Solving the Depth Interpolation Problem on a Parallel Architecture with Efficient Numerical Methods

CUCS-358-88

Dong Jae Choi

Department of Computer Science
Columbia University

ABSTRACT

Solving the Depth Interpolation Problem on a Parallel Architecture with Efficient Numerical Methods

Dong Jae Choi

Many constraint propagation problems in early vision, including depth interpolation, can be cast as solving a large system of linear equations where the resulting matrix is symmetric and positive definite (SPD). Usually, the resulting SPD matrix is sparse. We solve the depth interpolation problem on a parallel architecture, a fine grained SIMD machine with local and global communication networks. We show how the Chebyshev acceleration and the conjugate gradient methods can be run on this parallel architecture for sparse SPD matrices. Using an abstract SIMD model, for several synthetic and real images we show that the adaptive Chebyshev acceleration method executes faster than the conjugate gradient method, when given near optimal initial estimates of the smallest and largest eigenvalues of the iteration matrix.

We extend these iterative methods through a multigrid approach, with a fixed multilevel coordination strategy. We show again that the adaptive Chebyshev acceleration method executes faster than the conjugate gradient method, when accelerated further with the multigrid approach. Furthermore, we show that the optimal Chebyshev acceleration method performs best since this method requires local computations only, whereas the adaptive Chebyshev acceleration and the conjugate gradient methods require both local and global computations.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Three Disciplines

This thesis is interdisciplinary research across three areas: computer vision, numerical analysis, and parallel computer architectures. As such, we believe it is representative of a necessary approach to a broad range of sensor-related computational problems: they must be solved accurately, and they must be solved efficiently; but standard algorithms on standard conventional machines used to fail to perform satisfactorily.

Many middle-level constraint propagation computer vision problems, including depth interpolation, can be cast as solving a large system of linear equations with a symmetric positive definite (SPD) matrix. Usually, the resulting SPD matrix is sparse. We are interested in solving this problem in a computationally efficient way.

Many current iterative methods employ only local information in the matrix. Even when global information is used, it is done in an indirect fashion. We study the more efficient iterative methods well known in numerical analysis. In particular, we investigate those methods where global information in the matrix is obtained in adaptive fashion.

Secondly, we explore whether these efficient iterative methods can be run on a family of emerging parallel architectures, thus fully exploiting the computational power provided by these machines. We first define the architectural need imposed by our application and then show the implementation of the algorithms on a parallel architecture step by step. As a result of analysis, we predict how fast the algorithms will run and how much storage space they will require. But the other side of the coin is that, even with the most powerful hardware available now, we demonstrate how slowly this one particular problem in computer vision will run, (even not mentioning the whole vision task which is so easily and unconsciously performed by the human visual system.)

1.2 Perspective on Parallel Computation

In general, in the development of parallel machines, three perspectives on the machine should progress together. First, there is the research work on theoretical models of parallel computation. Second, there is the actual implementation of parallel hardware, based on existing technology and the accompanying development of software environments, high and low level languages, new programming techniques, and various software tools. Lastly, there is the development of the algorithms that can best utilize the new parallel machines. In fact, the urgent need of efficient execution of application programs often initiates the whole process of research and development.

Our research can be scrutinized under this theoretical framework. We have developed a model of computation for the single instruction multiple data (SIMD) class of machines imposed by our application need. Secondly, we have the algorithms; in our case, the efficient numerical methods, which we analyze for space and time complexity of parallel computation. Finally, we discuss the resultant advantage on actual execution of application, presenting numerical results and predictions of enhanced performances.

Our work contributes to a particular family of parallel machines, (SIMD architectures with local and global communication networks;) to the area of image processing, (the depth interpolation problem;) and more generally, to smoothness constraint propagation problems in early vision and other related sensing.

1.3 Acceleration of Execution Speed

We also present a survey of the iteration methods with particular stress on the execution speed. We start with wide classifications and narrow the scope until we reach the particular result we have achieved. This survey also pulls together numerical analysis and parallel computation.

First, we address the issue of execution on digital computers against analog processing on an analog mechanism. Digital processing has several advantages: flexibility, precision, and stability in terms of both maintenance and repeatability. But its problem is speed. We deal mainly with digital computation in our work, but we return to analog computation in a section of the last chapter where we will discuss the possible image processing hardware in future.

Second, we address the choice of using a serial method that can be run only on the sequential machines against a parallel method that can be run on a parallel architecture as well. The Gauss-Seidel method is a typical example of much used serial methods, which converges when the matrix is SPD. There are variations of it, such as the successive overrelaxation or the symmetric successive overrelaxation methods, which speed up the iteration process further on a serial machine. The Jacobi method is a first choice for the parallel method, but it is slower than the Gauss-Seidel method when executed on the sequential machines and it converges only under restricted conditions, (for example, when the matrix is irreducible with weak diagonal dominance, which happens to be not satisfiable in the depth interpolation problem.) However, there are still other more powerful methods, such as the Chebyshev and the conjugate gradient, which can be run also in a parallel fashion. These methods are indeed very powerful, and they are provably optimal, but they require global information. Naturally, our choice is on parallel methods. Specifically, we have worked on the adaptive Chebyshev acceleration, which is a variation of the Chebyshev, and the conjugate gradient methods. Even though the adaptive Chebyshev acceleration method uses the slow Jacobi as the underlying basic iterative method, it is faster than the conjugate gradient method when started with more accurate initial estimate of the largest and smallest eigenvalues of the iteration matrix, that is, with more accurate global information. Furthermore, when good estimates of the largest and smallest eigenvalues are available, we can use the optimal Chebyshev acceleration method, which takes less execution time since global computations needed to obtain better estimates of the eigenvalues are eliminated.

Third, the iteration methods can be accelerated further using a multigrid approach, where several coarse and fine grids of various resolutions are employed. The execution at coarse levels is faster but of limited precision, while at fine levels we have ordinarily slower response but finer details are obtained. Through executing many iterations on the coarser levels as far as possible, at least in the initial stages, the iteration process is speedied up. Others have tried the multigrid approach with the Gauss-Seidel method as the underlying iterative relaxation method; in our approach, we investigate the multigrid approach with two methods mentioned above, which are more naturally parallel, but use global information in the matrix. Here, we observe similar accelerations through the adoption of multigrid approach. Often, the degree of improvement is less dramatic since our methods are already powerful enough for the execution on the finest level only.

Lastly, we mention the parallel implementation issue again. The execution of the powerful numerical methods on a chosen parallel architecture, the SIMD machines with local and global communication networks, will be analyzed in detail both in terms of space and time complexity. The Connection Machine with a floating point accelerator unit, possibly of double precision arithmetic, is a good choice to run these algorithms efficiently. We shall provide the necessary technical guidance and relevant comparisons on the matter of machine selection, but our main concentration will be in a more idealized setting. That is, a tree machine with multiple mesh connections of different resolutions is nearly equivalent in power to a pyramid machine, and naturally it can execute this kind of matrix iteration problems in the fastest way imaginable. Such an idealized architecture will be used as the main vehicle and one particular abstract SIMD model based on this topology will be used extensively throughout our work.

1.4 Overview of Thesis

Chapter 2 examines the depth interpolation problem in detail. The formulation of the problem leads to solving a large system of linear equations with a SPD matrix. The previous work on this problem and other related work are reviewed. We will show how, as an evolutionary process, our approach can solve the same problem in a computationally efficient way.

Chapter 3 discusses three machines from the SIMD architecture family. An abstract model of SIMD computation whose features are drawn from the three SIMD machines is derived. The architectural features needed to solve a large system of linear equations with the sparse SPD matrices are elaborated further. The generalized abstract model which provides the power of the pyramid machine is then studied.

Chapter 4 introduces two computationally efficient methods that are well known in numerical analysis. The parallelization of the computations on a SIMD architecture is given and the computational and the communication cost of the computations are analyzed based on the SIMD model derived in Chapter 3. To this moment, the main focus is on the efficient execution at the finest level. Then, the multigrid method is reviewed to show eventually how two efficient methods can be run faster by employing several levels of the resolution.

Chapter 5 looks at numerical results and compares them.

Chapter 6 reviews the contributions made and suggests possible future directions.

The Appendix contains program listings and supplementary numerical results.

2. The Depth Interpolation Problem

In this chapter, we start the discussion of the depth interpolation problem in the context of early vision. We then describe the work done by Grimson [Grim 81] and Terzopoulos [Terz 84].[1] After discussion of further related work, we present the significance of our research.

2.1 Nature and Significance of the Problem

Visual processing has been formulated as a series of computational stages. According to Marr [Marr 82], perhaps the researcher most responsible for the popularity of this framework, each stage consists of a logically cohesive computation that takes as input some visual representation and produces as output a new representation, a more complete and useful description of the visual world.

Early vision is the set of processes that recover the physical properties of 3D surfaces from 2D images. The examples of early vision processes are edge detection, binocular stereo, structure from motion, shape from texture, shape from shading, optical flow, surface reconstruction, and so on.

Marr's framework of early vision is characterized by at least three major processing stages. The first stage transforms the intensity representations of the retina into a primary representation, called the *primal sketch*. Changes in the physical properties of surfaces almost always give rise to intensity changes in the images, and it is at the level of the primal sketch that the locations of these changes are made explicit. In the second processing stage, special processes, such as those concerned with stereo and analysis of motion, infer information about the shapes of surfaces from the contents of the primal sketch. Since inferences can typically be made only at those locations which have been marked in the primal sketch, so the information generated is sparse, and is collected into sparse representations of surface shape that are referred to as the *raw* $2\frac{1}{2}-D$ *sketch*. The final stage is one of full surface reconstruction in which the sparse representations are transformed into a *full* $2\frac{1}{2}-D$ *sketch* containing explicit information about surface shape at all

[1]Boult's dissertation [Boul 86] provides another good review on the research done by Grimson and Terzopoulos. For his critical analysis, see his chapter 7.

points in the scene, consistent with our perception.

Human perception is a vivid one of dense and coherent surfaces in depth. This suggests that there exists a middle level visual process that transforms the scattered information into a dense surface representation. The low level visual processes provide several visual cues to reconstruct the visible surfaces. One low level visual process, stereo, generates depth only at scattered edge points; another process, shape from texture, generates orientation at texel points that may be scattered as well. Given these sparse constraints, a depth interpolation process would compute the depth of the visible surfaces at every point explicitly.

2.2 Previous Work

2.2.1 Work done by Grimson

Grimson formulated one approach to the depth interpolation problem and studied it in the restricted context of depth constraints from stereopsis [Grim 81].

There is evidence that the human visual system detects intensity changes over a range of resolutions through the use of up to five independent, spatial-frequency-tuned, bandpass channels. Grimson proposed that before reconstruction begins, the multiple, sparse depth representations that are output through the different bandpass channels can be combined into a single raw $2\frac{1}{2}-D$ sketch in a way which maintains consistency across all scales. The raw $2\frac{1}{2}-D$ sketch then contains sparse depth information at the finest resolution possible. Next, a single reconstruction process operating at this finest level generates a unique full $2\frac{1}{2}-D$ sketch representing depth information at high resolution.

The input to the interpolation process consisted of the zero-crossings[2] of the convolved image, with depth information computed along zero-crossing contours. These contours tend to be scattered at random rather than distributed uniformly. In general, any one of a multitude of widely varying surfaces could fit the *explicit* conditions, the depth or surface orientation values

[2]Zero-crossings mark the sign changes of a bandpass filtered image.

imposed along the zero-crossing contours. The *implicit* conditions were that the surface should not impose any zero-crossing contours other than those which appeared in the convolved image.

Grimson suggested an "interpolation" method. He suggested that given a set of scattered depth constraints corresponding to points along the zero-crossing contours of the primal sketch, the surface which *best* fits the known constraints is that which passes through the known points exactly and minimizes the expression,[3]

$$\iint (v_{xx})^2 + 2(v_{xy})^2 + (v_{yy})^2 \, dx\,dy,$$

referred to as the *quadratic variation* of the surface v.[4] The interpolation approach is preferred. provided the known observations are exact. But since we have noise in practice, different approaches such as smoothing seem to be better; Terzopoulos uses smoothing only. Nevertheless, if we use very stiff "spring constants" to constrain the surface with the known values, then we return to the interpolation approach again.

In the implementation, Grimson was concerned with *biological feasibility*, which was important if one was to describe a model of the human visual system. A set of algorithmic criteria was stated as follows [Grim 81, p. 163].

- Parallelism : The need to process large amounts of input data in short amounts of time implies the use of computations that can be implemented in a parallel manner, using a large number of interconnected processors.
- Local support : If the number of processors involved in the computation is large, it becomes infeasible to connect each one to all of the others. Rather, there should only be local connections between the processors. Here, *local* means not only that the number of connections be small, but also that since the information being processed has a two-dimensional plane as an underlying coordinate system, the connections should also be local in a spatial sense.
- Uniformity : If it is possible, though not as critical as the first two, an algorithm that utilizes parallel networks of identical processors will be favored over other algorithms.

[3]The subscripts indicate partial differentiation. For example, $v_x = \frac{\partial v}{\partial x}$ and $v_y = \frac{\partial v}{\partial y}$.

[4]Actually, Grimson investigated both interpolation and approximation methods [Grim 81, p. 177]. He suggested that the conjugate gradient algorithm is appropriate for the case of approximating the surface, by requiring that the surface minimize an objective function (the quadratic variation) and pass near, but not necessarily through, the known points. He suggested also that the gradient projection algorithm is appropriate for the case of interpolating the surface, by requiring that the surface minimize an objective function (the quadratic variation) and pass exactly through the known points.

He used the gradient projection method to find an interpolated surface but slow convergence rates were observed in his work. For illustrated examples, it took 500-1000 iterations to reach the stopping condition. But it was indicated that a multigrid implementation might require only 25-50 iterations [Grim 83, p. 67].

2.2.2 Work done by Terzopoulos

Terzopoulos worked further on depth interpolation problem [Terz 84]. He proposed a computational theory of visible-surface representations and developed a visible-surface reconstruction process for generating them. Instead of Grimson's ''interpolation'' approach, he proposed an ''approximation'' method where the discrete potential energy functional associated with the surface is minimized. In his formulation, known depth constraints, or orientation constraints, or both, contribute as spring potential energy terms.

2.2.2.1 Visible Surface Representation

The physical model for surface reconstruction by Terzopoulos can be described as follows [Terz 84, p. 35].

- The implicit surface smoothness constraint is modeled by a thin, flexible plate.
- Ideal springs constrain the thin plate with explicit constraints; i.e., either depth measurements, or orientation measurements, or both classes of constraints.
- To handle discontinuities, the smoothness imposed by the thin plate is relaxed locally -
    - free boundaries are introduced along surface depth discontinuities and
    - patches of thin plates are joined by membrane strips along surface orientation discontinuities.

2.2.2.2 The Continuous Form of the Problem

Mathematical problems for which the existence, uniqueness, or stability of solutions cannot be guaranteed a priori are said to be *ill-posed.* Ill-posed problems cannot be solved in general, without imposing some additional restrictions on possible solutions. Through a number of systematic approaches, notably the regularization methods [Tikh 77], ill-posed problems can be solved by reformulating them as variational principles that are effectively computable. Unlike the original problems, the variational principle formulations are *well-posed* in the sense that a solution exists, is unique, and depends continuously on the data.

Terzopoulos' analysis [Terz 84, p. 73] reveals that the minimal sets of conditions under which the visible surface reconstruction problem is well-posed for a single surface patch are

- three noncolinear depth constraints, or
- two depth constraints as well as a single p or a single q orientation constraint, or
- a single depth constraint as well as a single p and a single q constraint, or
- a single p and a single q constraint with the center of gravity of the surface fixed,

where p and q are x and y components of the surface normal.

Since the visual processing of natural scenes readily yields many such constraints, the problem can be considered as being well-posed in general.

Many visible surface reconstruction algorithms rely on the minimization of discrete functionals associated with the surface. The ones Grimson and Terzopoulos used are given by the summation of the discrete potential energy functional for thin plate, the discrete functional for depth constraints, and the discrete functional for orientation constraints.

The continuous form of the surface reconstruction problem is transformed to yield the simplified approximate energy functional $\xi^h(v^h)$, where the region (the projection of a surface patch to two-dimensions on the retina) has been divided into identical square elements E with sides of length h through a uniform tessellation τ^h.

$$\xi^h(v^h) = \xi^h_p(v^h) + \xi^h_D(v^h) + \xi^h_O(v^h),$$

where

$$\xi^h_p(v^h) = \frac{1}{2}\sum_{E\in\tau^h}\int\int_E (v^h_{xx})^2 + 2(v^h_{xy})^2 + (v^h_{yy})^2 \, dx\,dy,$$

$$\xi^h_D(v^h) = \frac{1}{2}\sum_{(x_i,y_j)\in D} \beta_{(x_i,y_j)} [v^h(x_i,y_j) - d(x_i,y_j)]^2,$$

$$\xi^h_O(v^h) = \frac{1}{2}\sum_{(x_i,y_j)\in P} \alpha_{p_{(x_i,y_j)}} [v^h_x(x_i,y_j) - p(x_i,y_j)]^2$$
$$+ \frac{1}{2}\sum_{(x_i,y_j)\in Q} \alpha_{q_{(x_i,y_j)}} [v^h_y(x_i,y_j) - q(x_i,y_j)]^2,$$

where D denotes the set of points in the region at which the scattered depth information is present, and P and Q are the set of points at which p and q measurements are available. The values of $\alpha_{(x_i,y_j)}$ and $\beta_{(x_i,y_j)}$ are given by the spring constants.

Terzopoulos applied the *finite element method* for the discretization instead of the *finite difference method*, an older technique. He used a nonconforming finite element [Terz 84, p. 77].

2.2.2.3 Discretization of the Problem

Since the function is convex, to obtain the minimum, we set to zero its partial derivatives with respect to each of the displacements $u^h_{i,j}$ for node (i, j). The minimizing vector of displacements, u^h, is equal to the unique solution of a large system of linear equations:

$$A^h u^h = b^h.$$

The nonzero coefficients of each equation is specified as summations of *computational molecules* that denote multiplications of nodal variables by scalars. In the presence of constraints and discontinuities, a set of computational molecules computes the nonzero coefficients of the linear system by local computations involving simple multiplications and additions of nodal variables in a specified spatial arrangement. Terzopoulos uses four set of computational molecules, *plate molecules*, *depth constraint molecules*, *orientation constraint molecules*, and *membrane molecules*.

The plate molecules are shown in Figure 2-1. The circles, or computational atoms, denote (h^2 times) the nonzero coefficients of the nodal variables. Node (i, j) is indicated by a double circle in each molecule.

When a node is sufficiently distant from either constraints or discontinuities, only the plate molecules contribute. This gives the following nodal equation which relates $u^h_{i,j}$, for an interior node (i, j), to the nodal variables of the other nodes:

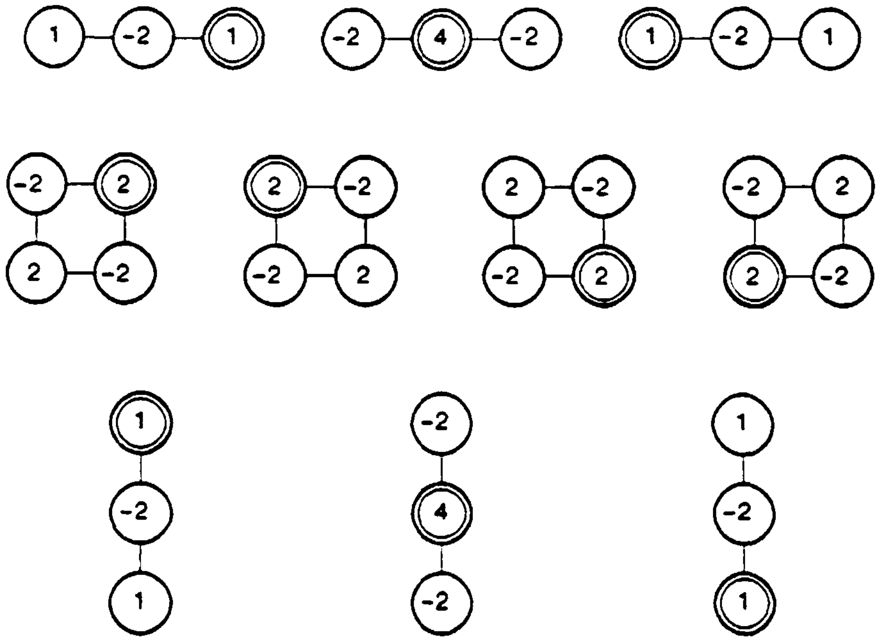

Figure 2-1: The Plate Molecules for Node (i, j)[5]

[5]taken from [Terz 84, p. 90]

$$\begin{aligned}
&(20 / h^2)\, u^h_{i,j} \\
&\quad - (8 / h^2)\,(u^h_{i-1,j} + u^h_{i+1,j} + u^h_{i,j-1} + u^h_{i,j+1}) \\
&\quad + (2 / h^2)\,(u^h_{i-1,j-1} + u^h_{i+1,j-1} + u^h_{i-1,j+1} + u^h_{i+1,j+1}) \\
&\quad + (1 / h^2)\,(u^h_{i-2,j} + u^h_{i+2,j} + u^h_{i,j-2} + u^h_{i,j+2}) \\
&= 0.
\end{aligned} \tag{1}$$

The nodal equation can be represented by a nodal molecule as illustrated in Figure 2-4. Note that the interior node molecule is obtained by summing the plate molecules in Figure 2-1.

The effects of depth constraints are represented by the depth constraint molecule shown in Figure 2-2. If node (i, j) is constrained with the depth constraint value $d^h_{i,j}$, then the nodal equation for this node is obtained by summing the depth constraint molecule with the plate molecules. Now, the nodal equation for depth constrained interior node is given by

$$\begin{aligned}
&(20 / h^2 + \beta^h_{i,j})\, u^h_{i,j} \\
&\quad - (8 / h^2)\,(u^h_{i-1,j} + u^h_{i+1,j} + u^h_{i,j-1} + u^h_{i,j+1}) \\
&\quad + (2 / h^2)\,(u^h_{i-1,j-1} + u^h_{i+1,j-1} + u^h_{i-1,j+1} + u^h_{i+1,j+1}) \\
&\quad + (1 / h^2)\,(u^h_{i-2,j} + u^h_{i+2,j} + u^h_{i,j-2} + u^h_{i,j+2}) \\
&= \beta^h_{i,j}\, d^h_{i,j}.
\end{aligned} \tag{2}$$

The value for $\beta^h_{i,j}$ is dependent on h but independent of (i, j) and is given by $\beta^h = \gamma_d / h^2$, where γ_d is constant. The value employed by Terzopoulos for γ_d is either .5 or 2.0 in most cases. He used also $\beta^h = \gamma_d / h$, where γ_d varies from .1 to 2.0. [See chapter 5 for a further discussion of β^h.]

The effects of orientation constraints are represented by the orientation constraint molecules shown in Figure 2-3. For instance, if node $(i\text{-}1, j)$ is p constrained then the nodal equation for node (i, j) is obtained by summing the upper left orientation constraint molecule in Figure 2-3 with the plate molecules. Now, the nodal equation for interior node (i, j) is given by

$$\begin{aligned}
&(20 / h^2 + \alpha^h_{i-1\,j} / (4 h^2))\, u^h_{i\,j} \\
&\quad - (8 / h^2)\,(u^h_{i-1\,j} + u^h_{i+1\,j} + u^h_{i\,j-1} + u^h_{i\,j+1}) \\
&\quad + (2 / h^2)\,(u^h_{i-1\,j-1} + u^h_{i+1\,j-1} + u^h_{i-1\,j+1} + u^h_{i+1\,j+1}) \\
&\quad + (1 / h^2 - \alpha^h_{i-1\,j} / (4 h^2))\, u^h_{i-2\,j} + (1 / h^2)\,(u^h_{i+2\,j} + u^h_{i\,j-2} + u^h_{i\,j+2}) \\
&= (\alpha^h_{i-1\,j} / (2 h))\, p^h_{i-1\,j}.
\end{aligned} \tag{3}$$

The value for $\alpha^h_{i\,j}$ is dependent on h but independent of (i, j) and is given by $\alpha^h = \gamma_p / h$, where γ_p is constant. The usual value employed by Terzopoulos for γ_p is 4.0. [Again see chapter 5.]

Note that at nodes where the effects of constraints exist, both the right and the left hand side of the nodal equations are modified as in equation (2) and (3). We will return to this later in section 4.1.3 when we discuss estimation of eigenvalues of the iteration matrix.

Smoothness constraints are inapplicable at a depth discontinuity. The depth discontinuities are treated as free boundaries. Consider a boundary node that is sufficiently near a depth discontinuity. Whenever a depth discontinuity node coincides with any constituent atoms of a plate or constraint molecule associated with the boundary node, that molecule is prohibited from taking part in the summation giving rise to the nodal equation. Suppose that the shape of the region is a square and consider a boundary node at the corner of the square. The nodal equation for this boundary node (ignoring constraints) is given by

$$\begin{aligned}
&(4 / h^2)\, u^h_{i\,j} \\
&\quad - (4 / h^2)\,(u^h_{i+1\,j} + u^h_{i\,j+1}) \\
&\quad + (2 / h^2)\,(u^h_{i+1\,j+1}) \\
&\quad + (1 / h^2)\,(u^h_{i+2\,j} + u^h_{i\,j+2}) \\
&= 0.
\end{aligned} \tag{4}$$

Figure 2-5 illustrates this boundary node (marked as a double circle) which is near depth discontinuity nodes (marked by X's). In this example, only three computational molecules from the set in Figure 2-1 participate in the construction of the nodal equation.

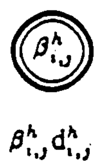

Figure 2-2: The Depth Constraint Molecule[6]

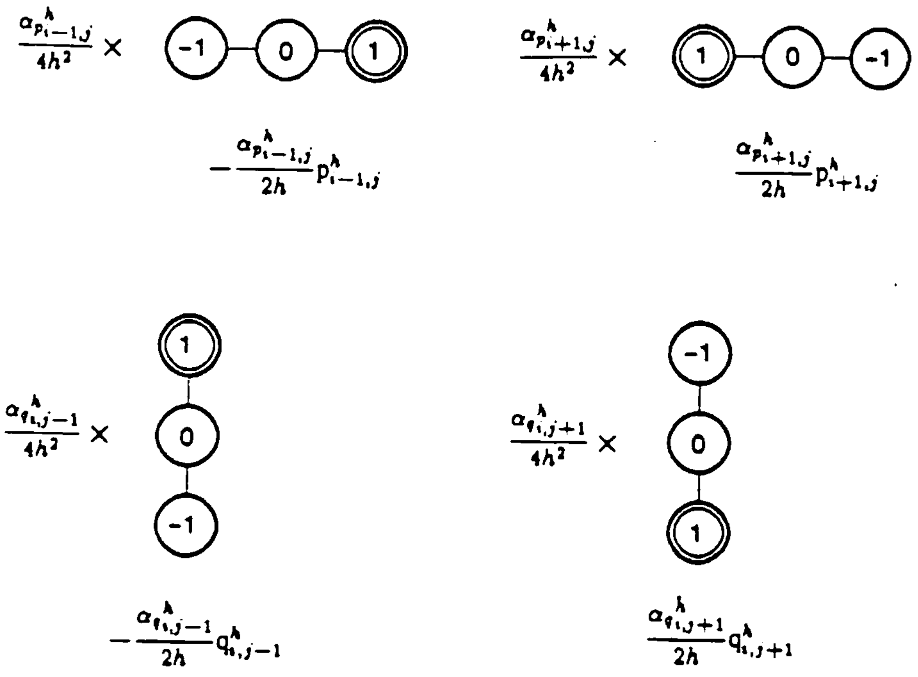
$\frac{\alpha_{p^h_{i-1,j}}}{4h^2} \times$
-1
0
1
$-\frac{\alpha_{p^h_{i-1,j}}}{2h} p^h_{i-1,j}$
$\frac{\alpha_{p^h_{i+1,j}}}{4h^2} \times$
1
0
-1
$\frac{\alpha_{p^h_{i+1,j}}}{2h} p^h_{i+1,j}$
$\frac{\alpha_{q^h_{i,j-1}}}{4h^2} \times$
1
0
-1
$-\frac{\alpha_{q^h_{i,j-1}}}{2h} q^h_{i,j-1}$
$\frac{\alpha_{q^h_{i,j+1}}}{4h^2} \times$
-1
0
1
$\frac{\alpha_{q^h_{i,j+1}}}{2h} q^h_{i,j+1}$

Figure 2-3: Orientation Constraint Molecules[7]

The upper left molecule is used only if $(i-1, j) \in P$, the upper right only if $(i+1, j) \in P$, the lower left only if $(i, j-1) \in Q$, and the lower right only if $(i, j+1) \in Q$.

[6]taken from [Terz 84, p. 92]

[7]taken from [Terz 84, p. 93]

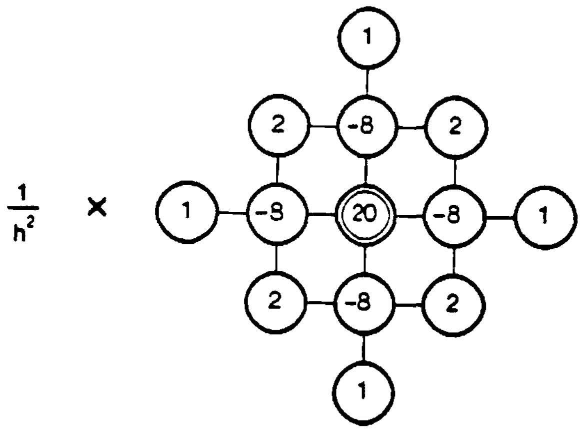

Figure 2-4: The Interior Node Molecule[8]

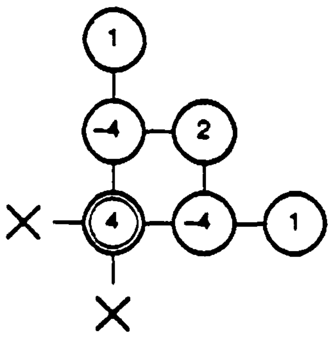

Figure 2-5: An Example of a Boundary Node Molecule[9]

---

[8]taken from [Terz 84, p. 91]

[9]taken from [Terz 84, p. 99]

2.2.2.4 Properties of the System Matrix

Definition 1: A real matrix A is *symmetric and positive definite* (SPD) if A is symmetric and if $(v, Av) > 0$ for every nonzero vector v.[10]

Because of the symmetric nature of the computational molecules, it can be easily shown that the resulting matrix is symmetric for any regional shapes. Furthermore, Terzopoulos shows the stronger result that the matrix generated is SPD [Terz 84, p. 100] :

> The finite element method has bestowed computationally desirable properties upon the system matrix, including sparseness, bandedness, symmetry, and positive definiteness.

In general, the depth interpolation problem is easier to solve when we have denser constraints. This statement can be given an informal, though quantitative, interpretation. Consider the restricted case of the depth constraints only. We can derive then the relation between the density of the depth constraints and the positive definiteness of the system matrix as follows.

The matrix A^h can be broken down as the sum of two matrices: $A^h = A^h_\phi + B^h$. The coefficients of the matrix A^h_ϕ are contributed by the plate molecules, while those of the matrix B^h are contributed by the depth constraint molecule. Suppose that we have $u^h_{i,j} = c$ for every node (i, j) where c is an arbitrary constant, i.e., the considered image is a plane of constant depth. We have

$$(u^h, A^h u^h) = (u^h, A^h_\phi u^h) + (u^h, B^h u^h).$$

For this special image, $(u^h, A^h_\phi u^h) = 0$ since c^2 can be factored out. [As an exercise, substitute $u^h = (c\ c\ \ldots\ c)^{\mathrm{T}}$ into the left hand side of the equation (1) or (4).] Since B^h is a diagonal matrix where the coefficient is given by

$$b_{l,m} = \begin{cases} \beta^h & \text{if } l = m \text{ and this node is depth constrained} \\ 0 & \text{otherwise}, \end{cases}$$

where $B^h = (b_{l,m})$ for $1 \leq l, m \leq n$. Here, n is the number of nodes in a depth continuous region. By factoring out c^2, it can be easily shown that $(u^h, B^h u^h)$ is proportional to the number of depth constrained nodes in the region. Thus, the matrix A^h becomes increasingly ill-conditioned as the density of the depth constraints gets sparser.

The matrix A^h is also sparse. Even for interior nodes which are sufficiently distant from a

---

[10]Given two vectors v and w of R^n, the inner product (v, w) of the vector v with w is defined by $(v, w) \equiv v^{\mathrm{T}} w$ where v^{T} denotes the transpose of the vector v.

boundary, they interact with only 12 neighbors, all of them at most only 2 nodes away, as illustrated in Figure 2-4. For nodes near discontinuities, even fewer neighboring nodes are involved.

2.2.2.5 The Multigrid Relaxation Method

Terzopoulos used a multigrid method [Bran 77] with the Gauss-Seidel relaxation method at each relaxation sweep to speed up the convergence rate. These multiresolution iterative algorithms are suited for implementation on massively parallel networks of simple, locally-connected processors; we shall discuss the multigrid method in full detail in section 4.3.

2.2.3 Related Work

It is shown that two iterative methods, the Chebyshev and the conjugate gradient methods, are provably optimal in terms of computational complexity [Trau 84].

Lee investigated the Chebyshev method on several low level vision problems [Lee 85]. For shape from shading [Ikeu 81] and optical flow [Horn 81] problems, previous researchers used the Gauss-Seidel method, which is slow.

Lee observed that the original matrices were not SPD and converted them to SPD. Furthermore, he derived the lower and upper bounds of the smallest and largest eigenvalues of the matrix, two quantities that are essential for the Chebyshev method. Due to the simplicity of the matrix involved, he could estimate these extreme eigenvalues, prior to major computation, i.e., before matrix iterations are started.

2.2.4 Significance of Current Work

We have discussed the depth interpolation problem at length, in technical detail. Here, we present the significance of our work, comparing it with existing works. We start the comparison with the most recent work and discuss technicalities again, but also discuss the bigger issues associated with our work.

One difficulty associated with the Chebyshev method used by Lee is that it needs good estimates of the largest and smallest eigenvalues of the underlying matrix. Unless this matrix is sufficiently structured, it may be analytically impossible to get good estimates [Golu 85]. In this thesis we shall study the adaptive Chebyshev acceleration method where the estimates of the extreme eigenvalues are adaptively improved. We shall study also the conjugate gradient method[11] where the iterative process is not started with any initial global information.

