multigrid method based on the additive correction strategy", *Numer. Heat Transfer*, vol. 9: 511-537, 1986.

13. Jespersen, D.C., "Multigrid methods for partial differential equations", in *Studies in Numerical Analysis, vol. 24*, 24, G.H. Golub (ed.), 1984, Mathematical Association of America, Inc.

14. Kelkar, K., “An efficient Iterative Solution Method for the Numerical Prediction of Heat Transfer Problems Involving Large Differences in Thermal Conductivites”, AIAA Paper No. 89-1684, 1986.

15. McCormick, S.F., *Multilevel Adaptive Methods for Partial Differential Equations*, Frontiers in Applied Mathematics, vol. 6, 1989, SIAM, Philadelphia.

16. Miller, T.F. and F.W. Schmidt, “Evaluation of a multilevel technique applied to the Poisson and Navier-Stokes Equations”, *Numer. Heat Transfer*, vol. 13: 1-26, 1988.

17. Patankar, S.V., *Numerical Heat Transfer and Fluid Flow*, 1980, Hemisphere, Washington.

18. Press, W.H., B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes in C*, 1989, Cambridge University Press, Cambridge.

19. Ragsdale, S., *Parallel Programming*, 1991, McGraw-Hill, New York.

20. Recktenwald, G., April 27, 1992, 1992, *personal communication*.

21. Settari, A. and K. Aziz, “A generalization of the Additive Correction Methods for the Iterative Solution of Matrix Equations”, *SIAM J. Numer. Anal.*, vol. 10: 506-521, 1973.

22. Vanka, S.P., “Block-implicit multigrid solution of Navier-Stokes Equations in primitive variables”, *J. Comp. Phys.*, vol. 65(1): 138-158, 1986.

23. White, F., *Viscous Fluid Flow*, 1974, Mc-Graw-Hill, New York.