These two iterative methods are numerically efficient but they require global information during the computational process. How can we get global information and also efficiently compute it? In other words, what kind of computational structure can support the computational requirement of these highly efficient numerical methods? The problem here is a straightforward one-to-one mapping of algorithms to computer architecture. Another related question is, if some computer architecture provides the convenient facilities to compute global information, then why we do not take full advantage of these capabilities? With the support of hardware, the implementation of these algorithms becomes quite easy.[12]

In our study, we follow the Terzopoulos' formulation on visible surface reconstruction and use the matrix derived by him. However, we present an alternative depth interpolation process using theoretically better iterative methods, which speed convergence and are amenable to certain classes of parallel computers.

This aspect of the problem reminds us of Marr's discussion about three levels at which an information processing device is understood [Marr 82, p. 24]. At the top level, we have the abstract computational theory of the device. At the middle level, the focus is on the representation and algorithm, used to implement the computational theory. At the bottom level, we are concerned with the details of how the algorithm and representation are realized physically - the neural networks in case of the human visual system or the computer architecture for the machine vision. From this point of view, the theoretical developments by Grimson and Terzopoulos were much influenced by the human visual system, partly because of the wide

[11]Note that the conjugate gradient method was already studied by Grimson for the approximation approach and mentioned by Terzopoulos as well.

[12]The adoption of our approach through the bottom-up thinking process might be a sort of reformulation of the problem. In [Mins 86, p. 144], we have : "We often self-impose assumptions that make our problem more difficult, and we can escape from this only by reformulating those problems in ways that give us more room."

discrepancies between the high performance of the human visual system and the low performance of the machine vision system, and partly by the top-down way of their thinking. This may have led them to insist too much closeness to human visual system even at the implementation level.

Our study was deeply influenced by the hardware implementation issue. In particular, a particular parallel architecture, a fine grained SIMD machine with local and global communication networks, turned out to meet our need. First, this family of machines is increasingly available. Second, all necessary computational requirement, including the computation of global information, can be met.[13]

In this work, we study two efficient numerical methods and their implementation on the parallel architecture chosen. With this partial result, we examine again the issue of computational efficiency raised by Terzopoulos. He used the Gauss-Seidel method in his implementation of multigrid method. Here, we apply two highly efficient and other methods to multigrid approach. Again, we analyze the additional computational requirement and the implementation detail on this parallel architecture.

In one sense, our study deals mainly with the bottom level hardware implementation issue and can be designated as further refinement on works done by Grimson and Terzopoulos. But the other aspect of our study is a bit more general. Traditionally, a lot of work in computer vision using the parallel machines, in particular, the SIMD class of machines, have been in the area of low-level vision. Our work extends the research into middle-level vision. As a concrete example, we have solved the depth interpolation problem. But our solution, the efficient execution of the sparse SPD matrix iterations on a parallel architecture, can be applied to other middle-level machine vision tasks as well as to other general areas such as map making.

In chapter 4, we shall review rigorously the mathematical theory behind the iterative methods and discuss the implementation of them on a parallel architecture. Before doing that, we shall discuss this particular parallel architecture in next chapter.

[13]We shall describe shortly the requirement on architecture more formally in next chapter, at section 3.2.1.

3. Architectural Background

3.1 SIMD Machines

Three single-instruction multiple-data stream (SIMD) machines are reviewed to illustrate the architectural background concepts that motivated the design of these machines for the eventual use in general AI and in particular in computer vision.

In computer vision problems, we have tremendous amounts of data, either raw data itself or the properties derived thereof, such as the depth. The conventional von Neumann machines suffer from two aspects. They have the famous ''von Neumann bottleneck'' between processor and memory [Back 78]. Also, serial processing offered by conventional machines or even moderately accelerated processing, such as from pipeline architectures, provide only limited processing power.[14] For a large number of low- and middle- level computer vision problems, we need identical processing repeatedly on whole or part of data. Therefore, concurrent, fine grained SIMD processing is a natural one for this kind of task.

Data are supplied usually in two-dimensional fashion, for instance, 1024×1024 images. The mesh communication is a local communication scheme that matches the natural structure of such data. In all three systems we review, we assume therefore SIMD processing and mesh communications.

In addition to these common characteristics, interesting features from each system are emphasized. First, we review the MPP, particularly stressing the arithmetic computational processing capability. Secondly, we discuss NON-VON, emphasizing its global communication capability, which is implemented with the tree topology. Thirdly, we discuss the Connection Machine, noting its more general (albeit more costly to implement) global communication

[14]The PIPE is one of the most powerful image processing engines ever developed [Aspe 87]. It is a special-purpose machine for real time low-level vision consisting of eight processing stages connected in a pipelined fashion. As an example of algorithms developed on the PIPE, see the convolution algorithms reported in [Stew 86].

The Warp Systolic Array is another high-performance highly parallel pipeline processor. In current implementation, 10 processors are connected in a linear topology, but they are also interconnected with crossbar switches. This machine is designed as an execution vehicle for systolic algorithms and has a high intercell bandwidth of 80 M bytes/sec. Some programs developed for computer vision application are 100×100 matrix multiplication, 512×512 FFT, 3×3 convolution, and the Hough transform.

capability, which is implemented with the boolean *n*-cube topology.

Though the problem tackled in this thesis does not require symbolic processing capabilities yet, we have to note that these capabilities will be needed as well either for high level computer vision problems or for general AI kind of tasks on top of computer vision. From this aspect, it is quite worthwhile to observe that efficient support of symbolic AI data representations, particularly, the predicate calculus in NON-VON or the semantic networks in Connection Machine, were considered from the beginning of the design phase of these machines.

3.1.1 Massively Parallel Processor

The Massively Parallel Processor (MPP) is organized of three basic components: a sequential controller, parallel array of processing elements (PEs), and a staging memory [Batc 83], [Pott 85].

The parallel array of PEs is interconnected in a 128×128 square mesh. It is possible for 128 columns of 1-bit data to be moved from/to staging memory to/from the parallel array. Each PE can communicate with its four neighbors. In MPP, each PE can send a bit to its neighbor in a single machine cycle, where the nominal machine cycle time is 100 nanosecond. On chip, the PEs are connected in a 2 by 4 mesh.

Each PE is a bit-serial unit, and contains five 1-bit registers and a 1-bit ALU which can perform boolean and arithmetic operations. Each PE is also connected to an off-chip random access memory storing 1024 bits. Since this address bus can be expanded up to 16 address lines, the PE memory can be expanded to 65,536 bits per PE.

Each PE has also a shift register whose depth can be set, under program control, from 2 to 30. With this shifter, floating point operations, such as aligning floating point fractions or normalizing floating point results, can be processed quite efficiently. For example, for a parallel array unit of size 128×128, the actual execution speed of 470 million addition operations per second, 291 million multiplication operations per second, and 165 million division operations per second have been reported with 32-bit floating point data format as shown in Table 3-1.

In the MPP, a lot of engineering effort has been spent on the staging memory. It is a large multidimensional-access memory. It is divided into main-stager, an input sub-stager, and an output sub-stager; words are 64-bit wide in main-stager and single bit wide in sub-stager. Staging memory buffers the data between the parallel array and a front-end computer; it can also reformat data.

It has been shown that the MPP provides very powerful support for certain image processing operations like the Fast Fourier Transform (FFT) which requires communication between pixels or points located far apart in the image. The staging memory allowed data to be permuted and routed at high speed rates to array memory for computation of FFT's of varying size and precision.

Operations	Execution speed (millions of operations per second)		
	8-bit integers	16-bit integers	32-bit floating point
Addition, Subtraction	6553	3343	470
Multiplication	1861	538	291
Division	1545	484	165

Table 3-1: Speed of Typical MPP Operations[15]

[15]taken from [Gilm 83, p. 166]

3.1.2 NON-VON Supercomputer

In NON-VON, each PE contains a on-chip memory (a 64 word 1-bit RAM and a 64 word 8-bit RAM), five 1-bit registers, five 8-bit registers, an 8-bit ALU, and two special combinatorial networks, called the I/O switch and the RESOLVE circuit [Shaw 84a].

The I/O switch is a matrix of pass transistors that routes data between the two internal buses and the three inter-PE communication buses (parent, left child, and right child) in the course of executing inter-PE communication instructions. Depending on the particular instruction, these switches may be configured in such a way as to support parent/child tree communications or left/right linear neighbor communications.

Any global communication from the bottom to the top of the tree can be performed by a sequence of level by level tree communications from the leaf to the top.

Supported by the tree communication capability, machine vision applications using hierarchical representations like quadtrees were developed [Huss 84]. The algorithms developed span different levels of computer vision tasks. They include image correlation, connected component labeling, the computation of geometric properties, and the Hough transform. Furthermore, the conducted research proposed two enhancements missing in original NON-VON architecture. These additions, part of later NON-VON architecture, are described below.

First, multiplication capability has been enhanced by combining the already existing 8-bit adder and shifter with a newly added barrel shifter. Two 8-bit numbers can be multiplied to produce 16-bit result in 28 cycles, about 7.0 microseconds, assuming a 250 nanosecond cycle time.

Second, the original communication paradigms supported by I/O switch, level by level tree communication between parent and children or one-dimensional array communication between left and right linear neighbors, did not support the local communication need of two-dimensional pixel arrays. The communication capability has been enhanced with mesh connections at leaf level of tree; leaf nodes account for half of the PEs in the system. Since mesh communication instructions are provided, NON-VON can perform all computations that require local cellular communications as efficiently as other machines can do.

Here, a labeling scheme that incorporates leaf mesh on binary tree is illustrated. Inorder labeled binary tree is shown in Figure 3-1. In quadtree representation, often used to represent region in machine vision, a node has four sons, labeled NW, NE, SW, and SE. In one particular algorithm that constructs a quadtree from a binary image, pixels are traversed recursively in NW, NE, SW, and SE order starting from the smallest block to progressively bigger ones [Same 80]. The same sequence is used to construct leaf mesh from leaf PEs of binary tree. Resulting mesh labeling at the leaf level is shown in Figure 3-2. This mesh labeling scheme can be easily extended to any number of coarse levels above the finest one at the leaf, due to the recursive nature of labeling algorithm. The mesh labeling at the coarse level just above the leaf is shown in Figure 3-3.

In terms of the execution speed, the instruction can be divided into three groups. All internal instructions and mesh communication instructions are executed in one machine cycle. All tree communication instructions involving data transfer between adjacent tree levels are executed in two machine cycles. Linear neighbor communication instructions and the RESOLVE instruction, where PEs even not at adjacent tree levels communicate with each other in a single instruction execution time, takes three machine cycles to complete execution.

The other applications developed on NON-VON includes the work on database and the production system [Shaw 84b]. In particular, [Hillyer 86] investigated the algorithms, performance analyses, and simulation results for the execution of database queries and production systems on NON-VON.

3.1.3 Connection Machine

In the first prototype of the Connection Machine (CM), there are 65,536 PEs [Hill 86]. Each PE contains 8 general purpose 1-bit registers, 4,096 bits of external memory, and a simple serial arithmetic logic unit. Each chip contains 16 PEs and one router unit which sends global communication messages through a packet switching network. The processor/router chip runs at a clock rate of 4 MHz. The topology of the communication network is a boolean n-cube[16] with a

[16]A boolean n-cube is an n dimensional cube; each vertex of the cube has a single neighbor in each of n orthogonal directions in n dimensional space. There are 2^n vertices in a boolean n-cube and each vertex is connected to n other vertices, one in each dimension.

```
                                            16
                                      /            \
                                 /                      \
                            /                                \
                       8                                          24
                    /     \                                   /        \
                  /          \                              /             \
              4                 12                        20                 28
            /   \             /    \                    /    \             /    \
          2       6         10       14               18       22        26       30
         / \     / \       /  \     /  \             /  \     /  \      /  \     /  \
        1   3   5   7     9    11  13   15          17   19  21   23   25   27  29   31
```

Figure 3-1: Inorder Labeled Binary Tree

```
 1   3   9  11

 5   7  13  15

17  19  25  27

21  23  29  31
```

Figure 3-2: Mesh Labeling at Leaf Level

```
 4  12

20  28
```

Figure 3-3: Mesh Labeling at Coarse Level

router at each vertex. On chip, the PEs are connected in a 4 by 4 square mesh, with each PE connected to its four nearest neighbors. This two-dimensional mesh pattern is extended across multiple chips into a larger mesh. The mesh communication system does not involve the router.

In the second prototype of the CM released recently, the size of the external memory was extended from 4K to 64K bits and a floating point arithmetic accelerator of single or double precision was added [TMC 87a]. The new prototype can be equipped with a fast frame buffer and can achieve I/O transfer rates of 320 Mbytes per second.

The flexibility of the global packet switching communication network supports irregular and dynamic communication patterns as well as regular ones. But the speed of the network communication mechanism is much slower than the mesh connections. Compared to the bandwidth of the mesh communication, the bandwidth of the packet switching communication is much smaller, roughly 1000 times smaller for the worst case and 30 times smaller for the typical case as shown in Table 3-2.

Two low level programming methodologies for programming the CM were presented in [Chri 84].

1. *n*-cube Model: Several low level operations can take advantage of the specific topology of the communication network, a boolean *n*-cube. This model exploits the fact that the fastest communication is between neighbors in the boolean *n*-cube.
2. Tree Model: This programming methodology deals with algorithms for binary trees of processors in the CM. It is used to support a graphical abstraction with arbitrary fan out, such as semantic networks. Two types of trees are considered:
   - Calculated Trees: The address of the parent and two children of a branch are calculated as a function of the address of the branch. Calculated trees are usually projected onto some other topology so that it can be treated as a tree. For example, a binary tree can be imposed on the boolean *n*-cube.
   - Explicit Trees: The address of the parent and children of a branch are stored explicitly by the branch. The advantage of explicit trees is that they can be manipulated quite easily.

For the high level languages, parallel versions of C and LISP have been developed [TMC 86]. In the LISP implementation, storages should be reclaimed through the garbage collection process. It is interesting that the Connection Machine supports the *parallel consing* as a primitive operation, where free cells are consed in parallel by exploiting the boolean *n*-cube topology [Chri 84, p. 49].

The utilization of several communication modes of the Connection Machine in implementing a set of low and intermediate level vision modules are reported in [Litt 86] and [Litt 87]. The implemented modules comprise edge detection, Hough transforms, and connected component labeling. And a variety of geometrical algorithms were designed for the Connection Machine, including several convex hull algorithms.

Classes of Permutations	Communication speed (bits per second)
Worst Case	3.2×10^{7}
Typical Case	1.0×10^{9}
2-D Pattern	3.3×10^{10}
FFT Pattern	5.0×10^{10}

Table 3-2: Communication Bandwidth of the CM[17]

3.2 Model of SIMD Computation

3.2.1 Requirements on the Architecture from the Application

As we will see in next chapter, a parallel architecture to support the particular structure of our application demands the following characteristics:[18]

- fine grained,
- SIMD,

[17]taken from [Hill 86, p. 71]

[18]When we compare the characteristics proposed here with the set of algorithmic criteria proposed by Grimson (see section 2.2.1), we see that ours is quite similar to, or has grown naturally out of, Grimson's criteria.

- local communication,
- global communication.

Many of the operations ought to be perfomed simultaneously on a subset of nodes properly chosen at each moment. Identical operations are carried out upon the data at each selected node. This first property naturally leads to a fine grained processor and a SIMD mode of execution.

Secondly, the matrix involved is sparse. In particular, in the depth interpolation problem even an interior node far removed from the region boundary interacts only with 12 neighboring nodes. Therefore, mesh interconnections between nodes are sufficient for handling all the local communication needs for matrix-vector multiplication.

Thirdly, what is needed as well is a fast global summary capability. In the iterative methods that were investigated, we need to compute various vector norms, a matrix norm, and inner products. This global communication need can be met well by any global network mechanisms, for instance, the tree topology or the boolean *n*-cube topology, superimposed on the underlying mesh.

3.2.2 Derivation of Abstract Basic SIMD Models

In this section, we derive two abstract models of SIMD computation. Various features of these abstract models will be extracted separately from the SIMD machines we have reviewed before.

The time complexity analysis of parallel machines involves two factors: the internal computational speed of each PE and the communication speed to move around data between the PEs.

The computational speed of each PE depends on the complexity of the hardware circuitry built into it. In computer vision applications, we have a tremendous amount of data to be processed. Therefore, it forces the designer to design each PE as simply as possible to accommodate more and more PEs. Nevertheless, we need good computational capability as well, such as the floating point calculations as required in this application.

One such PE design which made effort to meet these computational requirements is that of the MPP. We took the numbers of 32-bit floating point execution speed from Table 3-1 and converted them into equivalent machine cycles in Table 3-3. We took into account the 128×128 square mesh size and 100 nanosecond machine cycle time of the MPP.

In MPP (as in NON-VON and Connection Machine) mesh communication instructions, which handle local communications, execute in a single machine cycle. We assumed a single bit data path between PEs, and that floating point data is moved from the memory of one PE to the other PE's memory. For the transfer of each bit, we assumed a 5 machine cycle sequence: broadcast a source address, read a bit, send it through the mesh connection, broadcast a destination address, and then write it. Therefore, for 32-bit floating point data, it will take 160 machine cycles for completion.

For our global communication needs, we assumed the tree topology of NON-VON. We assumed that the instructions which carry out the tree communications between adjacent levels execute in 2 machine cycles following the experience of the NON-VON chip and prototype system design and implementation efforts. We assumed again a single bit data path between PEs. For the transfer of each bit, we assumed a 6 machine cycle sequence: broadcast a source address, read a bit, send it through the tree connection, broadcast a destination address, and then write it. Therefore, for 32-bit floating point data, it will take 192 machine cycles for completion.

Our discussion is summarized in Table 3-3. We will make extensive use of this model of SIMD computation in later chapters.[19]

For global communication, we could have also assumed the topology of the Connection Machine, where global communication is handled by the boolean *n*-cube topology. When the number of the communications bandwidth for the FFT Pattern in Table 3-2 is converted into equivalent machine cycles, we get 63, i.e., it will take 63 machine cycles to move a single bit to the cube neighbor. In the Connection Machine, there is a single bit data path between chips. We took into account the 256×256 square mesh size, the 4 MHz clock, and 4,096 routers. For 32-bit floating

[19]The speed of operations for our abstract model machine can be compared with the timings for the recent prototype of the Connection Machine, reported in tables of [Litt 87, p. 18] and [TMC 87a, p. 60]. For arithmetic operations on single precision floating point numbers, addition, subtraction, and multiplication take 25 microseconds while division takes 66.7 microseconds. For mesh communication, NEWS access of 32-bit takes 80 microseconds.

point data, it will take 2,016 machine cycles. The reason why we have such big numbers compared to the ones we have derived before with the tree topology is that the messages are delivered across each dimension in sequence for every *petit cycle* in the Connection Machine.

When we replace the tree topology with the boolean n-cube topology, we get another abstract model of SIMD computation. This one is shown in Table 3-4. We will call the model in Table 3-3 as Model I and the other in Table 3-4 as Model II.

3.2.3 A Global Summation Algorithm

An example is analyzed in detail to illustrate the power of the abstract SIMD models we have derived. Suppose that we have a $s \times s$ mesh and that s is some power of 2, i.e., $s = 2^k$ for some k. We want to compute the global sum of 32-bit floating point numbers at selected nodes of square mesh. For the sake of the simplicity, we assume that every node is enabled. In fact, this simple global summation algorithm turns out to be the part of the computation required for the calculation of a L_2-norm of a vector or an inner product of two vectors. For the vector calculation, we assume that each element of the vector is stored at the same address in the memory of each PE.

First, the tree topology is assumed for global communications and the Model I is applied to carry out the analysis of computational and communication cost associated with this algorithm. We assume that the square mesh is located at the leaf of the tree. As shown in Figure 3-1, PEs at the leaf of the tree account for the half of PEs in the tree, i.e., there are total of $2s^2 - 1$ PEs in the tree. The summation proceeds from the tree level just above the leaf to the top of the tree in following fashion: select PEs of current tree level, get data from left child, get data from right child, add these two data to get a sum, and then transfer control to the level just above. Throughout the computation, identical instructions are executed concurrently for the PEs of the same tree level. For each tree level, we have two data moves and one addition operations. At the end we have the global sum at the root node of the tree and the data from the root PE is read out to host computer. This operation requires one more data move. Since the number of levels is $(log_2 s^2)$, it takes $2\,(log_2 s^2) + 1$ tree communications and $(log_2 s^2)$ addition operations. By multiplying the number of operations by the number of machine cycles defined in Table 3-3, we

Operations	Data Type	Execution speed (machine cycles)
Addition, Subtraction	32-bit fl. point	348
Multiplication	32-bit fl. point	563
Division	32-bit fl. point	993
Mesh Communication	1 bit	1
Mesh Communication	32-bit fl. point	160
Tree Communication	1 bit	2
Tree Communication	32-bit fl. point	192

Table 3-3: Speed of Typical Operations for Abstract Model Machine (Model I)

Operations	Data Type	Execution speed (machine cycles)
Addition, Subtraction	32-bit fl. point	348
Multiplication	32-bit fl. point	563
Division	32-bit fl. point	993
Mesh Communication	1 bit	1
Mesh Communication	32-bit fl. point	160
Boolean *n*-cube Communication	1 bit	63
Boolean *n*-cube Communication	32-bit fl. point	2016

Table 3-4: Speed of Typical Operations for Abstract Model Machine (Model II)

get the total number of machine cycles, $(2 \times 192 + 348) \times (log_2 s^2) + 192$. For the 128×128 mesh, the total number of machine cycles is 10440, substituting s with 128.

Now, the boolean n-cube topology is assumed for global communications and the Model II is applied. To make our analysis simpler, it is assumed that each PE can talk directly to all of its own cube neighbors, i.e., global communication capability of the boolean n-cube is available at the PE level. Since we have s^2 PEs the size of the cube is $(log_2 s^2)$. The summation proceeds in following fashion for $i = 0, 1, \ldots, (log_2 s^2) - 1$: select PEs whose i-th bit in address is 0, get data from my cube neighbor whose i-th address bit is different from me, add the data obtained from my cube neighbor to my own in order to get a sum, and then transfer control for next address bit. At each step, we have one data move and one addition operations. At the end we have global sum at the PE whose address is 0 and data of this PE is read out to host computer. This operation requires one more data move. In summary, it takes $(log_2 s^2) + 1$ boolean n-cube communications and $(log_2 s^2)$ addition operations. By multiplying the number of operations by the number of machine cycles defined in Table 3-4, we get the total number of machine cycles, $(2016 + 348) \times (log_2 s^2) + 2016$. For the 128×128 mesh, the total number of machine cycles is 35112.

In summary, we can point out two aspects of the computation regardless of the particular topology employed. First, the communication pattern is quite regular and there is no collisions. Second, two summation algorithms are good from the point of view of the roundoff error analysis. The roundoff errors due to floating point arithmetic are smaller. Summing by pair leads to an error with a small cumulative constant ($\approx log_2 s^2$) whereas standard summing up on a serial machine has a constant $\approx s^2$.

The global summation algorithm was analyzed in detail. In a likewise manner, the other global operations such as counting, average, minimum, maximum, logical OR, and logical AND can be executed in $O(log_2 s)$ time as well.

3.2.4 Derivation of Abstract Extended SIMD Models

The pyramid machine, where several possible VLSI designs of this machine is proposed in [Dyer 81], can efficiently perform many quadtree and pyramid algorithms.[20]

In one organization of the pyramid machine, each node is connected to nine neighbors - its father processor on the level above it, its four son processors on the level below it, and its four nearest neighbors on its own level. This scheme can be conveniently implemented on a tree machine. Instead of a single mesh connection at the leaf of the tree, the mesh connections can be extended to every other tree levels from the leaf. [See the labeling schemes in Figure 3-1, 3-2, and 3-3.]

Now, the basic model described in section 3.2.2 can be generalized to include multiple mesh connections of different resolutions.

The mapping between each node and its four sons will be interpreted in two ways. In the first case, there is no special distinctions between its sons. In the second case, we think of mesh connections at two adjacent levels in terms of coarse and fine resolutions. Suppose that the node at coarse level is assigned with the (image) coordinate (x_i, y_j). Then the coordinates of its four son nodes can be assigned as follows: (x_i, y_j) for NW child, $((x_i + x_{i+1}) / 2, y_j)$ for NE child, $(x_i, (y_j + y_{j+1}) / 2)$ for SW child, and $((x_i + x_{i+1}) / 2, (y_j + y_{j+1}) / 2)$ for SE child. This assignment will be used later when we implement multigrid methods. [See Figure 4-1 and the equation (67) in section 4.3.]

Though we shall discuss the implementation of the multigrid method on a machine with the tree topology and multiple mesh connections, a realization of coarse and fine levels on a machine with the boolean n-cube topology, for example, the Connection Machine, is considered briefly. Since no separate physical PEs exist for the coarse level PEs, not like as in the pyramid and the tree machine, the coarse level PEs should be mapped to the finest level PEs.[21]

A simple scheme proposed here requires the doubling of the storage space allocated and handles

[20]See [Dyer 79] for a theoretical description of many of these algorithms.

[21]Recall our discussion in section 3.2.3. Suppose that we have a $s \times s$ mesh. For the tree topology, there are $2s^2 - 1$ PEs in the system. But for the boolean n-cube topology, there are just s^2 PEs in the system.

up to 4 coarse levels above the finest level. Suppose that the node at NW child position is assigned with the coordinate (x_i, y_j). Then the nodes at coarse levels having (x_i, y_j) as their coordinates are mapped to NW, NE, SE, and SW positions. For example, refer back to Figure 3-1, 3-2, and 3-3. PE4 is mapped to PE1 and PE16 is mapped to PE3. Under this arrangement, the communication between the adjacent fine and coarse levels are handled using a simple mesh communication instruction, except between the finest and the coarse level just above it where the interlevel communication is done by the moves to different memory locations in the same PEs. For the intralevel communication at the level l, where l is 1 for the coarsest and L for the finest level, the data is sent through the mesh connections to the nodes 2^{L-l} distances away. For example, refer back to Figure 3-1, 3-2, and 3-3. The intralevel communication from PE4, which is mapped to PE1, to PE12, which is mapped to PE9, is done by sending data from PE1 to PE9 via PE3.

In the pyramid machine and the tree machine with the multiple mesh connections, the intralevel communication at any level takes same amount of time because of the available physical connections. However, even though the mesh communication instructions can be executed in a single machine cycle, the length of the physical wire is longer at coarser levels.[22]

3.3 Conclusion and Summary

In this chapter, we reviewed three SIMD machines, the Massively Parallel Processor, the NON-VON Supercomputer, and the Connection Machine. We stressed the arithmetic and communication need to execute the efficient numerical methods, in our case, the adaptive Chebyshev acceleration and the conjugate gradient methods, for the iterations of the sparse SPD matrices, which are encountered frequently in several processes in early vision. We derived the abstract SIMD models to support the computational requirement and applied them to the global summation algorithm as an exercise. We analyzed the time taken for execution of this algorithm. In next chapter, we shall apply the basic and extended models to numerical methods in order to analyze the space and time complexity of the computation.

[22]Needless to say, the length of the physical wire is much more longer for any global communication networks, whether the supporting topology is tree or boolean n-cube.

We can always underestimate the heavy need of number crunching in early vision. The eventual introduction of the floating point arithmetic hardware in the Connection Machine testifies to this point.

We extended the basic SIMD model to handle the multigrid method as well. The iterations on the finest level only with either of two efficient methods improve already the computational efficiency greatly, compared with the slow methods, such as the Gauss-Seidel. But the multigrid mode of execution of these efficient and other methods on several coarse and fine levels speeds up the iteration process further. In the next chapter, we will review the mathematical theory behind the iterative methods and discuss the parallel implementation aspect of them under the abstract models of parallel computation developed in this chapter.

4. Iterative Methods

We review the well developed mathematical theory behind the iterative methods of numerical analysis and then discuss the parallel implementation of algorithms on a particular parallel architecture, a fine grained SIMD machine with local and global communication networks.

The single theoretical assumption we make for the adaptive Chebyshev acceleration and the conjugate gradient methods is that the matrix is SPD. In the parallel implementation, we require another assumption that the matrix is sparse for practical reasons. Note that these two assumptions are applicable to, but independent of, the depth interpolation problem itself.

4.1 Adaptive Chebyshev Acceleration Method

The derivation of the adaptive Chebyshev acceleration method when applied to any SPD matrix is shown first along with the introduction to basic iterative methods. Our discussion is based on [Youn 81] and in many places uses his equations and development. We show further how the adaptive Chebyshev acceleration method for sparse SPD matrices, where the Jacobi method is chosen as the underlying basic iterative method, can be run on a chosen parallel architecture.

4.1.1 Basic Iterative Methods

In section 2.2.2.3, the depth interpolation problem has been cast as solving a set of linear equations,

$$Au = b \tag{5}$$

where A is a given $n \times n$ SPD matrix and b is a given $n \times 1$ vector.

Using one of several known basic iterative methods, the equation (5) can be solved by the following iterative process

$$u^{(i+1)} = Gu^{(i)} + k, \qquad i = 0, 1, 2, \ldots \tag{6}$$

where G is the iteration matrix for the method and k is an associated vector.

We assume that

$$G = I - Q^{-1}A, \qquad k = Q^{-1}b \tag{7}$$

for some nonsingular matrix Q. The assumptions of (7) together with the fact that A is nonsingular imply that α is a solution to the related system

$$(I-G)u = k \tag{8}$$

if and only if α is also the unique solution to (5), i.e., $\alpha = A^{-1}b$.

An iterative method (6) whose related system (8) has a unique solution α which is the same as the solution of (5) is said to be *completely consistent*. Suppose that $\{u^{(i)}\}$ is the sequence of iterates determined by (6). Then complete consistency implies that if the sequence $\{u^{(i)}\}$ converges to some vector $\bar{u}$, then $\bar{u} = \alpha$.

The iterative method (6) is *convergent* if for any initial approximation $u^{(0)}$ the sequence $u^{(1)}$, $u^{(2)}$, ... determined by (6) converges to the unique solution $\alpha = A^{-1}b$. A necessary and sufficient condition for convergence is that $S(G) < 1.0$ where the *spectral radius* $S(G)$ of the real matrix G is defined as the maximum of the absolute values of the eigenvalues of G.

The error vector $\varepsilon^{(i)} \equiv u^{(i)} - \alpha$ satisfies the relation

$$\varepsilon^{(i)} = G^i \varepsilon^{(0)}. \tag{9}$$

Therefore, we have

$$\|\varepsilon^{(i)}\|_\beta \le \|G^i\|_\beta \|\varepsilon^{(0)}\|_\beta. \tag{10}$$

Thus, $\|G^i\|_\beta$ gives a measure by which the norm of the error has been reduced after i iterations.

The *average rate of convergence* for a basic iterative method (6) is defined by [Youn 81, p. 20]

$$R_i(G) = -(1/i)(log_e \|G^i\|_\beta). \tag{11}$$

It can be shown that if $S(G) < 1.0$, then

$$\lim_{i \to \infty} (\|G^i\|_\beta)^{1/i} = S(G). \tag{12}$$

Hence the *asymptotic rate of convergence* is defined by

$$R_\infty(G) = \lim_{i \to \infty} R_i(G) = -(log_e S(G)). \tag{13}$$

Frequently we refer to $R_\infty(G)$ as the *rate of convergence*. We see that as $S(G)$ approaches 1.0, the rate of convergence decreases. Small values of $S(G)$ (that is, $S(G)$ positive and near zero) gives a high convergence rate.

There are several well known basic iterative methods: the Jacobi, the Gauss-Seidel, the successive overrelaxation (SOR), and the symmetric successive overrelaxation (SSOR) methods. However, methods other than these basic iterative methods are used in practice because of the slow convergence rates of the basic iterative methods. The rates of convergence can be accelerated by two major classes of accelerations: polynomial acceleration methods or nonpolynomial acceleration methods. Note that the multigrid method is one of the nonpolynomial acceleration methods.

Before we proceed to discuss two basic iterative methods, some properties of real symmetric matrices are reviewed [Youn 81, p. 4].

Theorem 2: If the $n \times n$ matrix A is real and symmetric, then

1. the eigenvalues λ_i, $i = 1, \ldots, n$, of A are real, and
2. there exists a set of n real eigenvectors $\{\xi^{(i)}\}$ of A, i.e.,
   a. $A\xi^{(i)} = \lambda_i \xi^{(i)}, \quad i = 1, \ldots, n,$
   b. $\{\xi^{(i)}\}$ is a basis for the associated vector space, i.e., $\{\xi^{(i)}\}$ is a set of n linearly independent vectors, and
   c. $(\xi^{(i)}, \xi^{(j)}) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$

When the eigenvalues of the matrix A are real, we let $m(A)$ and $M(A)$ denote, respectively, the smallest and largest eigenvalues of the matrix A. Furthermore, if the matrix A is symmetric, then for any nonzero vector v,

$$m(A) \leq (v, Av) / (v, v) \leq M(A).$$

4.1.1.1 Jacobi Method

The Jacobi method is defined by

$$a_{i,i} u_i^{(l+1)} = - \sum_{j=1, j \neq i}^{n} a_{i,j} u_j^{(l)} + b_i, \qquad i = 1, 2, \ldots, n, \tag{14}$$

where $A = (a_{i,j})$ and $b = (b_i)$ for $1 \leq i, j \leq n$.

Hence, the iteration matrix G is related to the matrix A by

$$g_{i,j} = \begin{cases} 0 & \text{if } i = j \\ -a_{i,j} / a_{i,i} & \text{otherwise,} \end{cases} \tag{15}$$

where $G = (g_{i,j})$.

The vector k is related to the vector b by

$$k_i = b_i / a_{i,i}, \tag{16}$$

where $k = (k_i)$.

The Jacobi method is convergent if and only if $S(G) < 1.0$. It can be shown that $S(G) < 1.0$, for example, if the matrix A is *irreducible* with *weak diagonal dominance* [Youn 81, p. 25].

In our depth interpolation problem, the largest eigenvalue of G, $M(G)$, is less than 1.0 since the Jacobi method is *symmetrizable*. (The formal definition of the symmetrization property and its associated results will be described shortly in next section.) However, the smallest eigenvalue of G, $m(G)$, is less than -1.0.[23] Therefore, the Jacobi method itself is not convergent.

Nevertheless, the Jacobi method can be employed as the underlying iterative method of any polynomial acceleration methods. In such a case, it offers two nice aspects from the point of view of parallel computation. First, as shown in the equation (14), the Jacobi method displaces old values with new values simultaneously. Therefore the computational step where the matrix iteration computation is performed using the Jacobi method may be parallelized. This observation was utilized in our implementation of the adaptive Chebyshev acceleration method where the entire steps of the computations have been parallelized. Second, as shown in the equation (15), the sparsity of the iteration matrix G is preserved. Both aspects lead to efficient parallel computation, described in detail later.

4.1.1.2 Gauss-Seidel Method

The Gauss-Seidel method is defined by

$$a_{i,i}u_i^{(l+1)} = -\sum_{j=1}^{i-1} a_{i,j}u_j^{(l+1)} - \sum_{j=i+1}^{n} a_{i,j}u_j^{(l)} + b_i, \qquad i = 1, 2, \ldots, n. \tag{17}$$

When A is SPD, it can be shown that the Gauss-Seidel method always converges [Youn 81, p. 27].

In our depth interpolation problem, the Gauss-Seidel method always converges because A is SPD. This method has the advantage of being easy to implement and can serve as a quick and dirty solution. But it suffers from two aspects. Its convergence rate is rather slow and the method is inherently serial, thus not amenable to parallel computation.

[23]See the numerical result in section 5.2. There, $m(G)$ is estimated to be close to -2.2.

4.1.2 Extrapolated Method

For most of the acceleration methods, it is not necessary that the basic iterative method (6) be convergent. Normally, it is sufficient that the method be symmetrizable in the following sense.

Definition 3: The iterative method (6) is *symmetrizable* if for some nonsingular matrix W the matrix $W(I-G)W^{-1}$ is SPD. Such a matrix W is called a *symmetrization matrix.*

With this definition, we have the following results [Youn 81, p. 21].

Theorem 4: If the iterative method (6) is symmetrizable, then

1. the eigenvalues of G are real,
2. the largest eigenvalue $M(G)$ of G is less than 1.0, and
3. the set of eigenvectors for G is a basis for the associated vector space.

The symmetrization property need not imply convergence. If the iterative method (6) is symmetrizable, then the eigenvalues of G are less than 1.0 but not necessarily less than 1.0 in absolute value. Hence, the iterative method is not always convergent. However, there always exists a so-called extrapolated method based on the iterative method (6) which is convergent whenever the basic iterative method is symmetrizable.

The extrapolated method applied to the iterative method (6) is defined by

$$u^{(i+1)} = \gamma(Gu^{(i)} + k) + (1-\gamma)u^{(i)} = G_{[\gamma]}u^{(i)} + \gamma k, \tag{18}$$

where

$$G_{[\gamma]} \equiv \gamma G + (1-\gamma)I. \tag{19}$$

Here γ is a parameter that is often referred to as the "extrapolation factor." If the basic iterative method is symmetrizable, then the optimum value $\bar{\gamma}$ for γ, in the sense of minimizing $S(G_{[\gamma]})$, is given by

$$\bar{\gamma} = 2/(2-M(G)-m(G)). \tag{20}$$

Moreover, it easily follows that

$$S(G_{[\bar{\gamma}]}) = (M(G)-m(G))/(2-M(G)-m(G)) < 1. \tag{21}$$

Thus, the optimum extrapolated method is convergent.

In general, the Jacobi and the SSOR methods are symmetrizable while the Gauss-Seidel and the SOR methods are not. When the matrix A is SPD, the Jacobi method is symmetrizable with

$W = D^{1/2}$ where D is a diagonal matrix whose diagonal elements are taken from the matrix A.

When the basic iterative method is Jacobi, the extrapolated method is also called as the *weighted or damped Jacobi* method. [See the multigrid literature [Brig 87, p. 10] or the discussion in [Terz 84, p. 108].] Note that with $\gamma = 1$ we have the original Jacobi iteration method.

The Jacobi method is chosen as the underlying basic iterative method in this work since it is much simpler than the SSOR method, another symmetrizable basic iterative method.

4.1.3 Optimal Chebyshev Acceleration Method

The polynomial acceleration method, which involves the formation of a new vector sequence from linear combinations of the iterates obtained from the basic iterative method (6), is one of the approaches used to accelerate the rates of convergence of the basic iterative methods. With Chebyshev acceleration method, we assume again that the basic iterative method (6) is symmetrizable.

Consider a vector sequence $\{u^{(i)}\}$ determined by

$$u^{(i)} - \alpha = Q_i(G)(u^{(0)} - \alpha),$$

where $Q_i(G) = q_{i,0}I + q_{i,1}G + \cdots + q_{i,i}G^i$ is the matrix polynomial such that

$$\sum_{j=0}^{i} q_{i,j} = 1, \qquad i = 0, 1, \ldots .$$

Furthermore, the *virtual spectral radius* of $Q_i(G)$ is defined by

$$\overline{S}(Q_i(G)) = \max_{m(G) \le \lambda \le M(G)} |Q_i(\lambda)|. \tag{22}$$

The *virtual average rate of convergence* for a polynomial acceleration method is defined by

$$\overline{R}_i(Q_i(G)) = -(1/i)(\log_e \overline{S}(Q_i(G))), \tag{23}$$

and provided the limit exists, the *virtual asymptotic rate of convergence* is given by

$$\overline{R}_\infty(Q_i(G)) = \lim_{i \to \infty} \overline{R}_i(Q_i(G)). \tag{24}$$

We seek the particular polynomial acceleration method which is obtained by choosing the matrix polynomial sequence $\{Q_i(G)\}$ such that $\{\overline{S}(Q_i(G))\}$, $i = 1, 2, \ldots$, is minimized.

The Chebyshev polynomial of degree i is defined by the recurrence relation

$$T_0(g) = 1,$$

$$T_1(g) = g,$$

$$T_{i+1}(g) = 2gT_i(g) - T_{i-1}(g), \qquad i \geq 1.$$

Let $P_i(\lambda)$ be defined in terms of Chebyshev polynomials as

$$P_i(\lambda) = T_i(g(\lambda)) / T_i(g(1)),$$

where

$$g(\lambda) = (2\lambda - M(G) - m(G)) / (M(G) - m(G)).$$

It now follows that the polynomial $P_i(\lambda)$ is the unique polynomial which satisfies

$$\max_{m(G) \leq \lambda \leq M(G)} |P_i(\lambda)| = \max_{m(G) \leq \lambda \leq M(G)} |Q_i(\lambda)|. \tag{25}$$

We refer to the polynomial acceleration method based on $P_i(G)$ as the optimal Chebyshev acceleration method which has the form

$$\delta^{(i)} = Gu^{(i)} + k - u^{(i)}, \tag{26}$$

$$u^{(i+1)} = \rho_{i+1}(\gamma\delta^{(i)} + u^{(i)}) + (1 - \rho_{i+1})u^{(i-1)}, \tag{27}$$

where

$$\gamma = 2 / (2 - M(G) - m(G)), \tag{28}$$

$$\rho_1 = 1,$$

$$\rho_2 = 1 / (1 - .5\sigma^2),$$

$$\rho_{i+1} = 1 / (1 - .25\sigma^2\rho_i), \qquad i \geq 2, \tag{29}$$

and

$$\sigma = (M(G) - m(G)) / (2 - M(G) - m(G)). \tag{30}$$

We now examine the convergence rate of the optimal Chebyshev acceleration method. After a small amount of algebra, we obtain the virtual spectral radius of $P_i(G)$ as

$$\overline{S}(P_i(G)) = 2r^{i/2} / (1 + r^i), \tag{31}$$

where

$$r = (1 - \sqrt{1 - \sigma^2}) / (1 + \sqrt{1 - \sigma^2}). \tag{32}$$

Thus from (23) and (31), the average virtual rate of convergence for the optimal Chebyshev acceleration method is

$$\overline{R}_i(P_i(G)) = -\frac{1}{2}(log_e r) - \frac{1}{i}(log_e \frac{2}{1+r^i}), \qquad (33)$$

and the asymptotic rate of convergence defined by (24) is

$$\overline{R}_\infty(P_i(G)) = -\frac{1}{2}(log_e r). \qquad (34)$$

From (33) and (34), it easily follows that $\overline{R}_i(P_i(G)) < \overline{R}_\infty(P_i(G))$ for all finite i. In fact, it can be shown that $\overline{R}_i(P_i(G))$ is an increasing function of i. However, many iterations are often required before the asymptotic convergence is achieved. For example, if we have $r^i \approx 0.1$ after i iterations, then the average virtual convergence rate for these i iterations is only about one-half of its value for later iterations when the asymptotic convergence rate is achieved.

We can compare the optimum Chebyshev acceleration method with the optimum extrapolated method defined by the equation (18). For the optimum extrapolated method, we have by the equation (21) that

$$R_\infty(G_{[\gamma]}) = -log_e \sigma. \qquad (35)$$

For σ close to 1.0, we can show that

$$\overline{R}_\infty(P_i(G)) \sim \sqrt{2}\sqrt{R_\infty(G_{[\gamma]})}. \qquad (36)$$

Thus, the optimum Chebyshev acceleration method is an order of magnitude faster than the optimum extrapolated method.

The convergence rate of the optimal Chebyshev acceleration method is fastest when the largest eigenvalue, $M(G)$, and the smallest eigenvalue, $m(G)$, of the iteration matrix G for the related basic method are known. In Lee's work mentioned in section 2.2.3, he estimated the lower and upper bounds of the smallest and largest eigenvalues, although he worked on the Chebyshev method, not on the Chebyshev acceleration method.

In the depth interpolation problem, it is impossible to estimate the bounds of eigenvalues a priori due to the flexible nature of the matrix. For example, at nodes where the depth constraints exist, both the right- and the left-hand side of the matrix equation, $Au = b$, are modified, as seen in the equation (2). Since the matrix G is related to the matrix A by the equation (15), $m(G)$ and $M(G)$ depend only on the left-hand side of the original equation (5). Therefore, $m(G)$ and $M(G)$ depend on the number of depth continuous nodes in the region, the shape of the region, and the distribution of the constraints. However, $m(G)$ and $M(G)$ do not depend on the actual values of the constraints, since the constraint values appear on the right-hand side of the equation (5) only. Furthermore, $m(G)$ and $M(G)$ depend also on the choice of the values for the parameters β^h and α^h. [For example, see the equations (2) or (3).]

In general, the optimal estimates of the $m(G)$ and $M(G)$ are not known a priori but can be computationally determined by using the adaptive Chebyshev acceleration method.

4.1.4 Adaptive Chebyshev Acceleration Method

When estimates m_E and M_E are used for $m(G)$ and $M(G)$, respectively, the adaptive Chebyshev acceleration method has the form

$$\delta^{(i)} = Gu^{(i)} + k - u^{(i)}, \tag{37}$$

$$u^{(i+1)} = \rho_{i+1}(\gamma\delta^{(i)} + u^{(i)}) + (1 - \rho_{i+1})u^{(i-1)}, \tag{38}$$

where

$$\gamma = 2/(2 - M_E - m_E), \tag{39}$$

$$\begin{aligned} \rho_1 &= 1, && \text{if } p = 0,^{24} \\ \rho_2 &= 1/(1 - .5\sigma_E^2), && \text{if } p = 1, \\ \rho_{p+1} &= 1/(1 - .25\sigma_E^2\rho_p), && \text{if } p \geq 2, \end{aligned} \tag{40}$$

and

$$\sigma_E = (M_E - m_E)/(2 - M_E - m_E). \tag{41}$$

Because the basic iterative method is symmetrizable, we have the inequality $M(G) < 1.0$. Under the adequately general assumption of $m_E < M_E < 1.0$ and $m_E \leq m(G)$, it can be shown that the asymptotic rate of convergence of the adaptive Chebyshev acceleration method is an increasing function of M_E for $M_E \leq M(G)$. Also, the asymptotic rate of convergence is relatively insensitive

[24]Here, p is the degree of the Chebyshev polynomial currently being used.

to the estimate m_E as long as $m_E \leq m(G)$. For example, if $M(G)$ is close to 1.0 and $m(G) < -1.0$, then m_E need satisfy only $(m(G) - m_E) / |m(G)| \leq 0.1$ in order that the increase in the number of iterations using m_E be less than 4%. But using an estimate m_E which is considerably less than $m(G)$ may significantly increase the number of iterations required for convergence. However, iterative divergence may result if m_E does not satisfy $m_E \leq m(G)$.

In the adaptive procedure, a test is made during each iteration to determine whether or not the acceleration parameters currently being used are satisfactory. If the present parameters are judged unsatisfactory, the adaptive procedure then gives new improved estimates for the optimum acceleration parameters.

We implemented two adaptive Chebyshev acceleration procedures as given in [Youn 81]. In one algorithm (Algorithm 6-4.1 [Youn 81, p. 107]), the initial estimate of m_E is input and is not changed throughout computation. The estimate M_E is updated upward and converges to $M(G)$ from below. In the other algorithm (Algorithm 6-5.1 [Youn 81, p. 117]), estimates of both eigenvalues are updated. When the initial m_E is too high, it is adjusted downward. If not, it is not changed. The other estimate, M_E, is updated in the same fashion as in the previous algorithm. As both values approach their true values, the algorithm's rate of convergence increases.

We shall use several different vector and matrix norms. The definition of the L_2- and L_∞- vector norms that we use are the following:

$$\|v\|_2 \equiv (v, v)^{1/2} = (\sum_{j=1}^{n} |v_j|^2)^{1/2}, \tag{42}$$

$$\|v\|_\infty \equiv \max_{j=1,2,\ldots,n} |v_j|. \tag{43}$$

We also use a matrix norm

$$\|G\|_\infty \equiv \max_{i=1,2,\ldots,n} (\sum_{j=1}^{n} |g_{i,j}|). \tag{44}$$

Under the stronger assumption of $m_E < M_E < M(G) < 1.0$ and $m_E \leq m(G)$, one of the ways to compute the initial estimate of $m(G)$ is by computing a reasonable lower bound based on the matrix norm; that is, $m(G) \geq -\|G\|_\infty$ [Youn 81, p. 63]. The assumption that $M_E < M(G)$ can be satisfied easily. All that is required is that the initial estimate of M_E being smaller than $M(G)$.

The iterations are to be terminated whenever some norm of the error vector becomes sufficiently small. In the adaptive Chebyshev acceleration method, the pseudoresidual vector $\delta^{(i)}$ defined by the equation (37) is related to the error vector $\varepsilon^{(i)}$ by

$$\delta^{(i)} = (G - I)\varepsilon^{(i)}. \tag{45}$$

Under the stronger assumption of $m_E < M_E < M(G) < 1.0$ and $m_E \le m(G)$, it can be shown that for any L_β-norm [Youn 81, p. 70]

$$\lim_{i \to \infty} (\|\delta^{(i)}\|_\beta / \|\varepsilon^{(i)}\|_\beta) = 1 - M(G). \tag{46}$$

Thus, provided that i is sufficiently large and the current estimate M_E is approximately equal to $M(G)$, we have

$$\|\varepsilon^{(i)}\|_\beta \approx \|\delta^{(i)}\|_\beta / (1 - M_E). \tag{47}$$

Often, a relative error measure is desired rather than an absolute error measure; i.e., to terminate the iterative process whenever

$$\frac{\|\varepsilon^{(i)}\|_\beta}{\|\alpha\|_\eta} \le \zeta, \tag{48}$$

where ζ is the desired accuracy.

Using (47) and the approximation $\|\alpha\|_\eta \approx \|u^{(i+1)}\|_\eta$, we obtain the termination test

$$\frac{\|\varepsilon^{(i)}\|_\beta}{\|\alpha\|_\eta} \approx \frac{1}{1 - M_E} \frac{\|\delta^{(i)}\|_\beta}{\|u^{(i+1)}\|_\eta} \le \zeta. \tag{49}$$

Usually, the L_η-norm should be the L_∞-norm or the same L_β-norm used to measure $\varepsilon^{(i)}$.

4.1.5 Parallelization

The parallelization of the adaptive Chebyshev acceleration computation is now discussed in detail. The computation proceeds in two stages: pre-computation and a sequence of iterations until the convergence is reached.

As the model of computation for the discussion given in this section, we are assuming the abstract SIMD model we derived before in section 3.2.2. Specifically, we are referring back to the model in Table 3-3, where we have the mesh connection for the local communication and the tree connection superimposed on the underlying mesh for the global communication. In following discussion, we assume that the size of the mesh at the bottom of the tree is $s \times s$.

4.1.5.1 Pre-computation Stage

At the pre-computation stage, we compute the matrix A using a set of computational molecules in SIMD fashion with four types of given inputs: depth discontinuities, depth constraints, orientation discontinuities, and orientation constraints. For each node, it computes the necessary multiplication factors for each of 12 neighboring nodes and itself. The right-hand side vector b is also computed at this time. Once the matrix A is computed, an initial estimate of m_E can be also computed. The tree connections are used to calculate the L_∞-norm of G, $\|G\|_\infty$, and m_E might be initialized with the negated L_∞-norm, $-\|G\|_\infty$. The initial estimate of M_E can be simply set to 0.0 when a better estimate is not available.

4.1.5.2 Iteration Stage

At each iteration, computation goes through several steps. Here, the attention is focused on the calculation of the next iterate where all the matrix-vector multiplication and other vector operations are performed. The computations are (Algorithm 6-4.1 [Youn 81, p. 107][25]):

- $\delta^{(i)} = Gu^{(i)} + k - u^{(i)}$;
- compute $\|\delta^{(i)}\|_2$, $\|\delta^{(i)}\|_\beta$;
- $u^{(i+1)} = \rho_{i+1}(\gamma\delta^{(i)} + u^{(i)}) + (1 - \rho_{i+1})u^{(i-1)}$;
- compute $\|u^{(i+1)}\|_\eta$.

The vector norm $\|\delta^{(i)}\|_2$ is computed for parameter estimation, while $\|\delta^{(i)}\|_\beta$ and $\|u^{(i+1)}\|_\eta$ are computed for the iteration termination test.

Computation of $u^{(i+1)}$ is straightforward. Each node stores each element of the vectors so that a simple SIMD execution will update each one, independent of other nodes. No communication is needed. It would appear that storage is needed for each of the vectors $u^{(i-1)}$, $u^{(i)}$, and $u^{(i+1)}$. However, upon closer inspection it is easy to see that storage for only two vectors is required and that the elements of these vectors need not be moved each iteration.

Calculation of the L_2-norm of $\delta^{(i)}$, the L_β-norm of $\delta^{(i)}$ or the L_η-norm of $u^{(i+1)}$ are handled well using any global communication network. We assume here the tree topology for the global

[25]Complete listing of Algorithm 6-4.1 is given in Figure I-1 of Appendix, with further explanation on parameter estimation.

communication. Usually, the L_β-norm or the L_η-norm is either a L_2-norm or a L_∞-norm. When the L_2-norm is needed, every element in each PE is multiplied with itself. The summation of squared numbers is carried out from the bottom to the top of the tree, one level at a time[26]. The square root of the final resulting sum obtained at the top is the value desired. When the size of the mesh at the bottom of the tree is $s \times s$, the entire process takes $O(log_2 s)$ steps. When the L_∞-norm is desired, each PE calculates the absolute value of the element in it[27]. Then each PE located at non-leaf level of the tree compares two values coming from its own two sons and retains the bigger one. Both the comparison and the retaining of the biggest value is similarly carried out from the bottom to the top of the tree. The single value obtained at the top is the L_∞-norm desired. This too is a $O(log_2 s)$ process.

Finally, we are left with the calculation that involves the matrix-vector multiplication operation[28]. Computation of the pseudoresidual vector $\delta^{(i)}$ can be done in SIMD fashion using mesh interconnections only. The only step remaining at this point is the computation of matrix-vector multiplication term, $Gu^{(i)}$. In section 4.1.1.1, we have already seen that the Jacobi method is parallel (i.e., it simultaneously displaces old values with new values). Therefore, iterations based on the Jacobi method can be carried out in SIMD fashion with mesh interconnections to assemble current depth values of neighboring nodes. Furthermore, all the coefficients that contribute to this assembly are in the form

$$g_{i,j} = -a_{i,j} / a_{i,i}$$

since i is not equal to j. Since the factor in denominator, $a_{i,i}$, is common to all neighboring nodes, division by it is done only once, as the last step. By using the Jacobi method, neither explicit pre-computation of the matrix G nor any particular sophisticated ordering of matrix elements is needed. Put in other words, only local computation supported by the mesh topology is all that

[26]We have carried out the detailed analysis of the global summation algorithm in section 3.2.3. There, we analyzed the algorithm with two global communication networks, i.e., the first time with the tree topology and then with the boolean n-cube topology.

[27]The calculation of the absolute value is a constant operation when we deal directly with the representation of the floating point numbers. This operation just clears the sign of the fraction part to 0.

[28]The lower level detail of the SIMD implementation of this operation will be elaborated further in section I.2 of Appendix.

needed to multiply the vector $u^{(i)}$ by the matrix G.[29]

4.1.5.3 Space Complexity Analysis

Under our SIMD model of computation, each PE is provided with the fixed amount of memory. The space analysis shows the amount of memory to be allocated during the execution of the particular application.

The storage space needed in each PE is shown in Table 4-1.[30] The space allocated is divided into three groups. The first group is for the input vectors to the depth interpolation process, i.e., discontinuities and constraints informations. The second group is for the iteration matrix and its associated vector. The last group is for the output and other vectors which are updated at every iteration step. In the adaptive Chebyshev acceleration method, we need the space for three vectors, $u^{(i-1)}$, $u^{(i)}$, and $\delta^{(i)}$.

We analyzed the storage space requirement under general case, for example, when both the depth and the orientation constraints exist. But the simplification is possible when the depth constraints are present only.[31]

For this restricted case, the input to the depth interpolation process consists of the depth discontinuities and constraints only. In Table 4-1, the input vectors are reduced to 2 flags and 1 floating point number.

[29]Figure 2-4 shows the weighting factors for neighboring nodes of an interior node. But this figure depicts another important point which applies to any node. We can easily see that the depth value of any neighboring node can be brought in (in order to be multiplied by an appropriate weighting factor) through the execution of only one or two mesh communication instructions.

[30]A complete address map is given in Figure I-3 of section I.2 of Appendix.

[31]For instance, consider the depth constraints data from the stereo or laser rangefinder.

The coefficients of the matrix are stored in a form where $1 / h^2$ is factored out.[32] [See Figure 2-4 and the discussions in the section I.2 of Appendix.] For the restricted case of the depth constraints only, the coefficients which are contributed by four nodes at [0,2], [0,-2], [2,0], and [-2,0] are always 1.0, if they exist at all. [Refer to Figure 2-4 again.] Therefore, these four floating point numbers need not be stored. The marking of 4 corresponding flags is enough.

To summarize, we need 16 1-bit flags and 20 floating point numbers. Assuming 32 bits for the floating point number representation, we need 656 bits per PE. For the restricted case of the depth constraints only, we need 15 flags and 14 floating point numbers.

Description	Flag (1-bit)	Floating Point (32-bit)
Input Vectors	3	3
Iteration Matrix and Vector	13	14
Output and Other Vectors	0	3

Table 4-1: Storage Space Used (Adaptive Chebyshev Acceleration Method)

[32]For all three iterative methods, factoring out by a constant, $1 / h^2$, does not affect the computation. In the adaptive Chebyshev acceleration method, we use the Jacobi as the underlying basic iterative method. For the iteration matrix G, the coefficient for every non-diagonal element is given by $g_{i,j} = -a_{i,j} / a_{i,i}$. Since the coefficients of the A matrix are divided by another on the right-hand side, factoring out by a constant does not matter. In the conjugate gradient method, we adjusted every diagonal element of the matrix A to 1.0 in order to make the method converge. [See the discussion in section 4.2.3.2.] Since the division, $a_{i,j} / a_{i,i}$, is later done to pre-adjust the matrix A, storing coefficients after factoring out by a constant does not matter for this method as well. Finally, the Gauss-Seidel method is defined by the equation (17). In order to compute the $u_i^{(l+1)}$, all the terms on the right-hand side of the equation are divided by $a_{i,i}$, i.e., the division $a_{i,j} / a_{i,i}$ is encountered again.

4.1.5.4 Time Complexity Analysis

The parallel computations are analyzed in further detail in terms of basic operations and shown in three tables. In Table 4-2, we show *local* computations which require internal computations carried out inside of the PE or optionally using the mesh communication with nearby neighbors. The computations of two vectors, $\delta^{(i)}$ and $u^{(i+1)}$, fall into this group. The matrix-vector multiplication, $Gu^{(i)}$, which is an embedded step of the computation for the vector $\delta^{(i)}$, is presented under a separate column.

In Table 4-3, we show *global* computations which require tree communications for the global summary and other internal computations carried out prior to or during the global summary. The computations of three vector norms are put in this group.

In Table 4-4, we show the number of operations in *local* and *global* computations and then totals for each operation. Under the column designated "TOTAL", the number of operations are multiplied by the number of machine cycles which were defined before in Table 3-3. In summary, the total number of machine cycles required for each iteration of the adaptive Chebyshev acceleration method is given by $4392 \times (log_2 s) + 17712$. For the tree with 128×128 square mesh at the leaf, the total number of machine cycles is 48456.

The significance of the derived number is two-fold. First, it gives a rough estimate of the execution time. At the current stage of development, 100 nanoseconds is a reasonable machine cycle time for a typical SIMD machine. Therefore, prediction of execution time for an iteration of a particular method is possible. In next section, we shall carry out similar time complexity analysis for the conjugate gradient method as well. Second, when we compute the actual number of iteration steps, the overall performance comparison of the different iterative methods is possible.

The total time derived should be regarded as a lower bound. We analyzed the most time-consuming part only, i.e., the calculations and data moves associated with the floating point data items. We did not add up every detail of our implemented program. For example, we counted neither 1-bit marking and other SIMD operations, nor the processing of the scalar variables in the host processor. Nevertheless, the lower bound is not too low since the execution for SIMD part and the host processor can be overlapped.[33]

In the restricted case of the depth constraints only, a slightly faster execution is possible. In the matrix-vector multiplication $Gu^{(i)}$, a shifted-in depth value from each neighbor is continually multiplied by a corresponding pre-stored coefficient and added to an intermediate sum. For the restricted case, four coefficients are always 1.0, if they exist at all. Thus, four multiplication operations can be eliminated. However, note that the division $a_{i,j} / a_{i,i}$ can be done in the pre-computation stage for the general case. But this division operation should be performed at each iteration step for the restricted case.

In Table 4-2, to compute $Gu^{(i)}$ and $\delta^{(i)}$, for multiplication and division operations, we need 8 and 1 of them instead. In Table 4-4, we have 11 multiplication and 1 division operations for *local* computations but there is no change for *global* computations. The total number of machine cycles per iteration step is given by $4392 \times (log_2 s) + 16453$. For the tree with 128×128 square mesh at the leaf, the total number is 47197.

Before concluding this section, we discuss the time complexity analysis of the related iterative methods. When good estimates of $m(G)$ and $M(G)$ are available, say, through prior experimentation of the adaptive Chebyshev acceleration method, we can use them for $m(G)$ and $M(G)$. In this case, we can use either the optimal Chebyshev acceleration or the weighted Jacobi methods. Since the initial estimates are used with no further improvements to the end, more iteration steps might be needed. On the other hand, we do not compute the vector norms associated with the parameter estimation. Thus, in the parallel execution of these methods the global connections are not used and each iteration step takes less amount of time.

For the weighted Jacobi method, the equation (18) can be rewritten as

$$u^{(i+1)} = \gamma\delta^{(i)} + u^{(i)}, \tag{50}$$

using the definition of the pseudoresidual vector in the equation (26). [Compare the equation (50) with that of the optimal Chebyshev acceleration method, the equation (27).]

[33]For further details, see section I.1.2 of Appendix.

In Table 4-5, we show the analyzed result of the computations for the weighted Jacobi method. The total number of machine cycles for this method is 15099. In Table 4-6, we show similar result for the optimal Chebyshev acceleration method. The total number of machine cycles is 16573, slightly bigger than that of the weighted Jacobi method,[34] but significantly smaller than that of the adaptive Chebyshev acceleration method. In the restricted case of the depth constraints only, a slightly faster execution is also possible for two methods. The total number of machine cycles are 13840 and 15314 for the weighted Jacobi and the optimal Chebyshev acceleration methods, respectively.

We did not include the global computations associated with the iteration termination test in Table 4-6. There are three justifications for this decision. As explained already, by doing this, we can dispense with the global connections. Furthermore, we have not included the termination tests for other methods, for instance, the weighted Jacobi or the conjugate gradient methods. Lastly, when these methods are used under the multigrid approach, different termination tests are used for all methods.

4.2 Conjugate Gradient Method

The conjugate gradient method is well known in numerical analysis and its description can be found in many standard textbooks and papers, for example, in [Youn 81], [Golu 85], and [Wozn 80]. Here, we follow the description given in [Wozn 80].

We show the conjugate gradient algorithms for the solution of a large system of linear equations[36]

$$Ax = b \tag{51}$$

where A is an $n \times n$ SPD matrix and b is an $n \times 1$ vector.

[34]Recall that in the depth interpolation problem a node may interact with up to 12 neighbors. The term $Gu^{(i)}$, which is a part of the $\delta^{(i)}$ computation common to both methods, dominates the computational cost. Thus, the incremental cost incurred for the optimal Chebyshev acceleration method is small. In other problems where a node interacts with fewer neighbors, the relative portion of the incremental cost will be higher.

[36]In this section, we use x instead of u to denote the depth vector following the notations in [Wozn 80]. After this section, we will use both of them.

Operations	$Gu^{(i)}$	$\delta^{(i)}$	$u^{(i+1)}$
Addition, Subtraction	12	14	2
Multiplication	12	12	3
Division	0	0	0
Mesh Communication	16	16	0
Tree Communication	0	0	0

Table 4-2: Operations in Local Computation (Adaptive Chebyshev Acceleration Method)

Operations	$\|\delta^{(i)}\|_2$	$\|\delta^{(i)}\|_\infty$, $\|u^{(i+1)}\|_\infty$
Addition, Subtraction	$2\,(\mathit{log}\ s)$	$2\,(\mathit{log}\ s)$
Multiplication	1	0
Division	0	0
Mesh Communication	0	0
Tree Communication	$4\,(\mathit{log}\ s) + 1$	$4\,(\mathit{log}\ s) + 1$

Table 4-3: Operations in Global Computation (Adaptive Chebyshev Acceleration Method[35])

Operations	*local*	*global*	TOTAL (machine cycles)
Addition, Subtraction	16	$6\,(\mathit{log}\ s)$	$(6\,(\mathit{log}\ s) + 16) \times 348$
Multiplication	15	1	16×563
Division	0	0	0×993
Mesh Communication	16	0	16×160
Tree Communication	0	$12\,(\mathit{log}\ s) + 3$	$(12\,(\mathit{log}\ s) + 3) \times 192$

Table 4-4: Summary of Operations (Adaptive Chebyshev Acceleration Method)

[35]We assume the base of *log* is 2 unless stated otherwise.

Operations	$\delta^{(i)}$	$u^{(i+1)}$	TOTAL (machine cycles)
Addition, Subtraction	14	1	15×348
Multiplication	12	1	13×563
Division	0	0	0×993
Mesh Communication	16	0	16×160
Tree Communication	0	0	0×192

Table 4-5: Operations in Local Computation (Weighted Jacobi Method)

Operations	$\delta^{(i)}$	$u^{(i+1)}$	TOTAL (machine cycles)
Addition, Subtraction	14	2	16×348
Multiplication	12	3	15×563
Division	0	0	0×993
Mesh Communication	16	0	16×160
Tree Communication	0	0	0×192

Table 4-6: Operations in Local Computation (Optimal Chebyshev Acceleration Method)

4.2.1 Steepest Descent Method

We solve the equation (51) iteratively by constructing a sequence $\{x^{(i)}\}$ converging to the solution $\alpha = A^{-1}b$. Let B be a matrix which commutes with A: $BA = AB$.

Let $\|x\|_B = \|B^{1/2}x\| = (Bx, x)^{1/2}$, where $\|x\|_2 = (x, x)^{1/2}$.

The gradient iteration method constructs the sequence $\{x^{(i)}\}$ as follows. Let $x^{(0)}$ be an initial approximation and

$$\begin{aligned} r^{(i)} &= Ax^{(i)} - b, \\ x^{(i+1)} &= x^{(i)} - c_i r^{(i)}, \end{aligned} \tag{52}$$

where c_i is chosen in such a way that the error $e_{i+1} = \|x^{(i+1)} - \alpha\|_B$ is minimized. This yields

$$c_i = \frac{(r^{(i)}, B(x^{(i)} - \alpha))}{(r^{(i)}, Br^{(i)})}. \tag{53}$$

It is well known that $\{x^{(i)}\}$ converges to α and

$$\|x^{(i+1)} - \alpha\|_B \le \frac{(\kappa - 1)^{i+1}}{(\kappa + 1)^{i+1}} \|x^{(0)} - \alpha\|_B, \tag{54}$$

where $\kappa = \|A\| \, \|A^{-1}\|$ is the condition number of the matrix A, i.e., $\kappa = M(A) / m(A)$.

For $B = A$, the iteration (52), (53) is called the *steepest descent* method. It has, in general, very slow convergence and therefore is not recommended in practice. The conjugate gradient method is much more efficient.

4.2.2 Conjugate Gradient Method

Consider a class of iteration methods for which the error formula satisfies the relation

$$x^{(i)} - \alpha = W_i(A)(x^{(0)} - \alpha),$$

where W_i is a polynomial of degree at most i and $W_i(0) = 1$. We seek the polynomials W_i such that the error $e_i = \|x^{(i)} - \alpha\|_B$ is minimized. This means that the polynomials W_i are the solution of the following problem:

$$\|W_i(A)(x^{(0)} - \alpha)\|_B = \inf_{P \in W_i(0,1)} \|P(A)(x^{(0)} - \alpha)\|_B, \tag{55}$$

where $W_i(0, 1)$ is the class of polynomials of degree at most i normalized to unity at the origin. The solution of the problem (55) is given by the orthogonal polynomials derived as follows. Let

$$x^{(0)} - \alpha = \sum_{j=1}^{m} c_j \xi^{(j)}$$

where $\xi^{(j)}$ is an eigenvector of A associated with the eigenvalue λ_j: $A\xi^{(j)} = \lambda_j \xi^{(j)}$, $\lambda_1 < \lambda_2 < \cdots < \lambda_m$, with $m \le n$ and $c_j \ne 0$ for $j = 1, 2, \ldots, m$.

From the orthogonality of the polynomials W_i it follows that they satisfy a three-term recurrence formula. This form is defined as follows:

$$\begin{aligned} W_0(\lambda) &= 1, \\ W_1(\lambda) &= 1 - c_0\lambda, \\ W_{i+1}(\lambda) &= W_i(\lambda) - c_i\lambda W_i(\lambda) - u_i\{W_{i-1}(\lambda) - W_i(\lambda) + c_i\lambda W_i(\lambda)\}, \qquad i \ge 1, \end{aligned}$$

where

$$\begin{aligned} c_i &= \frac{(W_i, W_i)}{(\lambda W_i, W_i)}, \\ u_0 &= 0, \\ u_i &= \frac{\left(W_i - c_i\lambda W_i, \frac{1}{\lambda}(W_{i-1} - W_i) + c_i W_i\right)}{\left(W_{i-1} - W_i + c_i\lambda W_i, \frac{1}{\lambda}(W_{i-1} - W_i) + c_i W_i\right)}, \qquad i \ge 1. \end{aligned}$$

From this we get the three-term recurrence formula for the sequence $\{x^{(i)}\}$,

$$\begin{aligned} r^{(i)} &= Ax^{(i)} - b, \\ z^{(i)} &= x^{(i)} - c_i r^{(i)}, \\ y^{(i)} &= x^{(i-1)} - z^{(i)}, \\ x^{(i+1)} &= z^{(i)} - u_i y^{(i)}, \end{aligned} \tag{56}$$

where

$$\begin{aligned} c_i &= \frac{(r^{(i)}, B(x^{(i)} - \alpha))}{(r^{(i)}, Br^{(i)})}, \\ u_0 &= 0, \\ u_i &= \frac{(y^{(i)}, B(z^{(i)} - \alpha))}{(y^{(i)}, By^{(i)})}, \qquad i \ge 1. \end{aligned} \tag{57}$$

In infinite precision arithmetic, the conjugate gradient method (56), (57) solves the problem exactly in at most m steps, i.e., $x^{(k)} = \alpha$ for some $k \le m$.

From the equation (55) one can estimate the speed of convergence for initial approximations $x^{(i)}$, $i \le m$. Setting

$$P_i(\lambda) = T_i(g(\lambda)) / T_i(g(1))$$

in the equation (55), where T_i is the Chebyshev polynomial of degree i and

$$g(\lambda) = (2\lambda - \lambda_m - \lambda_1) / (\lambda_m - \lambda_1),$$

we get

$$\|x^{(i)} - \alpha\|_B \le 2 \frac{(\sqrt{a} - 1)^i}{(\sqrt{a} + 1)^i} \|x^{(0)} - \alpha\|_B, \tag{58}$$

where $a = \lambda_m / \lambda_1$. [Compare with the equation (54).]

In general, it seems that the choice $B = A^p$ for $p = 0$, 1 or 2 covers all cases of practical interest.

For $B = A^0 = I$ we minimize $\|x^{(i)} - \alpha\|$. For $B = I$, we cannot, in general, compute the coefficient c_i in the equation (57). To compute the coefficients c_i and u_i we assume that $A = M^* M$ and $b = M^* g$ for a nonsingular matrix M where M and g are given as data. This variant of the conjugate gradient method is called the *minimum error* method.

For $B = A^1$ we minimize $\|A^{1/2}(x^{(i)} - \alpha)\|$. This corresponds to the *classical conjugate gradient* method. After substituting B with A and $A\alpha$ with b, we have

$$c_i = \frac{(r^{(i)}, r^{(i)})}{(r^{(i)}, Ar^{(i)})},$$

$$u_0 = 0,$$

$$u_i = \frac{(y^{(i)}, Az^{(i)} - b)}{(y^{(i)}, Ay^{(i)})}, \qquad i \ge 1. \tag{59}$$

On examination of the equations (56) and (59), we find that we seem to need four matrix-vector multiplications. But two matrix-vector multiplications, $Ay^{(i)}$ and $Az^{(i)}$, can be eliminated with substitutions. Now we have

$$c_i = \frac{(r^{(i)}, r^{(i)})}{(r^{(i)}, Ar^{(i)})},$$

$$u_0 = 0,$$
$$u_i = \frac{(y^{(i)}, r^{(i)} - c_i Ar^{(i)})}{(y^{(i)}, Ax^{(i-1)} - Ax^{(i)} + c_i Ar^{(i)})}, \qquad i \geq 1. \tag{60}$$

For $B = A^2$ we minimize the residual vectors $r^{(i)}$, since $||A(x^{(i)} - \alpha)|| = ||r^{(i)}||$. This variant is called the *minimal residual* method.

Only $B = A^1$ will be used in this work, although analysis of the others is similar.

4.2.3 Parallelization

We show the parallel implementation of the classical conjugate gradient method defined by the equations (56) and (60). We assume the same model of computation used in section 4.1.5. We assume again that the size of the mesh at the bottom of the tree is $s \times s$.

4.2.3.1 Pre-computation Stage

The computation of the matrix A and the right-hand side vector b has been described in section 4.1.5.1. The discussion is not repeated here.

4.2.3.2 Iteration Stage

The matrix-vector multiplication operations, $Ax^{(i)}$ and $Ar^{(i)}$, of the conjugate gradient method proceed in the same fashion using the mesh connection as described in section 4.1.5.2.

In order for the conjugate gradient method not to diverge, every diagonal element of the matrix A was adjusted to 1.0 by division. Therefore, for non-diagonal elements of the matrix, we have similar coefficients, $a_{i,j} / a_{i,i}$, to those of the Jacobi method given in the equation (15). This led to further similarity in the computational steps of the matrix-vector multiplication operations for the conjugate gradient and the adaptive Chebyshev acceleration methods.

When the inner products are computed, two elements (drawn separately from two vectors) in each PE located at the leaf of the tree are multiplied together. Once the product is in place, the global summation is carried out from the bottom to the top of the tree using the tree connection. This too is a $O(log_2 s)$ process.

4.2.3.3 Space Complexity Analysis

The storage space needed in each PE is shown in Table 4-7. Compared to the adaptive Chebyshev acceleration method, the conjugate gradient method requires more space to carry out each iteration. In addition to the space for two vectors, $x^{(i-1)}$ and $x^{(i)}$, we need additional space to store the result of the matrix-vector multiplication, $Ax^{(i-1)}$, $Ax^{(i)}$, and $Ar^{(i)}$, and also the space for three other vectors, $z^{(i)}$, $y^{(i)}$, and $r^{(i)}$.

To summarize, we need 16 1-bit flags and 25 floating point numbers. In all, we need 816 bits per PE. For the restricted case of the depth constraints only, we need 15 flags and 19 floating point numbers. [See the discussion in section 4.1.5.3.]

Description	Flag (1-bit)	Floating Point (32-bit)
Input Vectors	3	3
Iteration Matrix and Vector	13	14
Output and Other Vectors	0	8

Table 4-7: Storage Space Used (Conjugate Gradient Method)

4.2.3.4 Time Complexity Analysis

The parallel computations are analyzed in further detail in terms of basic operations and shown in three tables. In Table 4-8, we show *local* computations. The computations of four vectors and two matrix-vector multiplications fall into this group. The matrix-vector multiplication operations, $Ax^{(i)}$ and $Ar^{(i)}$, which are embedded steps of the computation of the vector $r^{(i)}$ and the coefficient c_i, respectively, are presented together under a separate column.

In Table 4-9, we show *global* computations. The computations of four inner products to compute two coefficients, c_i and u_i, belong to this group.

In Table 4-10, we show the number of operations in *local* and *global* computations and then totals for each operation. Under the column designated "TOTAL", the number of operations are multiplied by the number of machine cycles. In summary, the total number of machine cycles required for each iteration of the conjugate gradient method is given by $5856 \times (log_2 s) + 34825$. For the tree with 128×128 square mesh at the leaf, the total number of machine cycles is 75817.

In the restricted case of the depth constraints only, a slightly faster execution is possible. [See the discussion in section 4.1.5.4.] In Table 4-8, to compute $Ax^{(i)}$, $Ar^{(i)}$, and $r^{(i)}$, we need 8 multiplication and 1 division operations. In Table 4-10, we have 18 multiplication and 2 division operations for *local* computations but there is no change for *global* computations. The total number of machine cycles per iteration step is given by $5856 \times (log_2 s) + 32307$. For the tree with 128×128 square mesh at the leaf, the total number is 73299.

4.3 Multigrid Method

We now discuss the extension of the iterative methods by a multigrid approach. For the theory of multigrid methods, we generally follow the description of [Terz 84], which is in turn based on the seminar work of [Bran 77]. We follow also that of [Brig 87], a recently published introduction to the subject with annotated suggested reading list. For another recent sophisticated treatment of the subject, see [McCo 87].

Operations	$Ax^{(i)}, Ar^{(i)}$	$r^{(i)}$	$z^{(i)}$	$y^{(i)}$	$x^{(i+1)}$
Addition, Subtraction	13	14	1	1	1
Multiplication	12	12	1	0	1
Division	0	0	0	0	0
Mesh Communication	16	16	0	0	0
Tree Communication	0	0	0	0	0

Table 4-8: Operations in Local Computation (Conjugate Gradient Method)

Operations	$(r^{(i)}, r^{(i)}), (r^{(i)}, Ar^{(i)})$	$(y^{(i)}, r^{(i)} - c_i Ar^{(i)})$	$(y^{(i)}, Ax^{(i-1)} - Ax^{(i)} + c_i Ar^{(i)})$
Addition, Subtraction	$2(log\ s)$	$2(log\ s) + 1$	$2(log\ s) + 2$
Multiplication	1	2	1
Division	0	0	0
Mesh Communication	0	0	0
Tree Communication	$4(log\ s) + 1$	$4(log\ s) + 1$	$4(log\ s) + 1$

Table 4-9: Operations in Global Computation (Conjugate Gradient Method)

Operations	*local*	*global*	TOTAL (machine cycles)
Addition, Subtraction	30	$8(log\ s) + 3$	$(8(log\ s) + 33) \times 348$
Multiplication	26	5	31×563
Division	0	0	0×993
Mesh Communication	32	0	32×160
Tree Communication	0	$16(log\ s) + 4$	$(16(log\ s) + 4) \times 192$

Table 4-10: Summary of Operations (Conjugate Gradient Method)

4.3.1 Multilevel Equations

We have considered the solution of a large system of linear equations $A^h u^h = f^h$. A total of $L - 1$ similar problems on increasingly coarser levels can be introduced to increase efficiency.

The hierarchy of problems is then given by the sequence of L linear systems of the form

$$A^{h_l} u^{h_l} = f^{h_l}, \quad 1 \le l \le L, \tag{61}$$

whose discrete solutions u^{h_l} for $1 \le l \le L$ constitute the hierarchy of full surface representations.

In general, solving the problems at coarser levels is faster because of two factors. First, the size of the matrices gets smaller. Second, the density of the constraints gets denser. However, it suffers from the loss of fine detail because of crude resolution.

To exploit the hierarchy of problems, the system can be solved at the coarsest level, and that solution can be used as an initial approximation in the iterative solution of the next finer level, proceeding in this way to the finest level L. This idea of using coarser grids to generate improved initial guesses is called *nested iteration*. However, it does not generate solutions having the accuracy of the finest level in any of the coarser levels. The way that a hierarchy of coarser solutions would maintain accuracies consistent with the solution of the finest level is to allow the coarser levels to access the high-resolution information in the finer levels. Multigrid algorithms provide such communication.

In general, the standard iterative methods, such as the weighted Jacobi or the Gauss-Seidel, decreases the error rapidly within the first few iterations, after which it decreases much more slowly. Any initial error can be divided between high- and low-frequency modes, that is, the oscillatory and smooth components of the error. The initial rapid decrease in error is due to the quick elimination of the high-frequency modes. The later slow decrease is due to the presence of the low-frequency modes, i.e., the iteration is much less effective in reducing the remaining smooth components. We can assume that enough *relaxation sweeps* on the fine grid eliminates the high-frequency components of the error. In fact, very few sweeps may be needed to nearly accomplish this.

The important point is that smooth modes on a fine grid look less smooth on a coarse grid. This suggests that when relaxation begins to stall, signaling the predominance of smooth error modes, it is advisable to move to a coarser grid, on which those smooth error modes appear more oscillatory and relaxation will be more effective.

Another powerful idea of using the residual equation to relax on the error is called *coarse grid correction*. In this procedure, we first relax on the fine level l until the convergence deteriorates, obtain an approximation v^{h_l}, and compute the residual $r^{h_l} = f^{h_l} - A^{h_l} v^{h_l}$. Then we relax on the residual equation $A^{h_{l-1}} e^{h_{l-1}} = r^{h_{l-1}}$ on the coarser level $l-1$ to obtain an approximation to the error $e^{h_{l-1}}$. Finally, we correct the approximation obtained on the fine level with the error estimate obtained on the coarser level: $v^{h_l} = v^{h_l} + e^{h_{l-1}}$.

The coarse grid correction acting on smooth modes produces smooth and oscillatory modes with very small amplitudes. Therefore, the coarse grid correction scheme is effective at eliminating smooth components of error [Brig 87, p. 75]. These two processes, relaxation and correction, complement each other remarkably. By applying them in tandem, the multigrid methods reduce the error very effectively.

The multilevel equations for L levels are given by [Terz 84, p. 110]

$$A^{h_l} u^{h_l} = g^{h_l}, \quad 1 \le l \le L, \tag{62}$$

where

$$g^{h_L} = f^{h_L},$$

$$g^{h_l} = A^{h_l}(I_{l+1 \to l} u^{h_{l+1}}) + I_{l+1 \to l}(g^{h_{l+1}} - A^{h_{l+1}} u^{h_{l+1}}), \quad 1 \le l \le L-1. \tag{63}$$

The original, right-hand side f^{h_l} of the l-th level problem occurs only on the finest level L. The right-hand sides of the coarser levels have been modified using information from the finer levels in order that the accuracy of the finest level be maintained throughout the coarser levels.

When the solution u^{h_l} of the equation (63) is available, the approximation on the fine level can be corrected by the replacement

$$u^{h_{l+1}} = u^{h_{l+1}} + I_{l \to l+1}(u^{h_l} - I_{l+1 \to l}\, u^{h_{l+1}}). \tag{64}$$

For the sequence of L linear systems in the equation (61) or (62), the system matrices are SPD. Suppose that the constraints at the finest level was obtained.[37] We can then show that the matrix at the finest level is SPD. [See the discussion in section 2.2.2.4.] Given the condition that the matrix at the finest level is SPD and the fact that the constraints at coarser levels are generated by sampling or by local averaging of values at the finer levels, we can show that the matrices at coarser levels are also SPD using similar arguments.

We can make another remark about the sequence of the smallest and largest eigenvalues, $m(G^{h_l})$ and $M(G^{h_l})$. In $A^{h_l} u^{h_l} = f^{h_l}$ and $A^{h_l} u^{h_l} = g^{h_l}$, the same matrix A^{h_l} is employed on the left-hand side of the equations. Note that $m(G^{h_l})$ and $M(G^{h_l})$ depend on the matrix A^{h_l} only.[38] Therefore, the estimates of the eigenvalues computed while solving the $A^{h_l} u^{h_l} = f^{h_l}$ can be used in the solution of $A^{h_l} u^{h_l} = g^{h_l}$ as well.

Either smaller size of the matrices or denser constraints leads to easier problems to solve. In the Chebyshev acceleration method, this amounts to smaller $M(G)$ values. In multigrid approach, both conditions, smaller matrices and denser constraints, are satisfied at coarser levels. Thus, the values of $M(G^{h_l})$ at coarser levels are smaller.

In a simplified multigrid implementation, a 2 : 1 decrease in grid resolution between adjacent levels is employed as shown in Figure 4-1. The grid nodes of coarser grids coincide with grid nodes on adjacent finer grids.

[37]For an instance of the generation of the depth constraints at the finest level, see the Grimson's approach which was discussed in section 2.2.1.

[38]Recall the discussion about the $m(G)$ and $M(G)$ at the end of section 4.1.3.

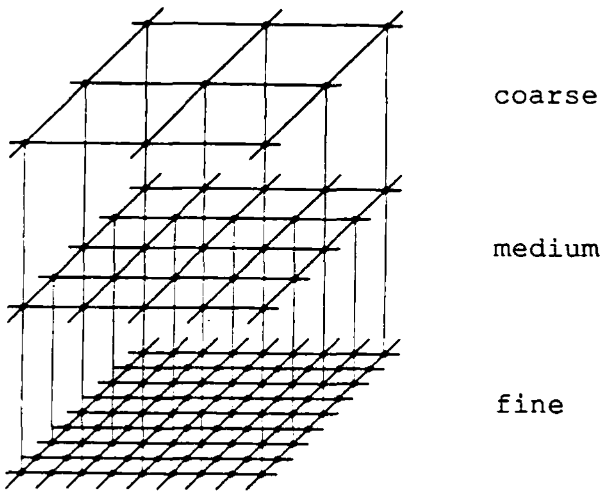

Figure 4-1: Typical Multigrid Organization[39]

4.3.1.1 Interlevel Computation

The issue of intergrid transfers is discussed at some length in [Bran 82].

For the fine-to-coarse restriction operation, $I_{l+1 \rightarrow l}$, simple injection or local averaging is used. In an injection, a coarse-grid node receives the value from the coincident fine-grid node.

For the coarse-to-fine prolongation operation, $I_{l \rightarrow l+1}$, polynomial Lagrange interpolation is employed. The two-dimensional interpolating polynomial of degree 3 in x and degree 3 in y is used wherever possible [Terz 84, p. 122]. When discontinuities occur such as near the region boundary, the degree of interpolation is reduced accommodating only nearby depth continuous nodes.

The two-dimensional Lagrange interpolating polynomial of degree m in x and degree n in y passing through the $(m+1)(n+1)$ points $(x_i, y_j, v(x_i,y_j))$, for $0 \le i \le m$ and $0 \le j \le n$, is given by

$$p_{m,n}(x,y) = \sum_{i=0}^{m} \sum_{j=0}^{n} X_{m,i}(x)\, Y_{n,j}(y)\, v(x_i, y_j), \tag{65}$$

with the Lagrangian interpolation coefficients

[39]taken from [Terz 85a, p. 157]

$$X_{m,i}(x) = \prod_{k=0,\,k\neq i}^{m} \frac{x - x_k}{x_i - x_k}, \quad 0 \le i \le m,$$

$$Y_{n,j}(y) = \prod_{k=0,\,k\neq j}^{n} \frac{y - y_k}{y_j - y_k}, \quad 0 \le j \le n. \tag{66}$$

When the third-degree interpolation is used, $m = n = 3$ are substituted into the equations above, (65) and (66). For four nodes at and near (x_1, y_1) at fine level, we have[40]

$$\begin{aligned}
p_{3,3}(x_1,y_1) &= v(x_1,y_1),\\
p_{3,3}((x_1+x_2)/2, y_1) &= (-v(x_0,y_1) + 9\,v(x_1,y_1) + 9\,v(x_2,y_1) - v(x_3,y_1))\,/\,16,\\
p_{3,3}(x_1, (y_1+y_2)/2) &= (-v(x_1,y_0) + 9\,v(x_1,y_1) + 9\,v(x_1,y_2) - v(x_1,y_3))\,/\,16,\\
p_{3,3}((x_1+x_2)/2, (y_1+y_2)/2) &= (\,v(x_0,y_0) - 9\,v(x_1,y_0) - 9\,v(x_2,y_0) + v(x_3,y_0)\\
&\quad - 9\,v(x_0,y_1) + 81\,v(x_1,y_1) + 81\,v(x_2,y_1) - 9\,v(x_3,y_1)\\
&\quad - 9\,v(x_0,y_2) + 81\,v(x_1,y_2) + 81\,v(x_2,y_2) - 9\,v(x_3,y_2)\\
&\quad + v(x_0,y_3) - 9\,v(x_1,y_3) - 9\,v(x_2,y_3) + v(x_3,y_3))\,/\,256.
\end{aligned} \tag{67}$$

4.3.2 Multilevel Coordination Schemes

In a *fixed scheme*, listed in Figure 4-2, the switching of levels occurs in a fixed manner from the coarsest level $l = 1$ to the finest level $l = L$ [Terz 84, p. 113]. (See also the *full multigrid V-cycle* algorithm in [Brig 87, p. 49].)

In the controlling procedure FMRA, a sufficient number of iterations are performed first to solve the coarsest level discrete system $A^{h_1} u^{h_1} = f^{h_1}$ to desired accuracy. Then the *currently finest* level, started with $l = 1$, is incremented. The first approximation on the new level is set by the interpolation and then the procedure FMC is invoked. When it terminates at level l, we have obtained a hierarchy of l representations. The *currently finest* approximation is then interpolated to the next finer level and the procedure FMC is called again until the finest level L is reached.

In the main computational procedure FMC, n_1 iterations are performed first at level l. It then performs a restriction to the next coarser level $l - 1$. Next, it calls itself recursively on the

[40]At coarse level l, for all i and j, we have $x_{i+1} - x_i = y_{j+1} - y_j = h_l$.

coarser level n_2 times. (In practice, only $n_2 = 1$ and $n_2 = 2$ are used.) Finally, it performs a prolongation from the coarser level back to level l, following up with n_3 more iterations on level l. On the coarsest level $l = 1$, the problem is solved to desired accuracy with the basic solution method, SOLVE. It can be easily shown that when FMC is invoked on level l it calls RELAX a total of $n_2^{l-k}(n_1 + n_3)$ times on level $k \neq 1$ and it calls SOLVE n_2^{l-1} times on level 1.

In general, the relaxation processes on the coarser scales suffer from increasingly large discretization errors, but they converge to the coarse solution relatively quickly since the size of the matrices is smaller and the density of the constraints is denser. Conversely, those on the finer scales are increasingly accurate, but exhibit a substantially slower response[43]. With the coarse-to-fine coupling, the fast response characteristics of the coarser relaxation processes is extended to the finer levels, but beyond a certain point the poor accuracy of the coarser levels corrupts the solution computed in the finest level. On the other hand, with the fine-to-coarse coupling, the accurate approximations computed on the finer levels improve the accuracy of the coarser approximations.

To resolve the dilemma, the interlevel coupling can be modified during the iterative process such that there is an initially strong but gradually weakening coarse-to-fine interaction, which accelerates convergence, and an initially weak but gradually strengthening fine-to-coarse interaction, which ultimately yields consistent accuracy on all levels [Terz 85a].[44]

For multilevel coordination schemes, there are actually two common approaches. The first, described above, is a fixed scheme, where the cycling parameters are chosen a priori and remain fixed throughout the course of the algorithm. The choice may be made on the basis of analysis or prior experimentation.

The second strategy is called an *accommodative scheme*. In this adaptive approach, the cycling

[43]For example, see the numerical result in Table 5-13 which will be discussed in next chapter.

[44]It is interesting to compare this control strategy with that of simulated annealing, which is a powerful and general method for finding global optima of functions that have many local optima. In [Gema 84], the schedule for reduction of temperature is given by $T = C / (log_e (1 + k))$ at the k^{th} iteration, where C is an appropriate energy constant. When T is large, energy increases are often accepted, enabling the system to jump out of local minima. As $T \rightarrow 0$, the system freezes, becoming almost deterministic in its descent towards a minimum of energy.

```
procedure FMC ($l, u, g$)

  If $l = 1$, then
     $u$ := SOLVE ($l, u, g$);
  else
  begin
     for $n_1$ times do $u$ := RELAX ($l, u, g$);

     $v$ := $I_{l \to l-1} u$;

     $d$ := $A^{h_{l-1}} v + I_{l \to l-1}(g - A^{h_l} u)$;

     for $n_2$ times do FMC ($l-1, v, d$);

     $u$ := $u + I_{l-1 \to l}(v - I_{l \to l-1} u)$;

     for $n_3$ times do $u$ := RELAX ($l, u, g$);
  end;


procedure FMRA

  Initialize $f^{h_1}, f^{h_2}, \ldots, f^{h_L}$. Clear $u^{h_1}, u^{h_2}, \ldots, u^{h_L}$ to zero.[41]

  $u^{h_1}$ := SOLVE ($1, u^{h_1}, f^{h_1}$);

  for $l$ := 2 to $L$ do
  begin
     $v^{h_l}$ := $I_{l-1 \to l}\, u^{h_{l-1}}$;

     FMC ($l, v^{h_l}, f^{h_l}$);
  end;
```

Figure 4-2: Multigrid Algorithm (Fixed Scheme)[42]

[41] Here, "zero" means $(0.0\ 0.0\ \ldots\ 0.0)^T$.

[42] taken from [Terz 84, p. 113-114]

parameters are determined on the run to account for variations in the patterns of convergence. An accommodative scheme requires a small overhead, but it can pay for itself by reducing unnecessary cycles and relaxation sweeps.

An example of an accommodative strategy is the following test, which determines when relaxation should end on a given level. The residual vectors, given by $r^{h_l} = g^{h_l} - A^{h_l} u^{h_l}$, are computed after successive relaxation sweeps.[45] When

$$\|r^{(new)}\|_{h_l} > \eta \, \|r^{(old)}\|_{h_l},$$

where η is a specified switching parameter, then relaxation on that grid is declared ineffective and a move is made to the next coarser grid. The parameter η can be determined for certain model problems. For instance, for the two-dimensional Poisson equation, $\eta \approx .6$ is reasonable [Brig 87, p. 61].

Terzopoulos used an algorithm with a fully accommodative scheme (Algorithm 7.1 [Terz 84, p. 110]), which is based on [Bran 77], as a main vehicle in his work. We will use the fixed scheme in next chapter to demonstrate the acceleration achieved by the multigrid approach.

4.3.3 Parallelization

The intralevel computation employs the standard relaxation. When serial relaxation methods such as the Gauss-Seidel is used, it is not amenable to the parallelization.

Terzopoulos used *work units* (WU) to measure the computational cost of multigrid methods following [Bran 77]. A work unit is defined as the amount of computation to perform one relaxation sweep on the finest level L. When the computation on a sequential processor is assumed, the work unit is proportional to the number of nodes where we can safely assume that same amount of computation is required for nearly all nodes. Since there are about one quarter the number of nodes on level $l-1$ as there are on level l, only $1/4^{L-l}$ work unit is required to perform a relaxation iteration on level l. Terzopoulos followed the convention of neglecting the cost of intergrid transfer operations which could amount to 15-20 percent of the cost of the entire

[45]Note that the parallel computation of the discrete L_2- or L_∞-norm of the residual vector, $\|r^{(n)}\|_{h_l}$, requires global connections as other vector norms.

cycle.

In our work, we shall use the iterative methods executable on a family of SIMD machines. We shall carry out the corresponding space and time complexity analysis of multigrid algorithms. As a measure of comparison for various iterative methods, we shall use principally the execution time, though work units can still be used. We shall return to this issue again in section 4.3.3.3.

The interlevel computation interacts with local nodes in parallel fashion so that this computational step is amenable to parallelization regardless of the method employed in the intralevel computation.

As our analysis will reveal, the amount of time taken to perform an interlevel computation, especially a prolongation operation, is not negligible compared to an intralevel computation, for example, an iteration for the adaptive Chebyshev acceleration or the conjugate gradient methods.

As the model of computation, we are assuming the extended SIMD model we derived before in section 3.2.4. Specifically, we assume that we have multiple mesh connections for the local communication at the number of fine and coarse levels sufficient enough to execute multigrid algorithms. We also assume tree connections, both for local interlevel computation and for global communication.

4.3.3.1 Pre-computation Stage

We assume that the discontinuities and constraints inputs are available at the finest level. Then, the discontinuities and constraints are propagated level by level to the coarsest one. A node at the coarse level is set to be depth continuous if at least one node at the adjacent fine level is continuous. For depth continuous nodes, the constraints can be obtained by sampling or by local averaging of values at the finer level.

Once the discontinuities and constraints are in place for all levels, the matrices A^{h_l} and their associated vectors f^{h_l} for $1 \leq l \leq L$ are computed simultaneously. However, a part of the computation, i.e., some terms contributed by the constraints molecules, may be computed separately for each level.

4.3.3.2 Space Complexity Analysis

For the multigrid algorithm, the increment of allocated memory is trivial. For the iterations on the finest level only, 1 bit flag is allocated to mark the depth continuous PEs at the leaf. For the multigrid method, instead of 1, we need L bits to mark the depth continuous PEs at each level separately. Another addition is the space for g^{h_l} vector in addition to f^{h_l} vector.

4.3.3.3 Time Complexity Analysis of Intralevel Computation

For the iterative methods at each level, we can use either the weighted Jacobi, the adaptive (and optimal) Chebyshev acceleration, or the conjugate gradient method. Now, the results derived before in section 4.1.5.4 and 4.2.3.4 can be used with slight modifications, taking into account the different size of the mesh at each level. Suppose that the size of the mesh at the finest level L is $s \times s$. Then the size of the mesh at level l is $(s / 2^{L-l}) \times (s / 2^{L-l})$.

At level l, the total number of machine cycles required for each iteration is given by $4392 \times ((log_2 s) - (L - l)) + 17712$ for the adaptive Chebyshev acceleration and $5856 \times ((log_2 s) - (L - l)) + 34825$ for the conjugate gradient methods, respectively. When good estimates of $m(G^{h_l})$ and $M(G^{h_l})$ are available, we can use the weighted Jacobi or the optimal Chebyshev acceleration method. Since these methods use only local connections, the computation time does not depend on the size of the mesh, i.e., independent of level l. The total number of machine cycles per iteration step is 15099 and 16573 for the weighted Jacobi and the optimal Chebyshev acceleration methods, respectively.

For the restricted case of the depth constraints only, the total number of machine cycles per iteration at level l is given by $4392 \times ((log_2 s) - (L - l)) + 16453$ for the adaptive Chebyshev acceleration and $5856 \times ((log_2 s) - (L - l)) + 32307$ for the conjugate gradient methods, respectively. The total number of machine cycles per iteration step is 13840 and 15314 for the weighted Jacobi and the optimal Chebyshev acceleration methods, respectively.

To measure the computational cost of multigrid algorithms, we can use work units or, more generally, the execution time. Under the framework of a sequential processor, work units are often used for a serial method such as the Gauss-Seidel. For a relaxation sweep on the finest level, we have 1.0 work unit. For three coarser levels, we have .25, .0625, and .015625, respectively. Under the framework of massive parallelism of a SIMD machine, identical operations are carried out simultaneously on a set of chosen PEs. Here, the execution time is a better measure since we have invested the hardware already. Instead of a few powerful processors, we have huge number of small though quite capable processors.[46] For an iterative method which uses both local and global communications such as the conjugate gradient, we have only small differences of execution time for fine and coarse grids. Suppose that the size of the mesh at the finest level is 128×128. On the finest level, an iteration of the conjugate gradient method takes 75817 machines cycles, while an iteration on three coarser levels take 69961, 64105, and 58249 machine cycles, respectively (with the only differences due to global computation). If we set the execution time on the finest level as 1.0, then we have .9228, .8455, and .7683 for the three coarser levels, respectively. Note the relatively small decreases compared to the work units. For an iterative method which uses only local communications such as the optimal Chebyshev acceleration, we would have same execution time for all levels. In next section, we will carry out the time complexity analysis of the computations required for the transfer of information between grids. By adding these costs together, we obtain a much more precise picture of parallel computation.

4.3.3.4 Time Complexity Analysis of Interlevel Computation

For the restriction operation, simple injection or local averaging is used. We consider simple injection first. In most cases, a coarse-grid node receives the value from the coincident fine-grid node, the NW child.[47] But for boundary nodes it may happen that NW child is not depth continuous. In this case, it receives the value from one of the children in the order of NE, SW, and SE child. This is done at two successive tree levels (i.e., one pyramid level). First, the data is received from the right child at the lower (finer) level and then overwritten by the data from the left child, thus favoring the data from children in the west direction. Then, a level up, identical operation is carried out to favor the data from children in the north direction. The entire operation takes four data moves through tree connection. [Refer back to Figure 3-1 and 3-2. See also

[46]In the series of Cray's supercomputers, a Cray 1, a Cray 2, and a Y-MP model have 1, 4, and 8 processors, respectively, while a Cray 3 and a Cray 4 will have 16 and 64 processors. In contrast, the Connection Machine has between 16K and 64K processors.

[47]Recall the discussion of coordinates assignment in section 3.2.4.

Figure 4-1.]

Now, we consider local averaging as the restriction operation. The depth data as well as the number of depth continuous nodes in the subtree are sent up through the tree connection. At the coarse level, the local average is obtained by dividing the sum of the depth data with the number of depth continuous nodes in 2×2 grid of the adjacent fine level.

For the prolongation operation, Lagrange interpolation is employed. The two-dimensional interpolating polynomial of degree 3 in x and y is used wherever possible. Near and on region boundaries, the degree is reduced to 2, 1, or even to 0. This Lagrange interpolation is a rather costly operation compared to the computation involved in iteration steps, though it is not required so often as iterations are. To show how costly it is, we can compare the third-degree interpolation formula in the equation (67) where 15 neighbors are involved with the nodal equations (1) or (2) where 12 neighbors are involved. Also, we can compare the prolongation operation, $I_{l-1 \to l}\, u^{h_{l-1}}$, in Table 4-11 with the nodal computations via the matrix-vector multiplication, $Ax^{(i)}$ and $Ar^{(i)}$ in Table 4-8 or $Gu^{(i)}$ in Table 4-2.

The restriction and prolongation operations are analyzed in terms of basic operations in Table 4-11 and 4-12. The total number of machine cycles required for each operation is obtained by multiplying the number of operations by the number of machine cycles defined in Table 3-3. In summary, the number of machine cycles for the prolongation operation in FMRA, $I_{l-1 \to l}\, u^{h_{l-1}}$, is 52383. The number for the restriction operation in FMC, $I_{l \to l-1}\, u$ and $A^{h_{l-1}} v + I_{l \to l-1}(g - A^{h_l} u)$ together, is 29912 or 36218, when simple injection or local averaging is used for the restriction operation, respectively. The number of machine cycles for the prolongation operation in FMC, $u + I_{l-1 \to l}(v - I_{l \to l-1}\, u)$, is 53847 or 57000, when simple injection or local averaging is used for the restriction operation, respectively. The execution time for these operations is same for all levels since these operations involve only local interactions between the adjacent coarse and fine level using the tree connections and identical processing is done in SIMD fashion for all PEs at the same tree level.

In the restricted case of the depth constraints only, a slightly faster execution is possible. To compute $A^{h_{l-1}} v + I_{l \to l-1}(g - A^{h_l} u)$, we need only 16 multiplication operations, instead of the numbers in Table 4-11 and 4-12. Similarly, we need only 2 or 3 division operations, instead of the numbers in Table 4-11 and 4-12, respectively. (For the matrix-vector multiplication operations, $A^{h_{l-1}} v$ and $A^{h_l} u$, see the discussion in section 4.2.3.4.) In summary, the number of machine cycles for the restriction operation in FMC, $I_{l \to l-1} u$ and $A^{h_{l-1}} v + I_{l \to l-1}(g - A^{h_l} u)$ together, is 27394 or 33700, when simple injection or local averaging is used for the restriction operation, respectively.

4.4 Conclusion and Summary

For the depth interpolation problem we investigated where the matrix A is SPD, the Jacobi method is not convergent but the other methods, the Gauss-Seidel, the weighted Jacobi, the optimal Chebyshev acceleration, the adaptive Chebyshev acceleration, and the conjugate gradient, are all convergent.

Among convergent methods, the Gauss-Seidel method is serial, thus not amenable to parallelization. But all other methods can be implemented on a parallel architecture, in this work, a fine grained SIMD machine with local and global communication networks. The iterations of the weighted Jacobi and the optimal Chebyshev acceleration methods critically depend on global parameters, the smallest and largest eigenvalues of the iteration matrix G, where good bounds for these two extreme eigenvalues should be obtained analytically or good estimates obtained through prior experimentation. However, the parallel implementation of these methods requires only local connections. The iterations of the adaptive Chebyshev acceleration and the conjugate gradient methods can be started with no global information, but the parallel implementation of these methods then demand both local and global connections. Nevertheless, since the adaptive Chebyshev acceleration method computationally determines the estimates of the extreme eigenvalues, the weighted Jacobi and the optimal Chebyshev acceleration methods can always use the estimates obtained.

The space and time complexity analysis reveals that the conjugate gradient method requires more storage space and longer execution time per iteration step than the adaptive Chebyshev acceleration method. But as will see in the next chapter, depending on the number of iteration steps required for convergence, the conjugate gradient method performs sometimes better in total

Operations	$I_{l \to l-1} u$	$I_{l-1 \to l} u^{h_{l-1}}$
Addition, Subtraction	0	40
Multiplication	0	28
Division	0	11
Mesh Communication	0	64
Tree Communication	4	8

Operations	$A^{h_{l-1}} v + I_{l \to l-1}(g - A^{h_l} u)$	$u + I_{l-1 \to l}(v - I_{l \to l-1} u)$
Addition, Subtraction	28	42
Multiplication	24	28
Division	0	11
Mesh Communication	32	64
Tree Communication	4	12

Table 4-11: Restriction (simple injection) and Prolongation Operations

Operations	$I_{l \to l-1} u$	$I_{l-1 \to l} u^{h_{l-1}}$
Addition, Subtraction	4	40
Multiplication	0	28
Division	1	11
Mesh Communication	0	64
Tree Communication	8	8

Operations	$A^{h_{l-1}} v + I_{l \to l-1}(g - A^{h_l} u)$	$u + I_{l-1 \to l}(v - I_{l \to l-1} u)$
Addition, Subtraction	32	46
Multiplication	24	28
Division	1	12
Mesh Communication	32	64
Tree Communication	8	16

Table 4-12: Restriction (local averaging) and Prolongation Operations

cost than the adaptive Chebyshev acceleration method, especially, when the latter is not started with good estimates of the extreme eigenvalues. Therefore, the nonadaptive former can serve as a good metric to check how the adaptive latter performs with given initial estimates.

Assuming multiple mesh connections with a tree topology, which has nearly the equivalent power of pyramid connections, the incremental storage space for the multigrid approach is rather insignificant. The execution time for the iterations on coarser levels takes less time since the global operations can be executed more quickly due to smaller mesh size. Even though the interlevel operations, prolongation and restriction processes, are not performed quite often as the iteration processes, the execution time for them are not negligible, especially, in the prolongation process where the two-dimensional Lagrange interpolation is employed.

The Gauss-Seidel, the weighted Jacobi, the optimal and adaptive Chebyshev acceleration, and the conjugate gradient methods converge when the matrix is SPD. But what about the multigrid acceleration of these methods? In our experimental results to be shown in next chapter, they converged and showed measurable degrees of acceleration. But theoretically, will they be valid only under the condition that the matrix at the finest level is SPD?

We can make both weak and strong statements. First, here is a weak statement. Given the above mentioned condition, we can show that the matrices at the coarser levels are also SPD. Therefore, the separate solutions at different resolutions are at least guaranteed to converge. [See the equation (61).] Nevertheless, regardless of the result of the theoretical question of convergence, the fast convergence rate at the coarser levels is still very attractive in the initial stage of the iterative process. Since the desired accuracy is quickly obtained at coarser levels, we can at least interpolate the resulting approximations to finer levels in order to get good initial guesses. Now, we can make a strong statement. For general problems, convergence analysis for multigrid methods is difficult. It is still an open area. However, we can give heuristic and qualitative arguments suggesting that the standard multigrid schemes, when applied to well-behaved problems, (for example, when the matrices are SPD), they not only work, but they work very effectively. Furthermore, convergence results for such problems can be proved quite rigorously [Brig 87, p. 54].[48]

We use a fixed scheme as the control strategy of multilevel coordination. Further study is called for the accommodative scheme when we employ iterative methods, such as the conjugate gradient or the adaptive (and optimal) Chebyshev acceleration, for relaxation sweeps.

A related issue is that one might think that optimal iterative methods may not be necessarily required for intralevel computation, especially since slower iterative methods such as the weighted Jacobi might not perform so badly at coarser levels. However, we should not ignore the fact that the optimal Chebyshev acceleration method is an order of magnitude faster than the weighted Jacobi method for the iterations on any size single-grid. [Refer back to the equation (36).] Recall that the size of the matrix is quite big; for 128×128 images, a typical size of the matrix is 10000×10000!

In conclusion, we arrive at the following suggestion. If the constraints of the images are not sparse, the execution carried out on the finest level only may be sufficient. For very sparse images, multigrid methods may be a better choice than the execution on the finest level only, since the multigrid methods may provide significantly faster response. In the multigrid methods, either the less powerful iterative methods, such as the weighted Jacobi, or the optimal methods, such as the adaptive (and optimal) Chebyshev acceleration or the conjugate gradient, may be used for relaxation sweeps. When we employ a fixed scheme as the control strategy of multilevel coordination, and the weighted Jacobi or the optimal Chebyshev acceleration as the relaxation methods, only local mesh and tree connections are sufficient, i.e., global connections are not needed. Nevertheless, even in this case, we may still need the adaptive Chebyshev acceleration method, at least in prior experimentations which are carried out for the separate solutions at different resolutions, in order to get good estimates of the smallest and largest eigenvalues of the iteration matrices. In the next chapter, we will show numerical results on several synthetic and real imagery, which behave in accordance with the suggestions we have drawn here.

[48]Same problem applies to relaxation algorithms in coarse-fine segmentation, edge linking, etc., throughout computer vision.

5. Numerical Results

We present the actual numerical results for the iterative methods that were discussed theoretically in previous chapter. For various surface reconstruction examples, we interpret the number of iterations under a SIMD model of parallel computation and show how fast the iterative methods will run on SIMD machines with local and global communication networks.

5.1 Overall Discussions of Images and Iterative Methods

5.1.1 Root Mean Square Error

When we solve the equation (51) iteratively, a sequence $\{x^{(i)}\}$ converging to the solution $\alpha = A^{-1}b$ is constructed where $x^{(i)}$ and α are $n \times 1$ vectors.

The root mean square error (RMSE) at the ith iteration is defined as follows:

$$RMSE^{(i)} = \left(\left(\sum_{j=1}^{n} [x_j^{(i)} - \alpha_j]^2\right) / n\right)^{1/2}. \tag{68}$$

In this chapter, we shall use the RMSE to compare the performances of the different iterative methods. The standard L_2- or L_∞- vector norms are also appropriate measures. Note that L_2-norm of the error vector and the RMSE is related. [Compare the definition of the RMSE in equation (68) with that of L_2-norm in (42).] There is another merit of using the RMSE rather than L_2- or L_∞- vector norms. In the multigrid method, we deal with several resolutions. At coarser levels, the dimension of the matrix, n, is smaller. Since the RMSE values are always normalized with respect to n, we can compare conveniently the results of iterations at coarse and fine levels.

For the real images, since we do not know the solution α, we cannot compute the RMSE values. Instead, we shall use the root mean square deviation (RMSD) which is defined as follows:

$$RMSD^{(i)} = \left(\left(\sum_{j=1}^{n_{cs}} [x_j^{(i)} - d_j]^2\right) / n_{cs}\right)^{1/2}, \tag{69}$$

where n_{cs} is the number of the known depth constraints. The summation is now restricted to the deviation of the computed depth values from the known depth constraints d.

Instead of the RMSE or RMSD, one can use different convergence criteria. For the single-grid algorithms, a convergence criterion specific to each iterative method may be used. For instance, for the adaptive Chebyshev acceleration method, see the iteration termination test in the equation (49). For the multigrid algorithms, the discrete L_2- or L_∞-norm of the residual vector may be used.

5.1.2 Iterative Methods

We will show the number of iteration steps to reduce the RMSE to specified fraction. For methods that can be implemented on a parallel architecture, we will compare the performance by the execution time, i.e., the number of machine cycles.

When the iterations are carried out on the finest level only, we will compare the performance of the conjugate gradient, the adaptive Chebyshev acceleration, and the Gauss-Seidel methods. In particular, we will show that on a parallel architecture the adaptive Chebyshev acceleration method executes faster than the conjugate gradient method if this method is started with more accurate initial estimate of the smallest and largest eigenvalues of the iteration matrix.

For the multigrid algorithms, we will compare the performance of the conjugate gradient, the adaptive (and optimal) Chebyshev acceleration, the weighted Jacobi, and the Gauss-Seidel methods. We will demonstrate the speed-up of multigrid execution over the single-grid algorithms on the finest grid only. We will show also that the optimal Chebyshev acceleration method with fairly good estimates of the eigenvalues executes faster than the adaptive Chebyshev acceleration or the conjugate gradient methods.[49]

5.1.3 Kinds of Images Used

We will investigate a small set of model problems in depth with various iterative methods. For an extensive set of experiments that might be run with the iterative methods studied in this work, see chapter 8 of [Terz 84].

[49] Recall that the optimal Chebyshev acceleration and the weighted Jacobi methods rely on the local computations only and therefore utilize local connections only.

5.1.3.1 Synthetic Images

The first synthetic image is a floating plane of constant depth, $\alpha = (1.0\ 1.0\ \ldots\ 1.0)^T$. The shape of the boundary of the plane is a square, with size 128×128. The depth discontinuities are assumed to be present outside of the square. Most of the nodes are interior nodes located well inside the boundary, as depicted in Figure 2-4. But along the boundary, there are several kinds of boundary nodes as well. For example, at one corner of the square, we have a boundary node as shown in Figure 2-5. The depth constraints are scattered randomly over the plane and the densities of the depth constraints are varied to 50%, 30%, and 15%.

The second synthetic image is a portion of a cylinder whose axis is parallel to the j direction. The synthetic depth for node $[i, j]$ is

$$\alpha_{[i,j]} = (1.0 - (i - r/2)^2 / (r)^2)^{1/2},$$

where $0 \leq i \leq 127$ and $r = 127.0$. The shape of the boundary is a square, with size 128×128. The depth constraints are scattered randomly as in the plane example for the single-grid experiments. But for the multigrid methods, the constraints lie along one axis only.

The last synthetic image is the upper hemisphere of a sphere where the depth is changing along both directions smoothly. The synthetic depth for node $[i, j]$ is

$$\alpha_{[i,j]} = (1.0 - ((i - r)^2 + (j - r)^2) / (r)^2)^{1/2},$$

where $0 \leq i, j \leq 127$ and $r = 63.5$. The shape of the boundary of the sphere is the biggest circle that can be contained in the square mesh, and all nodes outside the circle are assumed to be depth discontinuous. The size of the square mesh is 128×128. The depth constraints are scattered randomly, too. For some experiments, we assume that the orientation constraints are also available at all the nodes where the depth constraints exist. The densities of the constraints are varied to 50%, 30%, 15%, 5%, and 2%.

5.1.3.2 Real Images

For the real images, we used range data from the Utah range database [Hans 86]. Two examples of range data chosen for our experiments were those of a quasi-spherical object and the cylindrical portion of a soda can.

5.2 Discussion on Estimate of Extreme Eigenvalues

For the bulk of the simulation work, the first Young algorithm (Algorithm 6-4.1 [Youn 81, p. 107]) was used. In this algorithm, the estimate of $M(G)$ is updated upward during iterations but the estimate of $m(G)$ is not changed at all. When the estimate M_E is close enough to $M(G)$, the convergence is fast even in the case that m_E is not close enough to $m(G)$; in the adaptive Chebyshev acceleration method, M_E is more critical than m_E. Therefore, before embarking on this algorithm, we carried out an experiment to see how much we lose by running this algorithm with the reasonable and simple to compute initial estimates of $M_E = 0.0$ and $m_E = -\|G\|_\infty$. Another related problem is how we can get more accurate initial estimates to speed convergence.

In this experiment, the synthetic image used was a floating plane of constant depth, $\alpha = (1.0\ 1.0\ \ldots\ 1.0)^T$. The depth constraints were scattered randomly over the plane and the density of the depth constraints was 50%.

We used another more general algorithm of Young (Algorithm 6-5.1 [Youn 81, p. 117]), in which the estimate M_E is updated upward while the estimate m_E is updated downward if current estimate is bigger than the smallest eigenvalue, $m(G)$. This algorithm can also be used to provide more accurate initial estimates for subsequent images.

When we have no information at all, we can use 0.0 as initial estimate of M_E and pick a large enough number, say, -0.1 for m_E. When we started with these initial estimates, we got improved estimates of m_E, -2.157480 and -2.389980. These estimates were used from the iteration step 8 and 170, respectively, as shown in the first entry of Table 5-1[50]. These estimates are much smaller than the initial estimate, -0.1, but still considerably bigger than the calculated lower bound, $-\|G\|_\infty = -3.0$, which means that the lower bound is a rather conservative estimate.

We can examine the system matrix to get some clues for the possible interpretation of these two values. Most of nodes are the interior nodes. In general, as the depth constraints get sparser,

[50]This algorithm was run again on the images with sparser depth constraints. Same initial estimates, 0.0 and -0.1 were used for M_E and m_E, respectively. When the density of the depth constraints was 30%, the improved estimates of m_E, -2.180764 and -2.402566, were used from the iteration step 8 and 399, respectively. When the density was 15%, the improved estimate of m_E, -2.193811, was used from the iteration step 8.

more of these interior nodes are not depth constrained. The interior node with no depth constraint is illustrated in Figure 2-4 and its corresponding nodal equation is given in the equation (1). The definition of the matrix norm $||G||_\infty$ given in the equation (44) is reproduced here,

$$||G||_\infty \equiv \max_{i=1,2,\ldots,n} \left(\sum_{j=1}^{n} |g_{i,j}|\right).$$

When we compute the quantity $-\sum_{j=1}^{n} |g_{i,j}|$ for the interior node, we get $-44/20 = -2.2$, which is much closer to the final estimates of m_E we obtained above. Furthermore, when we compute the quantity $-\sum_{j=1}^{n} |g_{i,j}|$ for the boundary node whose corresponding nodal equation is given in the equation (4), we get $-12/4 = -3.0$, which coincides with the lower bound[51].

Therefore, -2.2 can serve as a more accurate initial estimate of m_E. The second entry in Table 5-1 shows the result where 0.0 is used as initial estimate of M_E and -2.2 for m_E. In Table 5-2, we show the number of iterations to attain the specified fraction of the initial RMSE value. The comparison of the first two columns confirms that -2.2 is a better initial estimate than -3.0. The difference in number of iterations is small in the beginning but becomes increasingly larger as the RMSE is reduced further.

For the more accurate initial estimate of M_E, we can pick a number that is slightly smaller than the final estimate at convergence. For this example, we picked 0.99. As the fourth and fifth entry in Table 5-1 shows, the initial estimate of M_E is quite good so that it is not changed for more than 100 initial iteration steps. In the adaptive Chebyshev acceleration method, the more accurate estimation of M_E is more critical than that of m_E for convergence. The remaining two columns in Table 5-2 show that effect. The RMSE values in the beginning rows are attained in much smaller number of iterations for initial M_E of 0.99 compared to that of 0.0. The absolute differences in number of iteration steps are maintained or increased as the iterations go on, though in a less degree between the second and the third column. But the overall relative ratios of iteration steps get smaller. This is due to the fact that nearly the same final estimate of M_E is obtained at the end of iterations even with different initial choice of estimates.

[51] For depth constrained nodes, we get bigger values. When the interior node is depth constrained, the nodal equation is given in the equation (2) and we get $-44/20.5 = -2.1463$ assuming .5 for γ. When the corner boundary node is depth constrained, we get $-12/4.5 = -2.6667$ assuming the same γ value. If the γ value is increased to 2.0, we get $-44/22.0 = -2.0$ for the interior node and $-12/6.0 = -2.0$ for the corner boundary node.

We have done the same experiment using the similar set of the estimates for the first Young algorithm. The only difference is that we used -2.3 instead of -2.2 for the initial estimate of m_{E}. The results are reported in Table 5-3 and 5-4. The overall performance is very similar to that in Table 5-1 and 5-2. Also, the choice of -2.3 as an initial estimate of m_{E} is not sensitive. When 0.0 was used as initial estimate of M_{E}, it took 192, 195, and 197 iterations to reduce the RMSE to .00001 for initial estimate m_{E} values of -2.2, -2.3, and -2.4, respectively. When 0.99 was used as initial estimate of M_{E}, the same phenomenon was observed, i.e., it took 167, 170, and 172 iterations, respectively. Note also that for initial estimate m_{E} of -2.2, it took same number of iteration steps for both Young algorithms, i.e., 192 and 167 iterations to reduce the RMSE to .00001 for initial estimate M_{E} of 0 and 0.99, respectively.

As discussed in section 4.1.3, the average virtual rate of convergence for the Chebyshev acceleration method increases to an asymptotic value and many iterations are often required before the asymptotic convergence is achieved. Thus, whenever there is a change of estimate of M_{E}, a new Chebyshev polynomial is generated with the better estimate but the convergence slows down for a while. For instance, compare the second and the fourth column of Table 5-4 where the same estimate of $m_{\mathrm{E}} = -2.3$ is used. When initial M_{E} was 0.0, i.e. for the second column, a new estimate of M_{E} is used from the iteration step of 48, as reported in Table 5-3. To reduce the fraction of the RMSE from .2 to .1, it takes 12 iterations in the second column while it takes only 7 iterations in the fourth column. The similar phenomenon can be observed again for two columns. As another example, in the fourth column, a new estimate of M_{E} is used from the iteration step of 110. To reduce the fraction of the RMSE from .0005 to .0002, it takes 16 iterations in the fourth column while it takes only 11 iterations in the second column.

Thus, it is clear from this discussion that using initial $m_{\mathrm{E}} = -2.3$ determined from the analysis of the quantity $-\sum_{j=1}^{n} |g_{ij}|$ for the interior node is good enough; and therefore we will use Young's first algorithm throughout this chapter.

i	m_E	M_E
1 – 7	−0.1	.0
8 – 14	−2.157480	.1
15 – 21	−2.157480	.386872
22 – 36	−2.157480	.984627
37 – 63	−2.157480	.987545
64 – 96	−2.157480	.989491
97 – 137	−2.157480	.990982
138 – 150	−2.157480	.992213
151 – 169	−2.157480	.996393
170 – 176	−2.389980	.1
177 – 183	−2.389980	.297588
184 –	−2.389980	.992941

i	m_E	M_E
1 – 7	−2.2	.0
8 – 14	−2.2	.899890
15 – 23	−2.2	.982372
24 – 48	−2.2	.986745
49 – 79	−2.2	.988929
80 – 119	−2.2	.990533
120 – 183	−2.2	.991806
184 –	−2.2	.992705

(Continued)

i	m_E	M_E
1 - 7	-3.0	.0
8 - 14	-3.0	.891406
15 - 25	-3.0	.981490
26 - 52	-3.0	.986737
53 - 87	-3.0	.988922
88 - 130	-3.0	.990515
131 - 201	-3.0	.991795
202 -	-3.0	.992691

i	m_E	M_E
1 - 107	-2.2	.99
108 - 126	-2.2	.990863
127 -	-2.2	.992609

i	m_E	M_E
1 - 121	-3.0	.99
122 - 142	-3.0	.990878
143 -	-3.0	.992618

Table 5-1: Change of Estimates of Extreme Eigenvalues (Algorithm 6-5.1)

RMSE	$m_E = -3.0$, $M_E = 0.0$	$m_E = -2.2$, $M_E = 0.0$	$m_E = -3.0$, $M_E = 0.99$	$m_E = -2.2$, $M_E = 0.99$
0.5	28	26	12	11
0.2	42	39	21	19
0.1	55	48	29	26
0.05	67	61	38	34
0.02	80	72	51	46
0.01	95	86	63	56
0.005	106	96	75	67
0.002	119	108	93	83
0.001	134	120	107	95
0.0005	147	132	121	110
0.0002	160	144	141	126
0.0001	170	153	157	140
0.00005	181	163	166	148
0.00002	197	177	178	159
0.00001	214	192	187	167

Table 5-2: Effects of Errors in Initial Estimates of Extreme Eigenvalues (Algorithm 6-5.1)

i	m_E	M_E
1 – 7	−2.3	.0
8 – 14	−2.3	.898760
15 – 23	−2.3	.982258
24 – 47	−2.3	.986702
48 – 79	−2.3	.988855
80 – 117	−2.3	.990471
118 – 180	−2.3	.991757
181 –	−2.3	.992662

i	m_E	M_E
1 – 7	−3.0	.0
8 – 14	−3.0	.891406
15 – 24	−3.0	.981490
25 – 49	−3.0	.986582
50 – 84	−3.0	.988772
85 – 125	−3.0	.990385
126 – 192	−3.0	.991694
193 –	−3.0	.992620

(Continued)

i	m_E	M_E
1 – 109	−2.3	.99
110 – 128	−2.3	.990867
129 –	−2.3	.992610

i	m_E	M_E
1 – 120	−3.0	.99
121 – 141	−3.0	.990868
142 –	−3.0	.992611

Table 5-3: Change of Estimates of Extreme Eigenvalues (Algorithm 6-4.1)

RMSE	$m_E = -3.0$, $M_E = 0.0$	$m_E = -2.3$, $M_E = 0.0$	$m_E = -3.0$, $M_E = 0.99$	$m_E = -2.3$, $M_E = 0.99$
0.5	28	27	12	11
0.2	42	39	21	19
0.1	56	51	29	26
0.05	66	61	38	34
0.02	80	73	51	47
0.01	95	87	63	57
0.005	105	97	75	69
0.002	119	109	93	84
0.001	136	124	107	97
0.0005	146	134	123	112
0.0002	159	145	141	128
0.0001	170	155	156	142
0.00005	181	165	165	150
0.00002	202	180	177	161
0.00001	212	195	187	170

Table 5-4: Effects of Errors in Initial Estimates of Extreme Eigenvalues (Algorithm 6-4.1)

5.3 Experiments on Synthetic Images

5.3.1 Single-grid Algorithms on the Finest Grid

5.3.1.1 Experiments on a Floating Plane

The synthetic image used was a floating plane of constant depth, $\alpha = (1.0\ 1.0\ \ldots\ 1.0)^T$. The densities of the depth constraints were varied to 15%, 30%, and 50%.

In Table 5-5, we show the number of iterations i to attain the specified fraction of the initial RMSE value.[52] The results are tabulated side by side for three different iterative methods, the conjugate gradient, the adaptive Chebyshev acceleration, and the Gauss-Seidel method.

We observe that the conjugate gradient method performs best in the sense that it takes the least number of iterations. The adaptive Chebyshev acceleration method comes next and the Gauss-Seidel method performs worst. But we should note that each step of the iteration of the first two methods is completely parallelized so that overall execution is much faster, compared to the Gauss-Seidel method where the computation is done in serial fashion. As the depth constraints become sparser, the discrete depth interpolation problem itself becomes inherently harder to solve and takes more iterations. Even here, the degradation in the Gauss-Seidel method turns out to be the worst.

Suppose that the size of the mesh at the leaf of the tree is $s \times s$. When there are depth constraints only, we have derived before that the total number of machine cycles per iterations are $4392 \times (\log_2 s) + 16453$ for the adaptive Chebyshev acceleration method (see section 4.1.5.4) and $5856 \times (\log_2 s) + 32307$ for the conjugate gradient method (see section 4.2.3.4), respectively. For the tree with 128×128 mesh, the number of machine cycles are 47197 for the adaptive Chebyshev acceleration method and 73299 for the conjugate gradient method. In Table 5-6, we show the normalized number of iterations for two methods in the first two columns. For the adaptive Chebyshev acceleration method, we use the same numbers as in Table 5-5. For the conjugate gradient method, we have multiplied the number of iterations in Table 5-5 by 73299 /

[52]The trivial initial approximation, $x^{(0)} = (0.0\ 0.0\ \ldots\ 0.0)^T$, was used. Thus, the initial RMSE was 1.0 for this example.

47197 = 1.5530.

After normalization, the conjugate gradient method still performs better than the adaptive Chebyshev acceleration method. This is in part due to the errors in the initial estimates of the eigenvalues. The initial estimates of the largest and the smallest eigenvalues, M_E and m_E, were 0.0 and $-\|G\|_\infty = -3.0$, respectively. In the first few iterations, the conjugate gradient method performs much better, but as more computations are done the estimates of the eigenvalues (in this case, M_E only) get better. The final estimates of M_E values at the iteration steps of 170, 271, and 495 were .991694, .996805, and .999064 when the densities of the depth constraints were varied to 50%, 30%, and 15%, respectively. [See Table I-1.] Thus, the ratio (or relative difference) of the number of iterations between the conjugate gradient and the adaptive Chebyshev acceleration methods gets smaller.

The third column in Table 5-6 shows the result from the adaptive Chebyshev acceleration method with more accurate initial estimates. For the initial estimates of M_E, we used the values slightly smaller than final estimates from previous runs: we used .99, .993, and .997 as the initial estimates of M_E. For the initial estimates of m_E, we used -2.3. With these near optimal initial estimates, the adaptive Chebyshev acceleration method performed better than or at least comparable to the conjugate gradient method, i.e., the normalized number of iterations were smaller nearly everywhere. It is not surprising since the iteration process is started with global information: here, the near optimal estimates of the largest and the smallest eigenvalues. In contrast, the conjugate gradient method is started with no global information, at least in the beginning. However, for several iteration steps in the first row in Table 5-6, we have a paradoxical result in the sense that the conjugate gradient method turns out to be faster. This can be explained by the fact that the convergence rate is slow in the beginning even for the optimal Chebyshev acceleration method, as mentioned in section 4.1.3.

Other numerical values are given in section I.3.1 of Appendix with further explanations.

RMSE	Conjugate Gradient			Chebyshev Accel.			Gauss-Seidel		
	50%	30%	15%	50%	30%	15%	50%	30%	15%
0.5	6	9	16	28	36	55	30	52	112
0.2	13	21	37	42	60	99	73	131	296
0.1	21	32	55	56	80	129	107	196	460
0.05	28	43	74	66	94	158	142	265	648
0.02	38	58	104	80	121	207	191	364	950
0.01	46	70	130	95	137	251	229	445	1231
0.005	53	82	153	105	161	297	269	531	1555
0.002	63	99	179	119	183	354	323	654	2024
0.001	71	110	200	136	208	384	365	752	2394
0.0005	79	122	220	146	224	416	408	855	2769
0.0002	89	136	247	159	247	470	465	995	3269
0.0001	96	146	267	170	271	495	509	1103	3650

Table 5-5: Number of Iterations (plane)

RMSE	Conjugate Gradient			Chebyshev Accel. (*initial* $m_E = -3.0$, $M_E = 0.0$)			Chebyshev Accel. (*initial* $m_E = -2.3$, $M_E \geq 0.99$)		
	50%	30%	15%	50%	30%	15%	50%	30%	15%
0.5	9.3	14.0	24.8	28	36	55	11	15	21
0.2	20.2	32.6	57.5	42	60	99	19	30	45
0.1	32.6	49.7	85.4	56	80	129	26	45	70
0.05	43.5	66.8	114.9	66	94	158	34	62	102
0.02	59.0	90.1	161.5	80	121	207	47	93	165
0.01	71.4	108.7	201.9	95	137	251	57	110	197
0.005	82.3	127.3	237.6	105	161	297	69	124	240
0.002	97.8	153.8	278.0	119	183	354	84	152	289
0.001	110.3	170.8	310.6	136	208	384	97	169	315
0.0005	122.7	189.5	341.7	146	224	416	112	184	344
0.0002	138.2	211.2	383.6	159	247	470	128	211	395
0.0001	149.1	226.7	414.7	170	271	495	142	224	422

Table 5-6: Normalized Number of Iterations (plane)

5.3.1.2 Experiments on a Cylinder

We show the number of iterations in Table 5-7 and the normalized values in Table 5-8. We have similar results, but the overall number of iterations are smaller, because of the parameter change in the nodal equations. For the plane example, we employed smaller value for γ, 0.5, but for the cylinder and the sphere example, we employed 2.0. This has the effect of giving the center nodes and the depth constraints more weight, and thus speeds up the convergence. [See the equation (2).]

For the adaptive Chebyshev acceleration method run with more accurate initial estimates, we proceeded in similar fashion. For the initial estimates of m_E, we used -2.3. When the initial estimates of m_E and M_E were $-\|G\|_\infty = -3.0$ and 0.0, we obtained .982331, .993159, and .998364 as the final estimates of M_E values at the iteration steps of 117, 191, and 380 when the densities of the depth constraints were varied to 50%, 30%, and 15%, respectively. As the improved initial estimates of M_E, we used .97, .98, and .99, respectively.

5.3.1.3 Experiments on a Sphere

We show the number of iterations in Table 5-9 and the normalized values in Table 5-10. We have similar results to those of the cylinder, but the overall number of iterations are a little bit smaller. The major difference here is that only limited accuracy could be obtained even after sufficient number of iterations for all iterative methods.[53] For example, even after 100 iterations were carried out with the conjugate gradient method over the sphere image of the 50% depth constraints, the fraction of the RMSE error was still .0063. In Table 5-9, we observe that the bulk of the accuracy, the fraction .01, was obtained after only 29 iterations. This phenomena was also observable in the cylinder examples, though in a much less degree.

For the adaptive Chebyshev acceleration method run with more accurate initial estimates, we proceeded in similar fashion. For the initial estimates of m_E, we used -2.3. When the initial estimates of m_E and M_E were $-\|G\|_\infty = -3.0$ and 0.0, we obtained .987428, .991676, and

[53]We can explain the limited accuracy as following. Starting with the plane, and then to the cylinder and the sphere examples, the depth values progressively change more rapidly at the boundary. We observe also that the nodes at the boundary are connected with a fewer number of neighboring nodes, i.e. they are less supported, because of the presence of the depth discontinuity. In fact, we deal with a free plate problem. [Recall the discussion at the end of section 2.2.2.3. For instance, compare the equation (1) with the equation (4).]

RMSE	Conjugate Gradient			Chebyshev Accel.			Gauss-Seidel		
	50%	30%	15%	50%	30%	15%	50%	30%	15%
0.5	4	7	12	17	23	36	8	15	36
0.2	9	15	28	27	39	67	21	42	109
0.1	13	22	42	33	52	94	33	66	187
0.05	18	29	56	42	66	123	45	94	288
0.02	24	40	82	53	85	164	62	138	470
0.01	29	49	100	60	102	203	77	178	647
0.005	34	59	116	70	114	232	92	222	843
0.002	41	70	135	79	136	273	114	287	1114
0.001	46	77	152	90	153	295	131	339	1323
0.0005	51	85	170	97	165	318	149	393	1532
0.0002	58	95	189	106	179	360	173	466	1803
0.0001	62	103	202	117	191	380	192	521	2000

Table 5-7: Number of Iterations (cylinder)

RMSE	Conjugate Gradient			Chebyshev Accel. (*initial* $m_E = -3.0$, $M_E = 0.0$)			Chebyshev Accel. (*initial* $m_E = -2.3$, $M_E \geq 0.97$)		
	50%	30%	15%	50%	30%	15%	50%	30%	15%
0.5	6.2	10.9	18.6	17	23	36	6	8	12
0.2	14.0	23.3	43.5	27	39	67	10	17	32
0.1	20.2	34.2	65.2	33	52	94	15	26	58
0.05	28.0	45.0	87.0	42	66	123	20	39	91
0.02	37.3	62.1	127.3	53	85	164	28	64	128
0.01	45.0	76.1	155.3	60	102	203	36	75	159
0.005	52.8	91.6	180.2	70	114	232	45	91	190
0.002	63.7	108.7	209.7	79	136	273	58	111	228
0.001	71.4	119.6	236.1	90	153	295	69	121	266
0.0005	79.2	132.0	264.0	97	165	318	75	131	294
0.0002	90.1	147.5	293.5	106	179	360	83	146	322
0.0001	96.3	160.0	313.7	117	191	380	89	160	344

Table 5-8: Normalized Number of Iterations (cylinder)

.997106 as the final estimates of M_E values at the iteration steps of 113, 190, and 383 when the densities of the depth constraints were varied to 50%, 30%, and 15%, respectively. As the improved initial estimates of M_E, we used .97, .98, and .99, respectively.

Up to this point, we have dealt with the images of the depth constraints only. For the sphere image, we have another result with both the depth and the orientation constraints. We assume the identical depth constraints but we assume also that the orientation constraints are available at the same nodes.[55] For the coefficients of the constraints terms in the nodal equations, we used $\beta^h = 2.0 / h^2$ for the depth constraints and $\alpha^h = 1.0 / h$ for the orientation constraints. [See the equations (2) and (3).]

We show the number of iterations in Table 5-11 and the normalized values in Table 5-12. The densities of the constraints are varied to 30%, 15%, 5%, and 2%. When we compare the results of the iterative methods on depth constraints only with the methods on both depth and orientation constraints for same densities (30% and 15%), we observe the improvements. The number of iteration steps to reach the .2, .1, and .05 of the initial RMSE values are smaller for all iterative methods when both constraints exist. However, the improvements are surprisingly marginal.

When both constraints exist, we need to use a slightly different normalizing factor. We have derived before that the total number of machine cycles per iterations are $4392 \times (log_2 s) + 17712$ for the adaptive Chebyshev acceleration method (see section 4.1.5.4) and $5856 \times (log_2 s) + 34825$ for the conjugate gradient method (see section 4.2.3.4), respectively, when the size of the mesh at the leaf of the tree is $s \times s$. For the tree with 128×128 mesh, the number of machine cycles are 48456 for the adaptive Chebyshev acceleration method and 75817 for the conjugate gradient method. For the adaptive Chebyshev acceleration method, we use the same numbers as in Table 5-11. For the conjugate gradient method, we have multiplied the number of iterations by $75817 / 48456 = 1.5647$ in Table 5-12.

[55] The p and q orientation constraints are defined by $p \equiv \frac{\partial u}{\partial i}$ and $q \equiv \frac{\partial u}{\partial j}$ where $u_{[i,j]}$ is the (synthetic) depth value at the node $[i, j]$. For the specific image considered, we have $u_{[i,j]} = (1.0 - ((i-r)^2 + (j-r)^2) / (r)^2)^{1/2}$. We can show that $p_{[i,j]} = (r-i) / (u \times r^2)$. Similarly, we have $q_{[i,j]} = (r-j) / (u \times r^2)$.

RMSE	Conjugate Gradient			Chebyshev Accel.			Gauss-Seidel		
	50%	30%	15%	50%	30%	15%	50%	30%	15%
0.5	4	6	11	17	23	36	8	15	35
0.2	9	14	26	27	38	65	21	41	103
0.1	13	20	39	33	50	88	32	63	172
0.05	17	27	53	41	61	113	43	88	259
0.02	23	36	71	49	76	147	60	126	406
0.01	29	47	***[54]	60	95	***	77	172	***

Table 5-9: Number of Iterations (sphere)

RMSE	Conjugate Gradient			Chebyshev Accel. (*initial* $m_E = -3.0$, $M_E = 0.0$)			(*initial* $m_E = -2.3$, $M_E \geq 0.97$)		
	50%	30%	15%	50%	30%	15%	50%	30%	15%
0.5	6.2	9.3	17.1	17	23	36	6	8	12
0.2	14.0	21.7	40.4	27	38	65	10	16	30
0.1	20.2	31.1	60.6	33	50	88	14	24	53
0.05	26.4	41.9	82.3	41	61	113	19	35	86
0.02	35.7	55.9	110.3	49	76	147	26	52	111
0.01	45.0	73.0	*****	60	95	***	34	73	***

Table 5-10: Normalized Number of Iterations (sphere)

[54]"***" refers to a number that could not be computed even if sufficient number of iterations were performed.

For the adaptive Chebyshev acceleration method, we used -2.3 for the more accurate initial estimates of m_E. When the initial estimates of m_E and M_E were -3.0 and 0.0, we obtained .986494, .995370, .999063, and .999835 as the estimates of M_E values at the iteration steps of 96, 123, 244, and 591 when the densities of the depth constraints were varied to 30%, 15%, 5%, and 2%, respectively. As the more accurate initial estimates of M_E, we used .98, .99, .998, and .9998, respectively.

5.3.2 Multigrid Algorithms

5.3.2.1 Experiments on a Cylinder

One of the two synthetic images we tried is a portion of a cylinder, the same one used in section 5.3.1.2. However, the depth constraints are not scattered randomly. The constraints lie along j = 8, 36, 64, 93, and 119 where $0 \le j \le 127$.[56]

We ran four-level multigrid algorithms to see how much speed up is achieved against iteration on the finest level only. The sizes of the images are reduced from 128×128 to 64×64, 32×32, and 16×16 as seen from the finest level ($l = 4$) to the coarsest one ($l = 1$). The depth constraints at the coarser levels are generated by sampling the same cylindrical surface. For this particular example, the depth constraints are constrained along one arc of the cylinder only so that the densities of the depth constraints are doubled for each coarser level. At the finest level, we have the density of 3.91%. For the coarser levels, we have the densities of 7.81%, 15.63%, and 31.25%, respectively.

Before running the multigrid algorithms, we obtained the result separately for each level. [See the equation (61)]. We show the number of iterations in Table 5-13.

The trivial initial approximations, $x^{(0)} = (0.0\ 0.0\ \ldots\ 0.0)^T$, were used for all levels. The initial RMSE values were .956515, .956709, .956742, and .956742 for the level l = 1, 2, 3, and 4, respectively.

In the adaptive Chebyshev acceleration method, we show the result with more accurate initial

[56]This example is similar to the one used in [Terz 84, p. 125].

RMSE	Conjugate Gradient				Chebyshev Accel.				Gauss-Seidel			
	30%	15%	5%	2%	30%	15%	5%	2%	30%	15%	5%	2%
0.5	6	11	29	69	23	35	76	168	15	35	152	661
0.2	13	24	66	166	38	61	154	380	39	98	519	2756
0.1	19	37	95	235	46	85	220	545	60	161	914	5221
0.05	25	50	126	356	58	109	280	757	83	239	1419	9145
0.02	35	72	***	***	76	148	***	***	123	409	****	****

Table 5-11: Number of Iterations (sphere with depth and orientation constraints)

RMSE	Conjugate Gradient				Chebyshev Accel.							
					(*initial* $m_E = -3.0$, $M_E = 0.0$)				(*initial* $m_E = -2.3$, $M_E \geq 0.98$)			
	30%	15%	5%	2%	30%	15%	5%	2%	30%	15%	5%	2%
0.5	9.4	17.2	45.4	108.0	23	35	76	168	8	12	26	60
0.2	20.3	37.6	103.3	259.7	38	61	154	380	15	28	70	146
0.1	29.7	57.9	148.6	367.7	46	85	220	545	22	48	129	222
0.05	39.1	78.2	197.1	557.0	58	109	280	757	31	77	213	399
0.02	54.8	112.7	*****	*****	76	148	***	***	48	112	***	***

Table 5-12: Normalized Number of Iterations (sphere with depth and orientation constraints)

estimates only. For the initial estimates of m_E, we used -2.3. As the improved initial estimates of M_E, we used .97, .99, and .999, for the level $l = 1, 2$, and 3, respectively.[57] For the finest level, we used .99995 as the estimate of M_E from the beginning to the end without any change at all.[58] In the optimal Chebyshev acceleration and the weighted Jacobi methods, for the real eigenvalues, $m(G^{h_l})$ and $M(G^{h_l})$, we simply used these initial estimates instead, assuming that these estimates are fairly good.

When the depth constraints exist only, we have derived in section 4.3.3.3 that at level l it takes $5856 \times ((log_2 s) - (L - l)) + 32307$ machine cycles for an iteration of the conjugate gradient method.[59] Thus, an iteration on each level from coarsest to finest respectively takes 55731, 61587, 67443, and 73299 machine cycles. We have derived also that at level l it takes $4392 \times ((log_2 s) - (L - l)) + 16453$ machine cycles for an iteration of the adaptive Chebyshev acceleration method. Thus, an iteration on each level from coarsest to finest respectively takes 34021, 38413, 42805, and 47197 machine cycles. For an iteration of the optimal Chebyshev acceleration and the weighted Jacobi methods, we have derived that at all levels it takes 15314 and 13840 machine cycles, respectively.

We can now normalize the number of iterations to compare the iterative methods. In Table 5-14, we show the normalized number of iterations. For the optimal Chebyshev acceleration method, we use the same numbers as in Table 5-13. The number of iterations for other iterative methods were multiplied by corresponding normalizing factors.

We compare first the unnormalized number of iterations of the adaptive and optimal Chebyshev

---

[57] When the initial estimates of m_E and M_E were -3.0 and 0.0, we obtained .979848, .997912, and .999869 as the estimates of M_E values at the iteration steps of 88, 279, and 1205 for the level $l = 1, 2$, and 3, respectively.

[58] In general, for a larger image with sparser constraints, the depth interpolation problem is harder to solve. In the Chebyshev acceleration method, the largest eigenvalue $M(G)$ is closer to upper bound, 1.0. [Recall our discussion in section 4.3.1.] For this large image with very sparse and highly artificial distribution of depth constraints, the adaptive Chebyshev acceleration procedure did not work out well since $M(G)$ was too close to 1.0. Note that .99995 is the largest value of the imposed upper bounds for initial estimate of M_E. [For the numerical values of the upper bounds, see the equation (72) in Appendix.] We need to extend the upper bounds of the equation (72) suggested by Young. He states that the upper bounds were imposed infrequently. In his numerical experiments, the typical size of the matrix was 100×100 or 1000×1000. But in our case, we are dealing with the depth continuous region of the size 128×128, so that the size of the matrix is 16384×16384. Further study is called for with regard to extending the upper bounds.

[59] The size of the mesh at the finest level L is $s \times s$.

acceleration methods. The used initial estimates of the extreme eigenvalues are quite good. They are not updated, i.e., same for both methods, until the RMSEs are reduced to .1 of initial values. At the coarsest level, the number of iterations are nearly same even until the RMSE is reduced to .01. We compare now the normalized number of iterations of the adaptive Chebyshev acceleration and the conjugate gradient methods. For all levels, nearly all numbers in the first three rows, i.e., until the RMSEs are reduced to .1, show that the adaptive Chebyshev acceleration method executes faster.

When the RMSEs are reduced to .1, the normalized numbers are 2575.1, 2490.2, and 808 for the conjugate gradient, the adaptive, and the optimal Chebyshev acceleration methods, respectively, for the finest level. The normalized numbers are 54.6, 51.1, and 23 for the coarsest level. When the RMSEs are reduced to .01, the normalized numbers are 6255.8, 11655.9, and 3782 for the finest level and 109.2, 117.7, and 55 for the coarsest level.

We discuss now the multigrid execution results. For the multilevel coordination, the fixed scheduling scheme was used. [For the listing, refer back to Figure 4-2]. At the coarsest level, procedure SOLVE performs iterations to desired accuracy. The fixed number of iterations performed by SOLVE are designated as t_0 and s_0, when invoked inside of procedure FMRA and FMC, respectively.

For the conjugate gradient method, four sets of parameters were used to obtain different precision of final RMSE values. We compare the amount of time taken to arrive at the same final RMSE values at the finest level, when iterations are carried out on the finest level only versus multigrid.

For the first set, $t_0 = 15$ was used. In Table 5-13, we can read that the RMSE value is reduced to .1 of initial value at the coarsest level. The actual value obtained was .101114. For the other parameters, $s_0 = 4$, $n_1 = 6$, $n_2 = 1$, and $n_3 = 6$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 27, 36, 24, and 12. The final RMSE value obtained on each level from coarsest to finest respectively is .0664448, .0675243, .0680527, and .0683314. When the relaxation is carried out on the finest level only, the RMSE value .0682560 is obtained after 683 iterations.

RMSE	Conjugate Gradient				Adapt. Chebyshev Accel.				Opt. Chebyshev Accel.			
	l=4	*l*=3	*l*=2	*l*=1	*l*=4	*l*=3	*l*=2	*l*=1	*l*=4	*l*=3	*l*=2	*l*=1
0.5	167	44	13	4	164	42	15	7	164	42	15	7
0.2	419	100	27	10	430	127	37	15	430	127	37	15
0.1	538	130	35	15	808	245	67	23	808	245	64	23
0.05	894	216	55	20	1470	400	94	32	1470	432	101	32
0.02	1088	268	78	25	2748	569	141	47	2748	771	171	45
0.01	1307	323	91	30	3782	677	170	53	3782	1045	231	55
0.005	1575	393	107	35	4850	748	189	59	4850	1330	293	66
0.002	1788	472	121	44	6308	846	214	68	6308	1721	379	85
0.001	2083	533	137	**	7459	935	237	**	7459	2038	455	**
0.0005	2235	585	***	**	8743	1065	***	**	8743	2419	***	**

(Continued)

RMSE	Gauss-Seidel				Weighted Jacobi			
	l=4	*l*=3	*l*=2	*l*=1	*l*=4	*l*=3	*l*=2	*l*=1
0.5	5422	375	44	12	####	####	144	40
0.2	####[60]	1280	123	31	####	####	403	102
0.1	####	2365	204	47	####	####	665	155
0.05	####	4033	313	65	####	####	1011	213
0.02	####	7073	512	89	####	####	1652	295
0.01	####	9572	686	108	####	####	2223	358
0.005	####	####	866	128	####	####	2817	426
0.002	####	####	1114	165	####	####	3635	550
0.001	####	####	1333	***	####	####	####	***
0.0005	####	####	****	***	####	####	****	***

Table 5-13: Number of Iterations on Fine/Coarse Levels (cylinder)

[60] "####" refers to a number that was not computed. It could be computed, but it would just show that some iterative methods are indeed very slow.

RMSE	Conjugate Gradient				Adapt. Chebyshev Accel.			
	l=4	*l*=3	*l*=2	*l*=1	*l*=4	*l*=3	*l*=2	*l*=1
0.5	799.3	193.8	52.3	14.6	505.4	117.4	37.6	15.6
0.2	2005.5	440.4	108.6	36.4	1325.2	355.0	92.8	33.3
0.1	2575.1	572.5	140.8	54.6	2490.2	684.8	168.1	51.1
0.05	4279.0	951.3	221.2	72.8	4530.5	1118.1	235.8	71.1
0.02	5207.6	1180.3	313.7	91.0	8469.2	1590.4	353.7	104.4
0.01	6255.8	1422.5	366.0	109.2	11655.9	1892.3	426.4	117.7
0.005	7538.6	1730.8	430.3	127.4	14947.5	2090.8	474.1	131.1
0.002	8558.1	2078.7	486.6	160.1	19440.9	2364.7	536.8	151.1
0.001	9970.1	2347.3	551.0	*****	22988.3	2613.5	594.5	*****
0.0005	10697.6	2576.4	*****	*****	26945.5	2976.8	*****	*****

(Continued)

RMSE	Opt. Chebyshev Accel.				Weighted Jacobi			
	l=4	l=3	l=2	l=1	l=4	l=3	l=2	l=1
0.5	164	42	15	7	######	######	130.1	36.1
0.2	430	127	37	15	######	######	364.2	92.2
0.1	808	245	64	23	######	######	601.0	140.1
0.05	1470	432	101	32	######	######	913.7	192.5
0.02	2748	771	171	45	######	######	1493.0	266.6
0.01	3782	1045	231	55	######	######	2009.0	323.5
0.005	4850	1330	293	66	######	######	2545.9	385.0
0.002	6308	1721	379	85	######	######	3285.1	497.1
0.001	7459	2038	455	**	######	######	######	*****
0.0005	8743	2419	***	**	######	######	******	*****

Table 5-14: Normalized Number of Iterations on Fine/Coarse Levels (cylinder)

The multigrid algorithm takes 6220089 machine cycles for iterations on four levels. To this, we should add the time taken by 3 prolongation operations invoked in procedure FMRA, 6 restriction operations in FMC, and 6 prolongation operations in FMC. We have derived in section 4.3.3.4 that it takes 52383, 27394, and 53847 machine cycles for a prolongation operation in FMRA, a restriction operation in FMC, and a prolongation operation in FMC, respectively, when the depth constraints exist only. Therefore, 644595 machine cycles are added to yield a total of 6864684. For the iterations on the finest level only, it takes 50063217 machine cycles. When the numbers of machine cycles are divided, we get a speed-up factor of 50063217 / 6864684 = 7.2929.[61]

When we compare the number of iterations to attain .1 of initial RMSEs on the coarsest and finest levels, translated into total number of machine cycles, we get a speed-up factor of 39434862 / 835965 = 47.1729. In contrast, the speed-up factor of multigrid execution result (7.2929) is much smaller than the speed-up factor obtained (47.1729) which can serve as a rough upper limit, though a too high one. This is due to two reasons. First, the iterations are performed on all levels. Though we carry out 27 iterations on the coarsest level, including t_0 iterations to reach an initial accuracy, we have 36, 24, and 12 iterations on other three fine levels. Second, interlevel computations are carried out, too. For this case, it amounts to 644595 / 6864684 = 9.39 percent of the entire execution time.

For the second set, $t_0 = 30$ was used. The RMSE value .00938218 was obtained at the coarsest level after 30 iterations. For the other parameters, $s_0 = 4$, $n_1 = 6$, $n_2 = 1$, and $n_3 = 6$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 42, 36, 24, and 12. The final RMSE value obtained on each level from coarsest to finest respectively is .00704917, .00727274, .00749481, and .00758961. When the relaxation is carried out on the finest level only, the RMSE value .00760190 is obtained after 1467 iterations. After carrying out a similar analysis, we get a speed-up factor of 107529633 / 7700649 = 13.9637.

For the third set, $t_0 = 44$ was used. The RMSE value .00194568 was obtained at the coarsest level

[61]When smaller parameter values are used, more speed-up is possible. For example, with $s_0 = 3$, $n_1 = 3$, $n_2 = 1$, and $n_3 = 3$, the final RMSE value obtained on each level from coarsest to finest respectively is .0793532, .0838957, .0858741, and .0867920. When the relaxation is carried out on the finest level only, the RMSE value .0868207 is obtained after 558 iterations. Here, we get a speed-up factor of 40900842 / 4339815 = 9.4246.

after 44 iterations. For the other parameters, $s_0 = 4$, $n_1 = 8$, $n_2 = 1$, and $n_3 = 8$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 56, 48, 32, and 16. The final RMSE value obtained on each level from coarsest to finest respectively is .00154726, .00181073, .00202964, and .00213553. When the relaxation is carried out on the finest level only, the RMSE value .00213782 is obtained after 1735 iterations. After carrying out a similar analysis, we get a speed-up factor of 127173765 / 10052667 = 12.6507.

For the last set, $t_0 = 44$ was used again. For the other parameters, $s_0 = 4$ and $n_2 = 1$ were used as before. For n_1 and n_3, we used different values for each level. In Table 5-13, at the coarsest level the RMSE value is not reduced further after 44 iterations. Therefore, we increased the number of iterations at finer levels to get smaller final RMSE values. For $n_1 = n_3$, we used 12, 36, and 108 for the level $l = 2$, 3, and 4, respectively. The total number of iterations performed on each level from coarsest to finest respectively is 56, 72, 144, and 216. The final RMSE value obtained on each level from coarsest to finest respectively is .000496926, .000519746, .000551625, and .000427701. When the relaxation is carried out on the finest level only, the RMSE value .000428212 is obtained after 2269 iterations. Here, we get a speed-up factor of 166315431 / 33744171 = 4.9287.

Since the conjugate gradient and the Chebyshev acceleration methods are performing already well on the single-grid, the speed-ups are relatively small (14 or less). They are also insensitive to parameter setting, i.e., they vary continuously with different values of s_0, n_1, and n_3 for a given t_0.

For the adaptive Chebyshev acceleration method, two sets of parameters were used.

For the first set, $t_0 = 23$ was used. The RMSE value .0955011 was obtained at the coarsest level after 23 iterations. For the other parameters, $s_0 = 4$, $n_1 = 7$, $n_2 = 1$, and $n_3 = 7$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 35, 42, 28, and 14. The final RMSE value obtained on each level from coarsest to finest respectively is .0627607, .0624723, .0621976, and .0620595. When the relaxation is carried out on the finest level only, the RMSE value .0620297 is obtained after 1174 iterations. After carrying out a similar analysis, we get a speed-up factor of 55409278 / 5307974 = 10.4389.[62]

Compared with the similar result of the conjugate gradient method for the first set of parameters, the final RMSE values are reduced further with less machine cycles. This remark applies to the other examples as well.

For the second set, $t_0 = 53$ was used. The RMSE value .00987263 was obtained at the coarsest level after 53 iterations. For the other parameters, $s_0 = 6$, $n_1 = 8$, $n_2 = 1$, and $n_3 = 8$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 71, 48, 32, and 16. The final RMSE value obtained on each level from coarsest to finest respectively is .00647283, .00672038, .00701148, and .00712965. When the relaxation is carried out on the finest level only, the RMSE value .00712748 is obtained after 4232 iterations. Here, we get a speed-up factor of 199737704 / 7028822 = 28.4170.[63]

When the switching of level occurs, the most recent estimates of the extreme eigenvalues are preserved. As each level is entered, a new Chebyshev polynomial is generated based on the stored estimates. For the set of parameters employed, t_0 was relatively big but s_0, n_1, and n_3 were not big enough for the change of estimates to occur.[64] The change of estimates occurred at the coarsest level only during the initial iteration steps when the desired accuracy was obtained. The initial estimate of M_E was .97 at the coarsest level. With $t_0 = 53$, the improved estimates of .973413 and .981453 were obtained at the iteration steps of 35 and 43, respectively. After this update of the estimate of M_E at the coarsest level, no further estimate changes were observed.

---

[62]When smaller parameter values are used, more speed-up is possible. With $s_0 = 4$, $n_1 = 6$, $n_2 = 1$, and $n_3 = 6$, the final RMSE value obtained on each level from coarsest to finest respectively is .0642141, .0649569, .0648902, and .0648521. When the relaxation is carried out on the finest level only, the RMSE value .0648646 is obtained after 1129 iterations. Here, we get a speed-up factor of 53285413 / 4811882 = 11.0737.

[63]For this example, we varied n_1 and n_3 while $n_1 + n_3$ was kept same as before. With smaller n_1 value, $n_1 = 6$ and $n_3 = 10$, the final RMSE value obtained on the finest level is .00847359. When the relaxation is carried out on the finest level only, the RMSE value .00847447 is obtained after 3967 iterations. Here, we get a speed-up factor of 26.6375. Since $n_1 + n_3$ is same, it takes same amount of computation for the multigrid algorithm. With larger n_1 value, $n_1 = 10$ and $n_3 = 6$, we get smaller final RMSE value on the finest level, .00745601. However, note that both final RMSE values are larger than that of the original $n_1 = 8$ and $n_3 = 8$. When the relaxation is carried out on the finest level only, the RMSE value .00745516 is obtained after 4163 iterations. Here, we get a speed-up factor of 27.9536.

[64]For the discussion of p^*, the integer threshold that governs the generation of new Chebyshev polynomial and is greater than 5, see section I.1.1 of Appendix.

One more remark should be made about the estimates of the largest eigenvalue. In general, the size of the matrices is smaller and the constraints are denser at coarser levels. Both facts lead to easier matrix iteration problems to solve. In terms of the adaptive Chebyshev acceleration method, we have smaller numerical values for the largest eigenvalue at coarser levels. Therefore, at every switching to finer levels, the improved initial estimate may be set as the bigger value from following two choices: the estimate at the current level or the one at the adjacent coarse level.

For the optimal Chebyshev acceleration and the weighted Jacobi methods, two sets of parameters were used. Recall that in the optimal Chebyshev acceleration method, given (estimates of the) eigenvalues are used throughout the computation with no changes.

For the first set of parameters, where $t_0 = 23$, $s_0 = 4$, $n_1 = 7$, $n_2 = 1$, and $n_3 = 7$ were used, we have the identical results of final RMSE values since there were no changes of eigenvalue estimates in the adaptive Chebyshev acceleration method. But the total execution time is faster since the computation of the optimal Chebyshev acceleration method employs the local connections only. After carrying out an analysis, we get 17978636 and 2466961 machine cycles for the execution on the finest level only and the multigrid, respectively. Therefore, we get a speed-up factor of 17978636 / 2466961 = 7.2878.

For the second set, $t_0 = 55$ was used. The RMSE value .00967034 was obtained at the coarsest level after 55 iterations. For the other parameters, $s_0 = 6$, $n_1 = 8$, $n_2 = 1$, and $n_3 = 8$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 73, 48, 32, and 16. The final RMSE value obtained on each level from coarsest to finest respectively is .00638630, .00662617, .00691268, and .00702903. When the relaxation is carried out on the finest level only, the RMSE value .00703077 is obtained after 4253 iterations. Here, we get a speed-up factor of 65130442 / 3232661 = 20.1476.

For the Gauss-Seidel method, two sets of parameters were used.

For the first set, $t_0 = 47$ was used. The RMSE value .0967473 was obtained at the coarsest level after 47 iterations. For the other parameters, $s_0 = 6$, $n_1 = 8$, $n_2 = 1$, and $n_3 = 8$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 65, 48, 32, and 16. It amounts to 28.015625 work units. The final RMSE value obtained on each level from coarsest to finest respectively is .0683368, .0680508, .0678371, and .0677086.[65]

Here, we did not compute a speed-up factor since it took too many iterations on the finest level. [See the Table 5-13.] The Gauss-Seidel method is indeed very slow for this example.

With same $t_0 = 47$, we varied n_2. We used $s_0 = 4$, $n_1 = 8$, $n_2 = 2$, and $n_3 = 8$. The total number of iterations performed on each level from coarsest to finest respectively is 103, 112, 48, and 16. It amounts to 36.609375 work units. The final RMSE value obtained on each level from coarsest to finest respectively is .0642819, .0640420, .0638443, and .0637243. For this example, more work units are required but smaller final RMSE values are obtained as well.

For the second set, $t_0 = 108$ was used. The RMSE value .00968758 was obtained at the coarsest level after 108 iterations. For the other parameters, $s_0 = 8$, $n_1 = 20$, $n_2 = 1$, and $n_3 = 20$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 132, 120, 80, and 40. It amounts to 69.5625 work units. The final RMSE value obtained on each level from coarsest to finest respectively is .00562791, .00568184, .00574512, and .00577894.

Before concluding this section, we examine the range of execution time for the conjugate gradient and the Chebyshev acceleration methods. For instance, we consider the result of the first parameter set where the accuracies obtained at the coarsest level are close to .1 of the initial RMSE value. At one extreme end, the iterations on the finest level only using the conjugate gradient method takes 50063217 machine cycles. At the other end, the multigrid execution with the optimal Chebyshev acceleration method takes 2466961 machine cycles. When we compare these two, we get a speed-up factor of 50063217 / 2466961 = 20.2935. In a similar way, for the second set, we get a speed-up factor of 107529633 / 3232661 = 33.2635.

The traced execution result of the first parameter set for each iteration method is given in section

[65]When smaller parameter values are used, smaller work units are obtained. With $s_0 = 4$, $n_1 = 6$, $n_2 = 1$, and $n_3 = 6$, the total number of iterations performed on each level from coarsest to finest respectively is 59, 36, 24, and 12. It amounts to 21.171875 work units. The final RMSE value obtained on each level from coarsest to finest respectively is .0733860, .0730684, .0728650, and .0727331.

I.3.2 of Appendix with further explanations.

5.3.2.2 Experiments on a Sphere

The second synthetic image we tried on the multigrid methods is the upper hemisphere of a sphere, the same one used in section 5.3.1.3, with both the depth and the orientation constraints. The densities of the constraints are 2% and 15%.

The three-level multigrid algorithms were run. The sizes of the mesh are reduced from 128×128 to 64×64 and 32×32. The constraints at the coarser levels are generated by sampling the same spherical surface. When the density at the finest level is 2.02%, we have the densities of 7.71% and 26.19% at the coarser levels. When the density at the finest level is 14.93%, we have the densities of 47.47% and 89.52% at the coarser levels.

Before showing the result of multigrid algorithm, we show the result for each level in Table 5-15 and 5-17. The initial RMSE values were .687068, .698895, and .707767 for the level $l = 1, 2,$ and 3, respectively.

For the adaptive Chebyshev acceleration method, we show the result with more accurate initial estimates only. For the initial estimates of m_E, we used -2.3. When the density at the finest level is 15%, we used .91, .97, and .99 for the level $l = 1, 2,$ and 3, respectively, as the improved initial estimates of M_E.[66] When the density at the finest level is 2%, we used .985, .995, and .9998 for the level $l = 1, 2,$ and 3, respectively.[67] In the optimal Chebyshev acceleration and the weighted Jacobi methods, we simply used these initial estimates for the real eigenvalues.

When both the depth and the orientation constraints exist, we have derived in section 4.3.3.3 that at level l it takes $5856 \times ((log_2 s) - (L - l)) + 34825$ machine cycles for an iteration of the conjugate gradient method. Thus, an iteration on each level from coarsest to finest respectively takes 64105, 69961, and 75817 machine cycles. We have derived also that at level l it takes

[66]When the initial estimates of m_E and M_E were -3.0 and 0.0, we obtained .917046, .971150, and .995370 as the estimates of M_E values at the iteration steps of 20, 55, and 123 for the level $l = 1, 2,$ and 3, respectively.

[67]When the initial estimates of m_E and M_E were -3.0 and 0.0, we obtained .988769, .998959, and .999869 as the estimates of M_E values at the iteration steps of 77, 261, and 705 for the level $l = 1, 2,$ and 3, respectively.

$4392 \times ((log_2 s) - (L - l)) + 17712$ machine cycles for an iteration of the adaptive Chebyshev acceleration method. Thus, an iteration on each level from coarsest to finest respectively takes 39672, 44064, and 48456 machine cycles. For an iteration of the optimal Chebyshev acceleration and the weighted Jacobi methods, we have derived that at all levels it takes 16573 and 15099 machine cycles, respectively.

In Table 5-16 and 5-18, we show the normalized number of iterations. We compare the normalized number of iterations of three iterative methods. We consider first the result of 15% density. When the RMSEs are reduced to .05, the normalized numbers are 228.7, 225.1, and 75 for the conjugate gradient, the adaptive, and the optimal Chebyshev acceleration methods, respectively, for the finest level. The normalized numbers are 42.5, 43.1, and 18 for the coarsest level. We consider now the result of 2% density. When the RMSEs are reduced to .1, the normalized numbers are 1075.1, 649.1, and 222 for the finest level and 104.4, 74.2, and 31 for the coarsest level.

For the conjugate gradient and the Chebyshev acceleration methods, we compare the amount of time taken to arrive at the same final RMSE values at the finest level, when iterations are carried out on the finest level only versus multigrid. For the Gauss-Seidel method, we compare work units.

We analyze the result of the conjugate gradient method first. When the density of the constraints is 15%, $t_0 = 5$ was used. The RMSE value .0666681, nearest to .1 of initial value, was obtained at the coarsest level after 5 iterations. For the other parameters, $s_0 = 3$, $n_1 = 3$, $n_2 = 1$, and $n_3 = 3$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 11, 12, and 6. The final RMSE value obtained on each level from coarsest to finest respectively is .0420933, .0377653, and .0334779. When the relaxation is carried out on the finest level only, the RMSE value .0325647 is obtained after 51 iterations.

For the three-level multigrid algorithm with $n_2 = 1$, 2 prolongation operations are invoked in procedure FMRA, 3 restriction operations in FMC, and 3 prolongation operations in FMC. We have derived in section 4.3.3.4 that it takes 52383, 29912, and 53847 machine cycles for a prolongation operation in FMRA, a restriction operation in FMC, and a prolongation operation in

RMSE	Conjugate Gradient			Adapt. Chebyshev Accel.			Opt. Chebyshev Accel.		
	l=3	*l*=2	*l*=1	*l*=3	*l*=2	*l*=1	*l*=3	*l*=2	*l*=1
0.5	11	4	1	12	6	4	12	6	4
0.2	24	8	2	28	10	8	28	10	8
0.1	37	13	5	48	14	11	48	14	11
0.05	50	18	11	77	20	18	75	20	18
0.02	72	**	**	112	**	**	132	**	**

RMSE	Gauss-Seidel			Weighted Jacobi		
	l=3	*l*=2	*l*=1	*l*=3	*l*=2	*l*=1
0.5	35	9	4	114	29	14
0.2	98	22	10	323	71	33
0.1	161	33	15	529	108	49
0.05	239	47	24	788	153	82
0.02	409	**	**	1351	***	**

Table 5-15: Number of Iterations on Fine/Coarse Levels (sphere : 15% density)

RMSE	Conjugate Gradient			Adapt. Chebyshev Accel.			Opt. Chebyshev Accel.		
	l=3	l=2	l=1	l=3	l=2	l=1	l=3	l=2	l=1
0.5	50.3	16.9	3.9	35.1	16.0	9.6	12	6	4
0.2	109.8	33.8	7.7	81.9	26.6	19.2	28	10	8
0.1	169.3	54.9	19.3	140.3	37.2	26.3	48	14	11
0.05	228.7	76.0	42.5	225.1	53.2	43.1	75	20	18
0.02	329.4	****	****	327.5	****	****	132	**	**

RMSE	Weighted Jacobi		
	l=3	l=2	l=1
0.5	103.9	26.4	12.8
0.2	294.3	64.7	30.1
0.1	482.0	98.4	44.6
0.05	717.9	139.4	74.7
0.02	1230.8	*****	****

Table 5-16: Normalized Number of Iterations on Fine/Coarse Levels (sphere : 15% density)

RMSE	Conjugate Gradient			Adapt. Chebyshev Accel.			Opt. Chebyshev Accel.		
	$l=3$	$l=2$	$l=1$	$l=3$	$l=2$	$l=1$	$l=3$	$l=2$	$l=1$
0.5	69	22	8	60	22	8	60	22	8
0.2	166	50	18	146	70	18	146	70	18
0.1	235	73	27	222	127	31	222	130	31
0.05	356	127	**	399	204	**	399	263	**

RMSE	Gauss-Seidel			Weighted Jacobi		
	$l=3$	$l=2$	$l=1$	$l=3$	$l=2$	$l=1$
0.5	661	91	18	2187	303	61
0.2	2756	308	52	####	1020	172
0.1	5221	551	91	####	1825	305
0.05	9145	1083	**	####	3548	***

Table 5-17: Number of Iterations on Fine/Coarse Levels (sphere : 2% density)

RMSE	Conjugate Gradient			Adapt. Chebyshev Accel.			Opt. Chebyshev Accel.		
	l=3	l=2	l=1	l=3	l=2	l=1	l=3	l=2	l=1
0.5	315.7	92.9	30.9	175.4	58.5	19.2	60	22	8
0.2	759.4	211.1	69.6	426.9	186.1	43.1	146	70	18
0.1	1075.1	308.2	104.4	649.1	337.7	74.2	222	130	31
0.05	1628.6	536.1	*****	1166.6	542.4	****	399	263	**

RMSE	Weighted Jacobi		
	l=3	l=2	l=1
0.5	1992.5	276.1	55.6
0.2	######	929.3	156.7
0.1	######	1662.7	277.9
0.05	######	3232.4	*****

Table 5-18: Normalized Number of Iterations on Fine/Coarse Levels (sphere : 2% density)

FMC, respectively, when both constraints exist. Therefore, 356043 machine cycles should be added to account for the restriction and prolongation operations.

For the iterations on the finest level only, it takes 3866667 machine cycles. For the multigrid algorithm, it takes 2355632 machine cycles. When the numbers of machine cycles are divided, we get a speed-up factor of 3866667 / 2355632 = 1.6415.

When the density of the constraints is 2%, $t_0 = 27$ was used. The RMSE value .0687782 was obtained at the coarsest level after 27 iterations. For the other parameters, $s_0 = 4$, $n_1 = 5$, $n_2 = 1$, and $n_3 = 5$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 35, 20, and 10. The final RMSE value obtained on each level from coarsest to finest respectively is .0672145, .0616802, and .0578548. When the relaxation is carried out on the finest level only, the RMSE value .0580171 is obtained after 263 iterations. After carrying out a similar analysis, we get a speed-up factor of 19939871 / 4757108 = 4.1916.

We consider now the result of the adaptive Chebyshev acceleration method. When the density of the constraints is 15%, $t_0 = 11$ was used. The RMSE value .0682193 was obtained at the coarsest level after 11 iterations. For the other parameters, $s_0 = 3$, $n_1 = 3$, $n_2 = 1$, and $n_3 = 3$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 17, 12, and 6. The final RMSE value obtained on each level from coarsest to finest respectively is .0424810, .0389937, and .0357214. When the relaxation is carried out on the finest level only, the RMSE value .0356335 is obtained after 77 iterations. After carrying out a similar analysis, we get a speed-up factor of 3731112 / 1849971 = 2.0168.

When the density of the constraints is 2%, $t_0 = 31$ was used. The RMSE value .0686763 was obtained at the coarsest level after 31 iterations. For the other parameters, $s_0 = 4$, $n_1 = 7$, $n_2 = 1$, and $n_3 = 7$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 39, 28, and 14. The final RMSE value obtained on each level from coarsest to finest respectively is .0632827, .0581492, and .0542605. When the relaxation is carried out on the finest level only, the RMSE value .0542546 is obtained after 272 iterations. After carrying out a similar analysis, we get a speed-up factor of 13180032 / 3815427 = 3.4544.

For the optimal Chebyshev acceleration and the weighted Jacobi methods, we repeated the experiments with two densities.

For the optimal Chebyshev acceleration method, we used the same estimates of the eigenvalues with the same parameters as in the adaptive Chebyshev acceleration method. We have the (nearly) same final RMSE values but the total execution time is faster.

When the density of the constraints is 15%, $t_0 = 11$ was used. The RMSE value .0682193 was obtained at the coarsest level after 11 iterations. For the other parameters, $s_0 = 3$, $n_1 = 3$, $n_2 = 1$, and $n_3 = 3$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 17, 12, and 6. The final RMSE value obtained on each level from coarsest to finest respectively is .0424810, .0389937, and .0357214. When the relaxation is carried out on the finest level only, the RMSE value .0353510 is obtained after 75 iterations. After carrying out an analysis, we get a speed-up factor of 1242975 / 936098 = 1.3278.

When the density of the constraints is 2%, $t_0 = 31$ was used. For the other parameters, $s_0 = 4$, $n_1 = 7$, $n_2 = 1$, and $n_3 = 7$ were used. We have the identical results of final RMSE values as in the adaptive Chebyshev acceleration method. After carrying out an analysis, we get a speed-up factor of 4507856 / 1698456 = 2.6541.

For the weighted Jacobi method, we repeated the experiments with two densities.

When the density of the constraints is 15%, $t_0 = 49$ was used. The RMSE value .0685087 was obtained at the coarsest level after 49 iterations. For the other parameters, $s_0 = 6$, $n_1 = 8$, $n_2 = 1$, and $n_3 = 8$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 61, 32, and 16. (This amounts to 27.8125 work units.) The final RMSE value obtained on each level from coarsest to finest respectively is .0445619, .0410180, and .0375995. When the relaxation is carried out on the finest level only, the RMSE value .0376335 is obtained after 762 iterations, which amounts to the same number of work units. After carrying out an analysis, we get a speed-up factor of 12628626 / 2162500 = 5.8398. (With the multigrid approach, work units are reduced by a factor of 762 / 27.8125 = 27.3978.)[68]

When the density of the constraints is 2%, t_0 = 305 was used. The RMSE value .0686376 was obtained at the coarsest level after 305 iterations. For the other parameters, s_0 = 7, n_1 = 10, n_2 = 1, and n_3 = 10 were used. The total number of iterations performed on each level from coarsest to finest respectively is 319, 40, and 20. (This amounts to 49.9375 work units.) The final RMSE value obtained on each level from coarsest to finest respectively is .0705693, .0654956, and .0619304. The number of machine cycles required for the execution of the multigrid algorithm is 6078564.

For the Gauss-Seidel method, we repeated the experiments with two densities.

When the density of the constraints is 15%, t_0 = 15 was used. The RMSE value .0657366 was obtained at the coarsest level after 15 iterations. For the other parameters, s_0 = 3, n_1 = 3, n_2 = 1, and n_3 = 3 were used. The total number of iterations performed on each level from coarsest to finest respectively is 21, 12, and 6. This amounts to 10.3125 work units. The final RMSE value obtained on each level from coarsest to finest respectively is .0386824, .0347583, and .0309097. When the relaxation is carried out on the finest level only, the RMSE value .0309190 is obtained after 257 iterations, which amounts to the same number of work units. With the multigrid approach, work units are reduced by a factor of 257 / 10.3125 = 24.9212.

When the density of the constraints is 2%, t_0 = 91 was used. The RMSE value .0687677 was obtained at the coarsest level after 91 iterations. For the other parameters, s_0 = 4, n_1 = 6, n_2 = 1, and n_3 = 6 were used. The total number of iterations performed on each level from coarsest to finest respectively is 99, 24, and 12. This amounts to 24.1875 work units. The final RMSE value obtained on each level from coarsest to finest respectively is .0682326, .0630561, and .0593441. When the relaxation is carried out on the finest level only, the RMSE value .0593413 is obtained

[68]We show also the result with smaller parameter values to compare with other iterative methods. With s_0 = 3, n_1 = 3, n_2 = 1, and n_3 = 3, the total number of iterations performed on each level from coarsest to finest respectively is 55, 12, and 6. (This amounts to 12.4375 work units.) The final RMSE value obtained on each level from coarsest to finest respectively is .0574017, .0546859, and .0520999. When the relaxation is carried out on the finest level only, the RMSE value .0520627 is obtained after 636 iterations. After carrying out an analysis, we get a speed-up factor of 10540428 / 1565872 = 6.7314. (With the multigrid approach, work units are reduced by a factor of 636 / 12.4375 = 51.1357.)

after 6007 iterations. With the multigrid approach, work units are reduced by a factor of 6007 / 24.1875 = 248.3514.

Compared to the poor speedups of the conjugate gradient and the Chebyshev acceleration methods, we have significantly higher speedups for the weighted Jacobi and the Gauss-Seidel methods. This is due to two reasons. First, for comparison, we used the execution time for the conjugate gradient and the Chebyshev acceleration methods, while we used the work units for the Gauss-Seidel method. Note that when we use both execution time and work units for the weighted Jacobi method, the speed-up factor of execution time is smaller. Second, compared to the conjugate gradient and the Chebyshev acceleration methods, the performance of the weighted Jacobi and the Gauss-Seidel methods on a single-grid is very poor.

Before concluding this section, we examine again the range of execution time for the conjugate gradient and the Chebyshev acceleration methods.

First, we consider the case where the density of the constraints is 15%. At one extreme end, the iterations on the finest level only using the conjugate gradient method takes 3866667 machine cycles. At the other end, the multigrid execution with the optimal Chebyshev acceleration method takes 936098 machine cycles. When we compare these two, we get a speed-up factor of 3866667 / 936098 = 4.1306.

Now, we consider the case where the density of the constraints is 2%. At one extreme end, the iterations on the finest level only using the conjugate gradient method takes 19939871 machine cycles. At the other end, the multigrid execution with the optimal Chebyshev acceleration method takes 1698456 machine cycles. When we compare these two, we get a speed-up factor of 19939871 / 1698456 = 11.7400.

5.4 Experiments on Real Images

5.4.1 Multigrid Algorithms

For the real images of range data, we used $\beta^{h_j} = 2.0 / h_j$[69] and $0.2 / h_j$ for a quasi-spherical object and a soda can, respectively.

5.4.1.1 Experiments on Actual Range Data from a Quasi-spherical Object

A data set of 976 depth constraints in a (x, y, z) format was scaled to generate data in a (i, j, d) format, where $0 \le i, j \le 127$ and $0.01 \le d \le 1.0$. A SIMD region labeling routine was then run to mark the depth continuous region that contains given depth constraints. The generated region contained 10628 nodes. Next, we patched the region by adding 84 more depth continuous nodes to get a smoother boundary.

The three-level multigrid algorithms were run. The sizes of the mesh are reduced from 128×128 to 64×64 and 32×32. The constraints at the coarser levels are generated by local averaging the constraints at the finest level. At the finest level ($l = 3$), 976 nodes are constrained out of 10712 depth continuous nodes. At the medium level ($l = 2$), 961 nodes are constrained out of 2748 depth continuous nodes. At the coarsest level ($l = 1$), 605 nodes are constrained out of 720 depth continuous nodes. Thus, the density on each level from coarsest to finest respectively is 84.03%, 34.97%, and 9.11%.

Before showing the result of multigrid algorithm, we show the result for each level in Table 5-19. The initial RMSD values were .779238, .747674, and .742538 for the level l = 1, 2, and 3, respectively.

For the adaptive Chebyshev acceleration method, we show the result with more accurate initial

---

[69]In comparison with $\beta^{h_j} = 2.0 / h_j^2$, used for the synthetic images of the cylinder and sphere examples, we can give following explanations. After $1 / h_j^2$ are factored out, $\beta^{h_j} = 2.0 / h_j^2$ contributes a constant weight of 2.0 for all levels, while $\beta^{h_j} = 2.0 / h_j$ contributes smaller weights of 0.8, 0.4, and 0.2 for the coarsest, medium, and the finest level when three-level multigrid scheme is used with $h_1 = 0.4$, $h_2 = 0.2$, and $h_3 = 0.1$. In physical terms, equal-strength springs are used for all levels in the synthetic images, while generally weaker, though gradually stronger, springs are used for coarser levels in the real images. These choices are meant to handle the noise present in the real images. Note that in the real images the constraints at coarser levels are more reliable since the averaging process responsible for the generation of these constraints tends to cancel out the noise. Since we use generally looser springs, the irregularities caused by spurious noise can be lessened. However, overall convergence is slowed down compared to the synthetic images of similar size with similar density of constraints. In the Chebyshev acceleration method, this effect is manifested as a bigger M(G) value which is closer to 1.0, the theoretical upper bound.

estimates only. For the initial estimates of m_E, we used -2.3. As the improved initial estimates of M_E, we used .98, .995, and .9995 for the level l = 1, 2, and 3, respectively.[70] In the optimal Chebyshev acceleration and the weighted Jacobi methods, we simply used these initial estimates for the real eigenvalues.

When the depth constraints exist only, we have derived in section 4.3.3.3 that at level l it takes $5856 \times ((log_2 s) - (L - l)) + 32307$ machine cycles for an iteration of the conjugate gradient method. Thus, an iteration on each level from coarsest to finest respectively takes 61587, 67443, and 73299 machine cycles. We have derived also that at level l it takes $4392 \times ((log_2 s) - (L - l)) + 16453$ machine cycles for an iteration of the adaptive Chebyshev acceleration method. Thus, an iteration on each level from coarsest to finest respectively takes 38413, 42805, and 47197 machine cycles. For an iteration of the optimal Chebyshev acceleration and the weighted Jacobi methods, we have derived that at all levels it takes 15314 and 13840 machine cycles, respectively.

In Table 5-20, we show the normalized number of iterations. We compare the normalized number of iterations of three iterative methods. When the RMSDs are reduced to .05, the normalized numbers are 607.9, 477.7, and 155 for the conjugate gradient, the adaptive, and the optimal Chebyshev acceleration methods, respectively, for the finest level. The normalized numbers are 68.4, 62.7, and 25 for the coarsest level.

For the conjugate gradient and the Chebyshev acceleration methods, we compare the amount of time taken to arrive at the same final RMSD values at the finest level, when iterations are carried out on the finest level only versus multigrid. For the Gauss-Seidel method, we compare work units. For the weighted Jacobi method, we compare both execution time and work units.

We analyze the result of the conjugate gradient method first. We used $t_0 = 9$. The RMSD value .0791359 was obtained at the coarsest level after 9 iterations. For the other parameters, $s_0 = 3$, $n_1 = 5$, $n_2 = 1$, and $n_3 = 5$ were used. The total number of iterations performed on each level from

[70]When the initial estimates of m_E and M_E were -3.0 and 0.0, we obtained .983780, .995785, and .999489 as the estimates of M_E values at the iteration steps of 50, 102, and 296 for the level l = 1, 2, and 3, respectively.

RMSD	Conjugate Gradient			Adapt. Chebyshev Accel.			Opt. Chebyshev Accel.		
	l=3	*l*=2	*l*=1	*l*=3	*l*=2	*l*=1	*l*=3	*l*=2	*l*=1
0.5	20	7	2	37	14	6	37	14	6
0.2	49	17	4	61	24	9	61	24	9
0.1	83	29	9	100	34	17	100	34	17
0.05	127	46	17	155	54	25	155	54	25

RMSD	Gauss-Seidel			Weighted Jacobi		
	l=3	*l*=2	*l*=1	*l*=3	*l*=2	*l*=1
0.5	401	51	10	1323	167	32
0.2	1017	128	23	3366	425	77
0.1	1556	195	35	5161	652	116
0.05	2312	289	49	####	981	168

Table 5-19: Number of Iterations on Fine/Coarse Levels (sphere : range data)

RMSD	Conjugate Gradient			Adapt. Chebyshev Accel.			Opt. Chebyshev Accel.		
	l=3	*l*=2	*l*=1	*l*=3	*l*=2	*l*=1	*l*=3	*l*=2	*l*=1
0.5	95.7	30.8	8.0	114.0	39.1	15.1	37	14	6
0.2	234.5	74.9	16.1	188.0	67.1	22.6	61	24	9
0.1	397.3	127.7	36.2	308.2	95.0	42.6	100	34	17
0.05	607.9	202.6	68.4	477.7	150.9	62.7	155	54	25

RMSD	Weighted Jacobi		
	l=3	*l*=2	*l*=1
0.5	1195.7	150.9	28.9
0.2	3042.0	384.1	69.6
0.1	4664.2	589.2	104.8
0.05	######	886.6	151.8

Table 5-20: Normalized Number of Iterations on Fine/Coarse Levels (sphere : range data)

coarsest to finest respectively is 15, 20, and 10. The final RMSD obtained on each level from coarsest to finest respectively is .0772859, .0769072, and .0722429. When the relaxation is carried out on the finest level only, the RMSD value .0720139 is obtained after 85 iterations.

For the three-level multigrid algorithm with $n_2 = 1$, 2 prolongation operations are invoked in procedure FMRA, 3 restriction operations in FMC, and 3 prolongation operations in FMC. We have derived in section 4.3.3.4 that it takes 52383, 33700, and 57000 machine cycles for a prolongation operation in FMRA, a restriction operation in FMC, and a prolongation operation in FMC, respectively, when the depth constraints exist only and local averaging is used for the restriction operation. Therefore, 376866 machine cycles should be added to account for the restriction and prolongation operations.

For the iterations on the finest level only, it takes 6230415 machine cycles. For the multigrid algorithm, it takes 3382521 machine cycles. When the numbers of machine cycles are divided, we get a speed-up factor of 6230415 / 3382521 = 1.8419.

We consider now the result of the adaptive Chebyshev acceleration method. We used $t_0 = 17$. The RMSD value .0790101 was obtained at the coarsest level after 17 iterations. For the other parameters, $s_0 = 4$, $n_1 = 6$, $n_2 = 1$, and $n_3 = 6$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 25, 24, and 12. The final RMSD value obtained on each level from coarsest to finest respectively is .0700527, .0695689, and .0672210. When the relaxation is carried out on the finest level only, the RMSD value .0669729 is obtained after 110 iterations. After carrying out a similar analysis, we get a speed-up factor of 5191670 / 2930875 = 1.7714.

For the optimal Chebyshev acceleration method, we used the same estimates of the eigenvalues with the same parameters as in the adaptive Chebyshev acceleration method. We have the same final RMSD values but the total execution time is faster. After carrying out an analysis, we get a smaller speed-up factor of 1684540 / 1311020 = 1.2849.

For the weighted Jacobi method, $t_0 = 116$ was used. The RMSD value .0777064 was obtained at the coarsest level after 116 iterations. For the other parameters, $s_0 = 12$, $n_1 = 16$, $n_2 = 1$, and $n_3 =$

16 were used. The total number of iterations performed on each level from coarsest to finest respectively is 140, 64, and 32. (This amounts to 56.75 work units.) The final RMSD value obtained on each level from coarsest to finest respectively is .0759227, .0739616, and .0721752. When the relaxation is carried out on the finest level only, the RMSD value .0721828 is obtained after 5243 iterations, which amounts to the same number of work units. After carrying out an analysis, we get a speed-up factor of 72563120 / 3643106 = 19.9179. (With the multigrid approach, work units are reduced by a factor of 5243 / 56.75 = 92.3877.)

For the Gauss-Seidel method, we used $t_0 = 35$. The RMSD value .0767458 was obtained at the coarsest level after 35 iterations. For the other parameters, $s_0 = 4$, $n_1 = 6$, $n_2 = 1$, and $n_3 = 6$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 43, 24, and 12. This amounts to 18.6875 work units. The final RMSD value obtained on each level from coarsest to finest respectively is .0692349, .0678860, and .0662348. When the relaxation is carried out on the finest level only, the RMSD value .0662035 is obtained after 1657 iterations, which amounts to the same number of work units. With the multigrid approach, work units are reduced by a factor of 1657 / 18.6875 = 88.6689.

We examine again the range of execution time for the conjugate gradient and the Chebyshev acceleration methods. At one extreme end, the iterations on the finest level only using the conjugate gradient method takes 6230415 machine cycles. At the other end, the multigrid execution with the optimal Chebyshev acceleration method takes 1311020 machine cycles. When we compare these two, we get a speed-up factor of 6230415 / 1311020 = 4.7523.

Again, note the relatively small speed-ups as in the synthetic examples.

5.4.1.2 Experiments on Actual Range Data from a Soda Can

The three-level multigrid algorithms were run. The sizes of the mesh are reduced from 128×128 to 64×64 and 32×32. The constraints at the coarser levels are generated by local averaging the constraints at the finest level. At the finest level ($l = 3$), 1483 nodes are constrained out of 12080 depth continuous nodes. At the medium level ($l = 2$), 1312 nodes are constrained out of 3084 depth continuous nodes. At the coarsest level ($l = 1$), 660 nodes are constrained out of 805 depth continuous nodes. Thus, the density on each level from coarsest to finest respectively is 81.99%, 42.54%, and 12.28%.

Before showing the result of multigrid algorithm, we show the result for each level in Table 5-21. The initial RMSD values were .926353, .919643, and .885797 for the level l = 1, 2, and 3, respectively.

For the adaptive Chebyshev acceleration method, we show the result with more accurate initial estimates only. For the initial estimates of m_E, we used -2.3. As the improved initial estimates of M_E, we used .995, .999, and .9999 for the level l = 1, 2, and 3, respectively.[71]

In Table 5-22, we show the normalized number of iterations. We compare the normalized number of iterations of three iterative methods. When the RMSDs are reduced to .2, the normalized numbers are 1258.8, 859.9, and 279 for the conjugate gradient, the adaptive, and the optimal Chebyshev acceleration methods, respectively, for the finest level. The normalized numbers are 160.9, 155.5, and 58 for the coarsest level.

We analyze the result of the conjugate gradient method first. We used $t_0 = 40$. The RMSD value .187266 was obtained at the coarsest level after 40 iterations. For the other parameters, $s_0 = 6$, $n_1 = 8$, $n_2 = 1$, and $n_3 = 8$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 52, 32, and 16. The final RMSD obtained on each level from coarsest to finest respectively is .166422, .185920, and .221586. When the relaxation is carried out on the finest level only, the RMSD value .222083 is obtained after 175 iterations.[72]

The iterations on three levels take 6533484 machine cycles. Then, 376866 machine cycles are added to account for the restriction and prolongation operations. Thus, for the multigrid algorithm, it takes 6910350 machine cycles. For the iterations on the finest level only, it takes 12827325 machine cycles. When the numbers of machine cycles are divided, we get a speed-up

[71]When the initial estimates of m_E and M_E were -3.0 and 0.0, we obtained .995935, .999189, and .999882 as the estimates of M_E values at the iteration steps of 61, 164, and 428 for the level l = 1, 2, and 3, respectively.

[72]For the multigrid algorithm, the discrete L_2-norm of the residual vector obtained on each level from coarsest to finest respectively is .00149900, .00164972, and .00557440. For the single-grid algorithm, the norm of the residual vector obtained on the finest level is .00831499.

RMSD	Conjugate Gradient			Adapt. Chebyshev Accel.			Opt. Chebyshev Accel.		
	l=3	*l*=2	*l*=1	*l*=3	*l*=2	*l*=1	*l*=3	*l*=2	*l*=1
0.8	25	8	7	52	19	10	52	19	10
0.6	58	12	11	85	31	16	85	31	16
0.5	74	20	13	103	38	20	103	38	20
0.4	98	33	19	125	47	26	125	47	26
0.2	263	67	40	279	92	62	279	92	58

RMSD	Gauss-Seidel			Weighted Jacobi		
	l=3	*l*=2	*l*=1	*l*=3	*l*=2	*l*=1
0.8		106	27			
0.6		258	66			
0.5		361	92			
0.4		500	126			
0.2		1144	282			

Table 5-21: Number of Iterations on Fine/Coarse Levels (soda can : range data)

RMSD	Conjugate Gradient			Adapt. Chebyshev Accel.			Opt. Chebyshev Accel.		
	l=3	*l*=2	*l*=1	*l*=3	*l*=2	*l*=1	*l*=3	*l*=2	*l*=1
0.8	119.7	35.2	28.2	160.3	53.1	25.1	52	19	10
0.6	277.6	52.8	44.2	262.0	86.6	40.1	85	31	16
0.5	354.2	88.1	52.3	317.4	106.2	50.2	103	38	20
0.4	469.1	145.3	76.4	385.2	131.4	65.2	125	47	26
0.2	1258.8	295.1	160.9	859.9	257.2	155.5	279	92	58

RMSD	Weighted Jacobi		
	l=3	*l*=2	*l*=1
0.8			
0.6			
0.5			
0.4			
0.2			

Table 5-22: Normalized Number of Iterations on Fine/Coarse Levels (soda can : range data)

factor of 12827325 / 6910350 = 1.8563.[73]

We consider now the result of the adaptive Chebyshev acceleration method. We used $t_0 = 62$. The RMSD value .185203 was obtained at the coarsest level after 62 iterations. For the other parameters, $s_0 = 7$, $n_1 = 9$, $n_2 = 1$, and $n_3 = 9$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 76, 36, and 18. The final RMSD value obtained on each level from coarsest to finest respectively is .167187, .186873, and .220360. When the relaxation is carried out on the finest level only, the RMSD value .220293 is obtained after 188 iterations.[74] After carrying out a similar analysis, we get a speed-up factor of 8873036 / 5686780 = 1.5603.

For the optimal Chebyshev acceleration method, we used $t_0 = 58$. The RMSD value .185933 was obtained at the coarsest level after 58 iterations. For the other parameters, $s_0 = 7$, $n_1 = 9$, $n_2 = 1$, and $n_3 = 9$ were used. The total number of iterations performed on each level from coarsest to finest respectively is 72, 36, and 18. The final RMSD value obtained on each level from coarsest to finest respectively is .167904, .187397, and .221142. When the relaxation is carried out on the finest level only, the RMSD value .221389 is obtained after 187 iterations. After carrying out an analysis, we get a smaller speed-up factor of 2863718 / 2306430 = 1.2416.

We examine again the range of execution time for the conjugate gradient and the Chebyshev acceleration methods. At one extreme end, the iterations on the finest level only using the conjugate gradient method takes 12827325 machine cycles. At the other end, the multigrid execution with the optimal Chebyshev acceleration method takes 2306430 machine cycles. When we compare these two, we get a speed-up factor of 12827325 / 2306430 = 5.5616.

[73]When smaller parameter values are used, more speed-up is possible. With $s_0 = 4$, $n_1 = 6$, $n_2 = 1$, and $n_3 = 6$, the final RMSD value obtained on each level from coarsest to finest respectively is .175867, .195881, and .233090. When the relaxation is carried out on the finest level only, the RMSD value .233180 is obtained after 166 iterations. Here, we get a speed-up factor of 12167634 / 5831262 = 2.0866.

[74]For the multigrid algorithm, the discrete L_2-norm of the residual vector obtained on each level from coarsest to finest respectively is .00041147, .00122146, and .00700397. For the single-grid algorithm, the norm of the residual vector obtained on the finest level is .00211884.

5.5 Implementation Experiences

We have rewritten the existing NON-VON simulator [Choi 85a] to handle the floating point operations. Our simulators ran on DEC-20, VAX 11/750, and IBM 4381 machines. The initial version of the software was written in PSL. By transporting our software progressively across these machines, we were able to handle bigger images. The size of the biggest image we could handle under PSL was 16 × 16, 32 × 32, and 64 × 64 respectively. Then the entire code was rewritten in FORTRAN 77 and transported to an IBM 4381. This brought two advantages for us. We could run bigger images, up to 128 × 128 images. Our program ran in the maximum virtual user space allowed, 16 Mbytes of virtual machine, under CMS. Secondly, this simulation ran 3 to 5 times faster.

It was straightforward to implement and test the SIMD control, mesh connections, and tree topology aspects of the computation. We later simulated the Gauss-Seidel method, which requires only mesh connections, for comparison.

The language VS FORTRAN provides three precisions for floating-point numbers [IBM 84]. A real number can occupy 4, 8, or 16 bytes of storage, which are, approximately, 6, 15, and 32 decimal digits. In our implementation, we chose the real numbers of 8 bytes long, i.e., double precision.

We examine the general SIMD programming techniques we used. Usually, each procedure has a set of associated boolean predicates which made the hierarchical construction of SIMD programs easier. The general form of the predicates is as follows: "For such and such PEs, such and such conditions hold." The predicates are classified as *input*, *output*, *pre-condition*, and *post-condition*. The input and output predicates state how and where the input vectors (or scalar values) are prepared before entering this procedure and the output vectors (or scalar values) are generated as the execution result. The pre-condition and post-conditions state what conditions hold, usually about the marking of the set on which operations are to be carried out. The statement about the post-condition helps the calling procedure. Sometimes the pre-conditions remain valid as post-conditions, or some other useful post-conditions are set up. In both cases, the calling procedure can utilize them so that it avoids the unnecessary new set up of context for following actions.

Inside the procedure, predicates are interspersed with actions. The actions are execution of a sequence of SIMD instructions or function calls. At the beginning of the procedure, the pre-condition and optional set up of the context establish the initial conditions. After each minor or major action, depending on the need of detail, new conditions are declared. In our programming development, this served two purposes: *verification* and *expectation*, which are really two sides of the same coin. By stating the condition explicitly, we know what we have done, where we are, and where we are going. In this sense, *verification* is carried out as the informal *proof of correctness* of the SIMD program.[75] The declaration of the conditions reveal the *expectation* or intent of the programmer. The explicit statement of the condition to hold at particular point of the program helps the debugging process. When the program does not work, i.e., does not follow the original intention of the programmer, we can mentally execute the sequence of actions or examine the dumped out contents of registers and memory, especially, when the program is under development using simulator, to find out the bug. But the tracing of the conditions usually helps to correct the wrong behavior of the program, where some action is missing or the sequence of actions has been executed in wrong order. The statement of conditions are declared in hierarchical manner as the sequence of actions are.

This general description will be elaborated further in section I.2 of Appendix where actual listing of SIMD programs are given. The final remark about the declaration of conditions in our program is that these conditions are used in *passive* sense. In our case, they are written as comments, enclosed in curly brackets. We can imagine a hypothetical *active* usage of conditions, i.e., enforcing the conditions through further redundant actions or dynamically checking them, at certain crucial points to ensure the correctness of program execution.

[75]For the rigorous treatment of the correctness proof of the programs running on the sequential machines, see [Mann 74].

5.6 Conclusion and Summary

The actual experiments on the adaptive Chebyshev acceleration and the conjugate gradient methods confirmed that the performance depends strongly on global information. In the case of the adaptive Chebyshev acceleration method, the global parameters are the largest and smallest eigenvalues, where the estimate of the largest eigenvalue M_E is more critical. When the adaptive Chebyshev acceleration method is started with the more accurate initial estimates of the extreme eigenvalues, we observe nearly always that it executes faster than the conjugate gradient method. However, when there are frequent changes or updates of M_E due to the error of the initial estimates, the overall execution time is often slower than steady conjugate gradient method which is not started with any initial global information. Recall that in the optimal Chebyshev acceleration method, many iterations are often required to achieve the asymptotic convergence rate. The slowdown occurs at every change of M_E, when a new Chebyshev polynomial is generated again.

In section 5.2, we gave an interpretation of the estimate m_E based on the system matrix, specifically, the nodal equation of an internal node. For the more accurate estimate of M_E, further theoretical study is needed. In practice, more accurate initial estimates, still low enough to satisfy the condition $m_E \le m(G)$ and $M_E \le M(G)$, may be prepared in a lookup table form. The search parameters for this table may be the number of the depth continuous nodes in the region, which is related to the size of the matrix, and the density of the constraints. Less influential parameters may be the shape of the region, i.e., the two-dimensional spatial extent of the region, and the distribution of the constraints on it. Note that these proposed parameters can be easily obtained through the global communication networks of the SIMD machine. Such global operations as counting, summation, average, minimum, and maximum can be executed in $O(\log_2 s)$ time when the size of the mesh is $s \times s$.

In our experiments, we used the fairly good estimates of m_E and M_E values in the following sense: either the execution time for the adaptive Chebyshev acceleration method was faster than the conjugate gradient method, or the initial estimates were hardly changed at all, or both, until the specified fraction of RMSE is attained. But, theoretically, how accurate the estimates should be? For the single-grid algorithm of the optimal Chebyshev acceleration method, section 4.4 of [Youn

81] discusses the sensitivity of the asymptotic virtual rate of convergence to the estimates of m_E and M_E. But for the multigrid algorithms, the theoretical anaylsis will be more difficult since we have both iterations and interlevel computations, restriction and prolongation. However, we can make general assertions about the accuracy of the estimates. In many cases, since given initial estimates may not be changed, they should be fairly accurate. Nevertheless, because computation does not depend on iterations only, they need not be so accurate as in the single-grid algorithms. Furthermore, the estimates on coarser levels can be computed more easily as well as accurately, since the sizes of the matrices are smaller and the constraints are denser. In practice, during initial iterations on the coarsest level, the Chebyshev acceleration method can be run adaptively for possible updates of the initial estimates. In the case of actual estimate changes, the estimates on other finer levels can be adjusted. After this adjustment, the Chebyshev acceleration method might be run non-adaptively.

For the images with very sparse constraints, the adaptive Chebyshev acceleration and the conjugate gradient methods accelerated by the multigrid approach demonstrated the speed-up by two or seven times or more compared to the execution on the finest level only. From these meager speed-up factors, one might conclude that the acceleration achieved by the multigrid approach may not be worthwhile for the additional hardware and software complexity. Nevertheless, for bigger images, more speed-up is possible. Therefore, the adoption of the multigrid approach should be made after a relevant cost/benefit analysis that depends on each application.

6. Conclusion and Future Directions

6.1 Contributions

In this thesis, we showed how a middle-level computer vision problem, in particular the smoothness constraint propagation problems in early vision which are cast as solving a large system of linear equations with a resulting matrix that is sparse SPD, can be run efficiently on a class of parallel computers. Specifically, we have worked on the depth interpolation problem.

Basically, the speed-ups of the computation have been achieved by two factors. First, we used theoretically better iterative methods. In the Gauss-Seidel method, only local information is used. However, in the Chebyshev (including the adaptive Chebyshev acceleration) and the conjugate gradient methods, local as well as global information are used in each step of the matrix iterations.

Second, all computational steps have been parallelized and can be run on any fine grained SIMD machines with local and global communication networks. We have analyzed the space and time complexity of the two iterative methods based on our abstract SIMD model derived from actual machines built. In particular, we have analyzed the computational and communication costs of parallel computing. Also, we have analyzed two modes of communications, local and global, necessary for local interactions and global summary, respectively.

We have shown the results from two methods, the Chebyshev acceleration and the conjugate gradient, and compared them with the results from the Gauss-Seidel method. We applied these methods to the synthetic and real images and found the degree of improvement.

We also applied the iterative methods to multigrid approach. We have listed in Table 6-1 the speed-up factors of the execution time obtained by the multigrid approach of various iterative methods. (The speed-up factors of the work units are enclosed in parenthesis.) The results in the top two rows are from the synthetic images, while those in the bottom two rows are from the real images of range data. The size of the mesh at the finest level is 128×128. The number of depth continuous nodes in the region, n, for four images are 16384, 12867, 10712, and 12080, respectively. Recall that the size of the matrix is $n \times n$. The densities of the depth constraints are

3.91%, 2.02%, 9.11%, and 12.28%, respectively. The initial accuracies, RMSE or RMSD values, attained at the coarsest level are .1 except the last row where .2 is attained. In general, more speed-up is possible as more iterations are required for convergence of the single-grid algorithms; as the size of the image gets bigger, the density of the constraints is sparser, or smaller errors are wanted.

In Table 6-2, we show the execution time of various iterative methods when the speed-up factors of Table 6-1 are obtained. (The work units are enclosed in parenthesis.) The multigrid execution of the adaptive Chebyshev acceleration method is faster than the conjugate gradient method for all images. Furthermore, the optimal Chebyshev acceleration method is the fastest one. However, we observe that the speed-up factors get smaller as we employ a more optimal mode of execution. In contrast, the speed-up factors achieved by the weighted Jacobi or the Gauss-Seidel methods are rather big, since the convergence rates of the single-grid algorithms (on the finest grid) are very slow. Note also the slowest execution time of the weighted Jacobi method compared to the fastest optimal Chebyshev method, even accelerated with the multigrid approach.

6.2 Implementation Restrictions

Our implementation examples were smaller than most standard camera generated images. The most recent version of our program handled up to 128 × 128 images due to the limitation of the virtual space of the machine where simulation was carried out. Often, images in the real world are 512 × 512 or 1024 × 1024. The SIMD machines now in existence seem to keep up with the demand. The initial prototype of the Connection Machine is a 16K machine, which means it can handle 128 × 128 images. When a bigger machine is built, it will be able to handle larger images.

6.3 Extension of Current Research Work

In this work, we presented a methodology of solving a large system of linear equations where the matrix is sparse SPD in a computationally efficiently way. In particular, we have worked on the depth interpolation problem. In mathematical terms, the particular matrix we worked on is derived from the biharmonic equation. Our methodology, an implementation of efficient iterative methods on a parallel architecture for sparse SPD matrices, can be applied in other areas. We can use other interpolating functions to generate different system matrices for the depth interpolation

Images	Conjugate Gradient	Adaptive Chebyshev	Optimal Chebyshev	Weighted Jacobi	Gauss-Seidel
Cylinder	7.2929	10.4389	7.2878	#######	########
Sphere	4.1916	3.4544	2.6541		(248.3514)
Quasi-spherical	1.8419	1.7714	1.2849	19.9179 (92.3877)	(88.6689)
Soda Can	1.8563	1.5603	1.2416		

Table 6-1: Speed-up Factors of Multigrid Approach

Images	Conjugate Gradient	Adaptive Chebyshev	Optimal Chebyshev	Weighted Jacobi	Gauss-Seidel
Cylinder	6864684	5307974	2466961		(28.015625)
Sphere	4757108	3815427	1698456	6078564 (49.9375)	(24.1875)
Quasi-spherical	3382521	2930875	1311020	3643106 (56.75)	(18.6875)
Soda Can	6910350	5686780	2306430		

Table 6-2: Execution Time of Multigrid Algorithms (in machine cycles[76])

[76]Recall that a cycle time is 100 nanoseconds for our abstract SIMD model.

problem. For instance, see chapter 9 of [Boul 86]. The other constraint propagation problems in early vision may be another choice for extension. For examples of the constraint propagation problems, such as shape from shading, optical flow, etc., see chapter 10 of [Terz 84]. But recall that our methodology is general enough so that it can be applied to the solution of any sparse, SPD matrices.

In terms of the depth interpolation problem in the fuller context of actual computer vision systems, there needs to be added another layer of modules to our solution. In our experiments, we always used the constraints and the discontinuities as the input, and concentrated on efficient solutions of the matrix iteration. In practice, we have two stages, the segmentation and the iteration, where two can alternate to reconstruct a better surface. The segmentation process detects the discontinuities. In the beginning, we have an initial gross segmentation, for example, based on the thresholding of the constraints. But after sufficient number of iteration steps, we may need a refined segmentation which can now be based on the computed smooth surface values. This is addressed in [Schu 83] where the general issue of segmentation versus iteration is considered, though it deals with the motion problem. Especially, for the boundary detection, see chapter 8 of [Schu 83]. For the detection of the discontinuities in the depth interpolation problem, see chapter 9 of [Terz 84] and [Terz 85b]. See also the recent work of [Blak 87].

In the concluding section of chapter 5, we stressed the preparation in a table form of the estimates of the largest and smallest eigenvalues, especially the largest one, of the iteration matrices, if one desires the practical use of the optimal and adaptive Chebyshev acceleration methods. More experiments and further theoretical work will be needed for preparation of these tables. Further investigation of the effects of the density and distribution of the constraints and the shape of the surface upon the extreme eigenvalues will be needed.

In the multigrid methods, we have only explored the effects of a fixed scheme for a multilevel coordination strategy. The study of an accommodative scheme will be beneficial, since it is more flexible and can be executed on the fly with no prepared parameters. Also, it can eliminate possible unnecessary relaxation sweeps, thus increasing speed-up over the single-grid method. It is not clear how great an effect this may have; perhaps the speed-up using theoretically better iterative methods will continue to be found meager, especially since such adaptivity is dependent on global computation of the residual norms.

6.4 Future of Parallel Architecture for Image Processing

We proceed in two steps in this final section. We describe the prospects of near term future first, and then speculate about what may arrive in not too distant future. For the longer term prospect, we only mention possibilities, though research may lead ultimately to such realizations in terms of a large scale system implementation.

The three layers of computer vision problems require different processing needs. On the one hand, low level problems, typified by the edge detection, and middle level ones, for example the depth interpolation problem investigated in detail in this thesis, require SIMD architectures with demanding numerical computational power. On the other, high level problems, for example object recognition, are not defined well, in terms of both algorithms and architectures. The more useful architecture may be in software, not in hardware. In this sense, high level problems may require MIMD architectures (or MSIMD architectures) with symbolic computational power, too. In chapter 3, we noted already that two SIMD machines, the NON-VON and the Connection Machine, support symbolic processing as well as numeric processing. For MIMD architectures, some interesting machines are based on global shared memory, such as the Ultracomputer[77] [Gott 86]. For an integrated vision system, we will need features of both architectures.

As a short exercise, one might review the implementation of the connected component labeling algorithms on three parallel architectures although of middle level vision. [Huss 84] provides the one based on the NON-VON, [TMC 87b] on the Connection Machine, and [Humm 87] on the Ultracomputer. Note that the NON-VON and the Connection Machine are SIMD machines while the Ultracomputer is a MIMD machine. However, in terms of global communication topology, the NON-VON is based on the tree, while the Connection Machine and the Ultracomputer are based on the variations of boolean *n*-cube.

[77]The theoretically inclined reader may consult [Vish 83] where formal parallel computer models are surveyed. The Ultracomputer is interesting since it embodies one of the earliest direct implementation of these theoretical models.

The human brain is one of the most complex structures in the universe. It is thought to consist of perhaps 10^{11} individual neurons. A given neuron in the brain may receive several thousand synaptic connections from other neurons. Hence if the human brain has 10^{11} neurons, then it has at least 10^{14} synapses.[78] The brain is a vast network of connections among neurons. At each point where a nerve fiber forms a connection or synapse onto another neuron, information is transferred and may be transformed or processed. Information is continuously flowing through the multitude of synaptic contacts and networks in the brain. During evolution certain regions of cell bodies where these connections occur have expanded enormously, as in the human cerebral cortex. The brain is not simply a collection of special structures but a vast information processing system [Thom 85].

How can we ever, (or perhaps never,) achieve the performance of the human brain as an information processing system? Part of the answer may lie in the great expansion of the number of processors and the other in the increase of the number of connections. We discussed the SIMD machines with local and global communication networks. When the future version of the Connection Machine that can handle a 1024×1024 image will become available, it will have only 10^6 processors compared to 10^{11} neurons in the human brain. For the connections, instead of several thousand connections at every neuron, it will have only 24 connections per PE, where 4 are for local mesh connections to nearest neighbors and 20 for global connections to boolean cube neighbors.[79] We see the obvious limitations in terms of the number of processors and connections.

One of the promising researches to alleviate this shortfall is the electronic neural networks [Jack 86], [Graf 86]. In recent research work, the electronic neurons are fully interconnected, at least locally. The proposed network complexity is 10^4 neurons and 10^8 synapses. The authors hope

---

[78]For image processing, in each human eye there are about 126 million photoreceptor cells whose impulses are channeled into 1 million ganglion cells. Information from the outside world is increasingly simplified and abstracted as the information travels from the outside to the visual cortex of the brain. There are over 100 million neurons in the human visual cortex, and we do not yet know well the extent of their specialization.

[79]Recall that our application can be run on any SIMD machine with local and global communication networks, for example, the Connection Machine. But when it is run on the tree machine with multiple mesh connections or the pyramid machine, the number of connections is still smaller, 7 for the tree machine and 9 for the pyramid machine, respectively. For each node, there are 4 local mesh connections to its neighbors, 1 to its own parent, 2 or 4 to its own children.

that neural-net algorithms will become one of the main stream AI tools and some neural-net hardwares will become standard tools in pattern classifying machines, where their highly parallel and regular structure will be fully utilized. As an application in the image processing with neural nets, a feature map can be produced by moving image processing kernels through all possible image locations.

Another possibility is optical processing. It provides two advantages compared to existing semiconductor technology. First, the processing may be done at the speed of the photons instead of the electrons. One of the main reasons of the recent scaling down of the supercomputer project in Japan was the technological barrier of building high speed semiconductors, which was expected to be surmountable.[80] Instead, we have steady development in optical processing elements and storage devices. Second, we can have flexible optical connections that are arbitrarily changeable under program control.

The final possibility is the analog, or hybrid, analog-digital processing. Even though the digital computer has been in use for more than 40 years, a lot of processing in nature is done by the analog computation. Analog processing has aroused a lot of interest recently in the vision community, and we can mention two examples. One is the replicating of the function of a neuron [Koch 84]. The other is the depth interpolation problem itself, and more generally, the smoothness constraint propagation problems in early vision. Terzopoulos mentions a cascaded electrical resistance network where there are 2 processors and 10 local connections for an internal depth constrained node [Terz 84, p. 219]. The solutions are obtained as node voltages after the injection of currents, and the imposition of the depth constraints as appropriate voltages. The analog computation is supposed to provide speedier execution but limited precision compared to the digital computation. In our research, we have shown how the computation is done in a computationally efficient way using a parallel digital computer. It may be simply a temporary stop gap measure, and eventually be superceded by an analog computation mechanism.

[80]For a speedier chip, gallium arsenide potentially offers far greater (five to seven times faster) speed than silicon. In addition to speed, it dissipates less heat, so chips can be packed closer together. And it can emit light, allowing for its use in optical fiber communication systems. The technology for processing gallium arsenide is still far behind that available for silicon, but it has the best chance to replace silicon in applications that require speed.

I. The Appendix

I.1 Algorithm 6-4.1.

I.1.1 Listing of Algorithm 6-4.1.

The listing of a procedure for the adaptive Chebyshev acceleration method which uses the L_2-norm (Algorithm 6-4.1 [Youn 81, p. 107]) is reproduced in Figure I-1 and further explanation of the details of this algorithm is given below. In this algorithm the adaptive parameter estimation utilizes only the L_2-norm of the pseudoresidual vector δ.

The initial approximation is input as $u^{[0]}$. The trivial initial approximation, $u^{(0)} = (0\,0\,\ldots\,0)^{\mathrm{T}}$, was usually chosen.

The counter i is for the current iteration step number, while the counter p is for the degree of the Chebyshev polynomial currently being used.

We assume that the input estimate m_E for $m(G)$, the smallest eigenvalue of G, satisfies $m_E \leq m(G)$. The other algorithm (Algorithm 6-5.1 [Youn 81, p. 117]) detects when an initial estimate m_E is greater than $m(G)$ and obtains a new estimate for $m(G)$ if needed. We have given the discussion for numerically getting estimate m_E in section 5.2.

The initial estimate M_E for $M(G)$, the largest eigenvalue of G, should satisfy $m_E \leq M_E < 1.0$. When $m_E < 0.0$ and it is known that $M(G) > 0.0$, then $M_E = 0.0$ is appropriate if no better choice is available.

As discussed in section 4.1.3, the average virtual rate of convergence for the Chebyshev acceleration method increases to an asymptotic value and many iterations are often required before the asymptotic state is reached. Thus if M_E is changed too frequently, the optimum asymptotic convergence rate will never be achieved. A damping factor F is used to prevent M_E from being changed too often. In the following parameter change test, the current estimate M_E is judged to be unsatisfactory if

$$\|\delta^{(i)}\|_2 \;/\; \|\delta^{(q)}\|_2 \;>\; [2\,r^{p/2} \,/\, (1 + r^p)]^F, \tag{70}$$

with r being defined by

$$r = (1 - \sqrt{1 - \sigma_E^2}) / (1 + \sqrt{1 - \sigma_E^2}), \tag{71}$$

and q denoting the last iteration step at which the previous estimate M_E was used, i.e., $q = i - p$.

The constant F is a strategy parameter[81] which is chosen in the range 0-1. Choosing $F = 1$ may result in changing parameters very frequently. On the other hand, with $F = 0$ one would never change parameters no matter what value of $M_E < 1.0$ is chosen. Numerical studies indicate that F in the range 0.65-0.85 is appropriate but that the effectiveness of the adaptive Chebyshev acceleration method is relatively insensitive to F. By choosing $F < 1$, we are in effect resigning ourselves to an average convergence rate which may be only F times the optimum attainable [Youn 81, p. 66].

A new estimate M_E' is obtained as the largest real λ that satisfies

$$T_p(g_E(\lambda)) / T_p(g_E(1)) = \|\delta^{(i)}\|_2 \; / \; \|\delta^{(q)}\|_2,$$

where

$$g_E(\lambda) = (2\lambda - M_E - m_E) / (M_E - m_E).$$

Let

$$B = \|\delta^{(i)}\|_2 \; / \; \|\delta^{(q)}\|_2$$

and

$$Q = 1 / T_p(g_E(1)) = 2 r^{p/2} / (1 + r^p).$$

If the condition, $m_E < M_E < M(G) < 1.0$ and $m_E \leq m(G)$, is satisfied and if $B > Q$, then the new estimate M_E' is given by

$$M_E' = \frac{M_E + m_E}{2} + \frac{(2 - M_E - m_E)}{2(1 + r)} \frac{(X^2 + r)}{X},$$

where

$$X = [((1 + r^p) / 2)(B + \sqrt{B^2 - Q^2})]^{1/p}.$$

The new estimate M_E' obtained may be greater than $M(G)$. Two precautionary steps are taken in Algorithm 6-4.1 to ensure that all estimates M_E used are less than or equal to $M(G)$. First is the requirement that each Chebyshev polynomial generated to be at least of degree p^* before the

---

[81]By strategy parameter, it is meant that no mathematical basis exists for choosing this parameter and that the optimum parameter value is likely to be problem dependent. Usually, the effectiveness of the process is relatively insensitive to the value chosen for a strategy parameter [Youn 81, p. 66].

estimate M_E can be changed. The strategy is to pick p^* to be the smallest integer greater than 5 that satisfies

$$r^{p^*} \le d,$$

or, equivalently,

$$p^* \ge (\log d) \;/\; (\log r).$$

The constant d is a strategy parameter lying in the range 0-1. Numerical studies indicate that d in the range 0.03-0.15 is appropriate. We note that p^* could equivalently be defined as the smallest integer greater than 5 such that the ratio of average to asymptotic convergence rates is greater than some constant, say $\bar{d}$. A value of $\bar{d}$ in the range 0.6-0.42 is equivalent to d in the range 0.03-0.15. For instance, $\bar{d} = 0.481$ is equivalent to $d = 0.1$ [Youn 81, p. 105].

The second precautionary step is that of imposing upper bounds on the M_E estimates. Specifically, if $s - 1$ Chebyshev polynomials have been generated, then for polynomial s, we require that $M_E \le \tau_s$, where the strategy parameters τ_s are chosen to be [Youn 81, p. 109]

$$\begin{aligned} &\tau_1 = 0.948, \quad \tau_2 = 0.985, \quad \tau_3 = 0.995, \\ &\tau_4 = 0.9975, \quad \tau_5 = 0.9990, \quad \tau_6 = 0.9995, \\ &\tau_s = 0.99995 \ \text{ for } s \ge 7. \end{aligned} \tag{72}$$

For the termination test, we use

$$\frac{1}{1 - M_E'} \frac{\|\delta^{(i)}\|_\infty}{\|u^{(i+1)}\|_\infty} \le \zeta, \tag{73}$$

where ζ is the stopping criterion number and M_E' is the best available estimate for $M(G)$. $\zeta = 10^{-6}$ was often used in [Youn 81].

I.1.2 Overlapped Execution

The computations in Algorithm 6-4.1 can be broken down into three big parts: *Next_Iteration*, *Calculate_New_Iterate*, and the rest.

All the SIMD computations are handled in *Calculate_New_Iterate* part. Calculations of δ, $u^{[b]}$ are strictly necessary for next iterate, while $\|\delta\|_2$, $\|\delta\|_\infty$, and $\|u^{[b]}\|_\infty$ are needed for possible parameter change or iteration termination. In the calculation of $\|\delta\|_2 \equiv \sqrt{(\delta, \delta)}$, the SIMD part computes (δ, δ) and the host computes square root of it. Since $||\delta||_2$ is needed for parameter change test, the calculation on the host part can be done in background.[82]

In *Next_Iteration* part, as long as there is no parameter change, i.e., $p \geq 1$, the involved calculation on the host is trivial. Therefore, the computation time is dominated by the SIMD part and the overlapped execution of SIMD and host processor is simple. The only precaution needed is providing spaces for enough copies of u vector and corresponding scalar pointers (a, b, c, d, $\ldots$) so that we can back up. This precaution is justified if parameter change and iteration termination test performed on the host processor takes longer time than the calculations delegated to the SIMD hardware.

*Calculate_New_Estimate_*M_E', *Convergence_Test*, *Parameter_Change_Test*, and the part of *Next_Iteration* for the case of $p = 0$ constitute time-consuming scalar processing on the host processor. The computation depends on some results from the SIMD part: the value of (δ, δ) which is provided to compute $||\delta||_2$, and the values of $||\delta||_\infty$ and $||u^{[b]}||_\infty$ as well. It also needs other scalar variables such as r, p, m_E, and M_E.

Normally, the *Next_Iteration* and *Calculate_New_Iterate* parts proceed at full speed assuming that there is no need of parameter change. When it turns out that parameter change is necessary, the case of $p = 0$ in *Next_Iteration* is started and the SIMD part is set idle and the pointers are set back to the correct u vectors. This action is quite similar to the preemption in conventional pipeline processing where computational results that are done in advance but no longer relevant are thrown out. When the convergence is reached, similar action takes place for the SIMD part.

In the conjugate gradient method, this kind of sophisticated overlapped execution is unnecessary because the scalar processing on the host is simple. [See the equations (56) and (60).]

[82]If this computation, computing the square root of some value, needs to be speeded up, it may be dispatched to a special floating point arithmetic hardware unit.

DATA DEFINITION PART

Real MATRIX:

G;

Real VECTOR:

k;

$u^{[0]}$, $u^{[1]}$;

δ;

Integer SCALAR:

i;
p, p^*;

s, s_{init};

a, b, c;

Real SCALAR:

ζ;

F;
d;

m_E;
M_E, M_E';
τ_s;

γ, σ_E, ρ;
r;

DELNP, *DELNPI*;
DELNE;
YUN;

Q, B;
X;

PROGRAM PART

```
Input: ($\zeta$, $M_E$, $m_E$, $u^{[0]}$)

Initialize:

    If $M_E \le 0.948$, then
        $s_{init} = 0$;
    elseif $M_E \le 0.985$, then
        $s_{init} = 1$;
    elseif $M_E \le 0.995$, then
        $s_{init} = 2$;
    elseif $M_E \le 0.9975$, then
        $s_{init} = 3$;
    elseif $M_E \le 0.999$, then
        $s_{init} = 4$;
    elseif $M_E \le 0.9995$, then
        $s_{init} = 5$;
    else
        $s_{init} = 6$;[83]

    $F = 0.75$;
    $d = 0.1$;[84]

    $u^{[1]} = (0\,0\,\ldots\,0)^T$;

    $i = 0$;
    $p = -1$;

    $M_E' = M_E$;
    $s = s_{init}$;

    $a = 0$; $b = 1$;
```

[83]In order to use the large initial M_E estimates unmodified as input, while satisfying the imposition of upper bounds set by the equation (72), we introduced the variable s_{init} to Algorithm 6-4.1. For instance, in section 5.3.1.1, when the adaptive Chebyshev acceleration method was run with more accurate initial estimates, we used .99, .993, and .997, respectively, as the initial M_E estimates.

[84]$F = 0.75$ and $d = 0.1$ were often used, for example, in [Youn 81, p. 123].

Next Iteration:

```
$i = i + 1$;
$p = p + 1$;

If $p = 0$, then
        Begin
        $s = s + 1$;

        If $M_E' > \tau_s$, then $M_E' = \tau_s$;

        $M_E = M_E'$;

        $\rho = 1.0$;
        $\gamma = 2 / (2 - M_E - m_E)$;
        $\sigma_E = (M_E - m_E) / (2 - M_E - m_E)$;
        $r = (1 - \sqrt{1 - \sigma_E^2}) / (1 + \sqrt{1 - \sigma_E^2})$;

        $p^* = [(\log d) / (\log r)]$;
        If $p^* < 6$, then $p^* = 6$;
        End
        else
        Begin
        If $p = 1$, then $\rho = 1 / (1 - .5\sigma_E^2)$;

                else $\rho = 1 / (1 - .25\sigma_E^2\rho)$;
        End
```

Calculate New Iterate:

```
$\delta = Gu^{[a]} + k - u^{[a]}$;
$DELNP = \|\delta\|_2$; $DELNE = \|\delta\|_\infty$;

$u^{[b]} = \rho(\gamma\delta + u^{[a]}) + (1 - \rho)u^{[b]}$;
$YUN = \|u^{[b]}\|_\infty$;

$c = a$; $a = b$; $b = c$;
```

Calculate New Estimate M_E':

```
If $p \leq 2$, then
        Begin
        If $p = 0$, then $DELNPI = DELNP$;
        Go to Next Iteration;
        End
        else
        Begin
        $B = DELNP \,/\, DELNPI$;
        $Q = 2 r^{p/2} / (1 + r^p)$;

        If $B \geq 1.0$, then
                Begin
                Go to Next Iteration;
                End

        If $B > Q$, then
                Begin
                $X = [((1 + r^p)/2)(B + \sqrt{B^2 - Q^2})]^{1/p}$
                $M_E' = \frac{M_E + m_E}{2} + \frac{(2 - M_E - m_E)}{2(1+r)}\frac{(X^2 + r)}{X}$;
                End
                else $M_E' = M_E$;
        End
```

Convergence Test:

```
If $(DELNE \,/\, YUN) \leq \zeta(1 - M_E')$, then STOP(converged);
```

Parameter Change Test:

```
If $p \geq p^*$, then
        Begin
        If $B > Q^F$, then $p = -1$;
        End

Go to Next Iteration;
```

Figure I-1: Complete Listing of Algorithm 6-4.1[85]

[85]taken from [Youn 81, p. 107-109]

I.2 Listing of SIMD Programs

The lower level details of the parallel implementation under SIMD framework is given here. Particularly, we discuss the preparation of the system matrix at the pre-computation stage and the matrix-vector multiplication operation at the iteration stage.[86]

Our program is constructed out of a layer of procedures where the bottom layer consists of a set of instructions and operations. Our implementation work is based on a simulator of a particular SIMD architecture called NON-VON, whose short description was given in section 3.1.2. In Figure I-2, we have a partial summary of the semantics for the instructions and operations defined in the extended NON-VON simulator. We have shown only a part of definitions just needed for illustrating the SIMD program listings presented in this section. The complete description of the semantics can be found in [Shaw 82], [Shaw 84a], and [Choi 85a]. In general, the instructions and operations are executed on enabled PEs only with a single exception of the enable instruction (NVENBL) which enables every PE again regardless of its previous state of the enable flag (EN1). For the mesh or tree communication instructions, the semantics is defined as follows: each data item is delivered to enabled receiving PEs only, regardless of the states of sending PEs.

In NON-VON, we have five 1-bit flags (A1, B1, C1, IO1, and EN1), five 8-bit registers (A8, B8, C8, IO8, and MAR), a 1-bit memory (MEMORY1), and an 8-bit memory (MEMORY8). In the extended simulator, we conveniently introduced four floating-point registers (AF, BF, CF, and IOF) and floating-point memory (MEMORYF). For the instructions, we have equivalent ones operating on the floating-point registers and memory. For example, we have 1-bit and 8-bit *broadcast* instructions which load the designated register (or directly a memory cell pointed to by contents of its MAR) of the enabled PEs with the data item broadcasted from the host processor. To perform this function we have introduced the floating-point version of this instruction. [Compare the semantics of the instructions NVBC1, NVBC8, and NVBCF in Figure I-2.]

In Figure I-3, we show how the data in each SIMD PE are organized for our sample program. In

[86]The implementation detail of the computation of global informations will not be given. The higher level description seems to be adequate. [See the discussion of the global summation algorithm in section 3.2.3, the computation of the vector norms in section 4.1.5.2, and the computation of the inner products in section 4.2.3.2.]

```
Operands :
    src1, dst1 = {$A1, $B1, $C1, $IO1, $EN1}
    src8, dst8 = {$A8, $B8, $C8, $IO8, $MAR}
    srcf, dstf = {$AF, $BF, $CF, $IOF}
    rs1 = {$A1, $B1, $C1, $EN1}
    rsf = {$AF, $BF, $CF}
    nbr = {$E, $W, $N, $S}
    rtype = {$RC, $LC, $P, $RN, $LN}


NON-VON Instructions :
    NVENBL()                      EN1 of all PEs := 1

    NVBCR1(bit)                   MEMORY1[(MAR)] := bit
    NVBCR8(byte)                  MEMORY8[(MAR)] := byte
    NVBC1(dst1, bit)              dst1 := bit
    NVBC8(dst8, byte)             dst8 := byte

    NVRRM1(dst1)                  dst1 := (MEMORY1[(MAR)])
    NVRRM8(dst8)                  dst8 := (MEMORY8[(MAR)])
    NVWRM1(src1)                  MEMORY1[(MAR)] := (src1)
    NVWRM8(src8)                  MEMORY8[(MAR)] := (src8)

    NVMOV1(src1, dst1)            dst1 := (src1)
    NVMOV8(src8, dst8)            dst8 := (src8)

    NVAND1()                      C1 := (A1) AND (B1)

    NVMSH1(nbr)                   B1 of nbr PE := (B1)

    NVRCV1(rs1, rtype)            rs1 := (IO1) of rtype PE


Floating Point Extension of NON-VON Instructions :
    NVBCRF(ft)                    MEMORYF[(MAR)] := ft
    NVBCF(dstf, ft)               dstf := ft

    NVRRMF(dstf)                  dstf := (MEMORYF[(MAR)])
    NVWRMF(srcf)                  MEMORYF[(MAR)] := (srcf)

    NVMOVF(srcf, dstf)            dstf := (srcf)

    NVMSHF(nbr)                   BF of nbr PE := (BF)

    NVRCVF(rsf, rtype)            rsf := (IOF) of rtype PE


Basic Floating Point Operations :
    FAUC()                        CF := (CF) + (BF)
    FSUC()                        CF := (CF) - (BF)
    FMAUC()                       CF := (CF) + (AF) * (BF)
    FMSUC()                       CF := (CF) - (AF) * (BF)
```

Figure I-2: Part of NON-VON Instructions (Extension Included)

the data definition part of the listing of Algorithm 6-4.1 given in Figure I-1, we have matrix, vectors, and scalar variables. Since the scalar variables are allocated in the familiar manner in the host processor which is a conventional serial machine, our attention is focused on the SIMD implementation of data structures, i.e., the matrix and vectors.

The data in the SIMD part are allocated for every PE at the fixed same memory addresses. They are of two kinds, 1-bit flags and floating-point numbers.

The data in Figure I-3 form three groups. The first group is the input information provided to the depth interpolation process: depth discontinuities, depth constraints, orientation discontinuities, and orientation constraints.[87]

Given these inputs, the pre-computation stage computes the system matrix A and the vector b using the series of the computational molecules. [See the discussion about the derivation of the system matrix in section 2.2.2.3 and the parallelization in section 4.1.5.] The second group provides the space for the coefficients of the system matrix A and the vector b thus generated. Note that in the depth interpolation problem we have a sparse matrix, i.e., even an interior node interacts with 12 neighbors as shown in Figure 2-4 and a boundary node at a corner with only 5 neighbors as shown in Figure 2-5. As typical in the organization of SIMD data, the data spaces (in our problem, the matrix coefficients) are allocated to cover all possible configurations (in our case, an interior node turns out to be the most general configuration). The 13 matrix coefficients, one for itself and 12 for neighboring PEs, are stored at designated addresses according to the spatial arrangement and the corresponding flags are marked.[88] Usually, a flag and a number are allocated in a pair, if the number is conditionally assigned. For instance, for each depth constrained node, the flag is marked first and then the associated depth constraint is stored.

The third group of the addresses are allocated for the vectors which are repeatedly updated for every iteration step at the iteration stage. These vectors include the depth vector.

[87]For the sake of simplicity, we assumed that for every orientation-constrained node both p and q constraints exist.

[88]In the general case, for instance, when both the depth and the orientation constraints exist, one more pre-computation step may be necessary. For every depth continuous PE, all other terms are divided by the diagonal element, i.e., by the number at $FM0M0.

```
C  Continuity/constraints :

   PARAMETER($1DDSC=1, C 1 if depth is continuous.
  *          $FSYND=1) C contains synthetic depth.

   PARAMETER($1DCS=4,  C 1 if depth is constrained.
  *          $FDCS=4)  C contains depth constraint.

   PARAMETER($1OCS=2,  C 1 if orientation is constrained.
  *          $FPCS=20, C contains p constraint.
  *          $FQCS=21) C contains q constraint.


C   System matrix, vector :

                       C  coefficients of matrix A (G).
   PARAMETER($1M0P2=5,$FM0P2=5)     C for node [0,2]
   PARAMETER($1N1P1=6,$FN1P1=6)     C for node [-1,1]
   PARAMETER($1M0P1=7,$FM0P1=7)     C for node [0,1]
   PARAMETER($1P1P1=8,$FP1P1=8)     C for node [1,1]
   PARAMETER($1N2M0=9,$FN2M0=9)     C for node [-2,0]
   PARAMETER($1N1M0=10,$FN1M0=10)   C for node [-1,0]
   PARAMETER($1M0M0=11,$FM0M0=11)   C for node [0,0]
   PARAMETER($1P1M0=12,$FP1M0=12)   C for node [1,0]
   PARAMETER($1P2M0=13,$FP2M0=13)   C for node [2,0]
   PARAMETER($1N1N1=14,$FN1N1=14)   C for node [-1,-1]
   PARAMETER($1M0N1=15,$FM0N1=15)   C for node [0,-1]
   PARAMETER($1P1N1=0,$FP1N1=0)     C for node [1,-1]
   PARAMETER($1M0N2=3,$FM0N2=3)     C for node [0,-2]

   PARAMETER($FF=24)   C  vector b (k).


C   Computed vectors

   PARAMETER($FUCA=18, C  depth vector u[0].
  *          $FUCB=19) C  depth vector u[1].
   PARAMETER($FDLT=2)  C  pseudoresidual vector δ.
```

Figure I-3: Address Map (Adaptive Chebyshev Acceleration Method)

In Figure I-4, we have a SIMD listing of one of the procedures from the pre-computation stage that compute the coefficients of the system matrix A and the vector b. This procedure computes the contributions made by the upper left plate molecule in Figure 2-1 and the upper left orientation constraint molecule in Figure 2-3, i.e., if three consecutive PEs in a row are depth continuous when they are seen from the rightmost PE.

The declarations of global constants (either defined by simulator or user program) and arrays (simulator defined) come first. With the size of the tree set here, the size of the mesh at the leaf level is 128×128. The global arrays for the registers and memory of PEs are implemented with labeled COMMON statement of FORTRAN.

When computation begins, it starts with a plate molecule. All depth continuous PEs whose two neighbors on the left side are also depth continuous need to be selected. For the depth continuous PEs their contents of IO1 were set to 1 before this procedure was called. This *pre-condition* is declared at the header of the procedure. The sequence of conditions are declared as comments. [Recall the general description about conditions in section 5.5.] Note the sequence of actions leading to the selection of the desired set of nodes. First, all PEs are enabled. Second, we are only interested in the marked set of the pre-condition. Third, the flag from west neighbor is received and ANDed with its own flag to select all pair of PEs which are depth continuous. Finally, the flag from west neighbor is received again and ANDed to select all three consecutive depth continuous PEs in a row. Note that the flag of every PE is transmitted by the execution of each mesh communication instruction since every PE is enabled up to the final selection time. Because the flag was received twice, the one received at the most recent step is the one from two neighbors away.

For the selected set, the three coefficients, [1, -2, and 1],[89] are added to at three addresses ($FN2M0, $FN1M0, and $FM0M0) after the flags are marked.

When the orientation constraint molecule is applied, the more restricted set is selected. Three consecutive PEs in a row should be depth continuous and the left neighbor of the rightmost PE be

[89]Compared to the nodal equation (3), $1 / h^2$ is factored out. [See the Figure 2-4.]

p orientation constrained as well. The desired condition is generated as follows: the orientation constraint flag read at left neighbor is shifted in to be ANDed together with the retained condition of the three consecutive depth continuous nodes.

The three coefficients, $[2p_{-1,0}, -1/h,$ and $1/h]$,[90] are added to at three addresses ($FF, $FN2M0, and $FM0M0) where $p_{-1,0}$ is the p constraint of the left neighbor.

In Figure I-5, we have the SIMD listing of the matrix-vector multiplication operation performed at each iteration step. Note the declaration of all four conditions, i.e., *input*, *output*, *pre-condition*, and *post-condition*, at the header. The operation performed at every depth continuous PE is a rather simple one. After checking whether this particular term is present or not in the nodal equation of this PE, each element value shifted-in from its neighbor is multiplied by the appropriate coefficient and added to the intermediate sum. In the listing, we show two such sequences. The first one gets the value from its west neighbor and multiplies it by the coefficient at $FN1M0. The second one gets the value from the west neighbor of its west neighbor and multiplies it by the coefficient at $FN2M0. Observe again the pipeline-like shift-in of data. When the PE sends data to its east neighbor, it gets data from its west neighbor and at next shift, the PE gets data from two neighbors away.

[90]We assumed that $\alpha^h = \gamma_p / h$, where γ_p is 4.0. [See the nodal equation (3).]

```
C Pre : { For depth continuous PEs at chosen level,
C         their (IO1) = 1.                              }
C
C ( 1, -2, [1] )

      SUBROUTINE FPHWWM(HINV)

      INTEGER $MINPE
      PARAMETER ($MINPE = 1)
      INTEGER $MAXPE
      PARAMETER ($MAXPE = 32767)

      INTEGER $E, $W, $N, $S
      PARAMETER ($E = 1, $W = 2, $N = 3, $S = 4)

      INTEGER*2 $EN1($MINPE:$MAXPE),
     *           $A1($MINPE:$MAXPE),
     *           $B1($MINPE:$MAXPE),
     *           $C1($MINPE:$MAXPE),
     *          $IO1($MINPE:$MAXPE)
      COMMON /NV1/ $EN1, $A1, $B1, $C1, $IO1

      INTEGER*2 $MAR($MINPE:$MAXPE),
     *           $A8($MINPE:$MAXPE),
     *           $B8($MINPE:$MAXPE),
     *           $C8($MINPE:$MAXPE),
     *          $IO8($MINPE:$MAXPE)
      COMMON /NV8/ $MAR, $A8, $B8, $C8, $IO8

      REAL*8   $AF($MINPE:$MAXPE),
     *         $BF($MINPE:$MAXPE),
     *         $CF($MINPE:$MAXPE),
     *        $IOF($MINPE:$MAXPE)
      COMMON /NVF/ $AF, $BF, $CF, $IOF

      INTEGER $1N2M0, $FN2M0
      PARAMETER ($1N2M0 = 9, $FN2M0 = 9)
      INTEGER $1N1M0, $FN1M0
      PARAMETER ($1N1M0 = 10, $FN1M0 = 10)
      INTEGER $1M0M0, $FM0M0
      PARAMETER ($1M0M0 = 11, $FM0M0 = 11)
      INTEGER $1OCS, $FPCS, $FQCS
      PARAMETER ($1OCS = 2, $FPCS = 20, $FQCS = 21)
      INTEGER $FF
      PARAMETER ($FF = 24)

      REAL*8 HINV
      CALL NVENBL()

      CALL NVMOV1($IO1, $A1)
C { (A1) = 1 if this PE is depth continuous. }

      CALL NVMOV1($IO1, $B1)
C { (B1) = 1 if this PE is depth continuous. }
      CALL NVMSH1($E)
C { (B1) = 1 if west neighbor PE is depth continuous. }
      CALL NVAND1()
      CALL NVMOV1($C1, $A1)
C { (A1) = 1 if both this PE and
C   its west neighbor PE are depth continuous. }

      CALL NVMSH1($E)
      CALL NVAND1()
      CALL NVMOV1($C1, $EN1)
C { Only those PEs are enabled
C   where three contiguous horizontal PEs
C   seen from rightmost PE are depth continuous. }

C   rel-x = -2 and rel-y = 0.

      CALL NVBC8($MAR, $1N2M0)
      CALL NVBCR1(1)
      CALL NVBCF($BF, 1.0D0)
      CALL NVRRMF($CF)
      CALL FAUC()
      CALL NVWRMF($CF)

C   rel-x = -1 and rel-y = 0.

      CALL NVBC8($MAR, $1N1M0)
      CALL NVBCR1(1)
      CALL NVBCF($BF, 2.0D0)
      CALL NVRRMF($CF)
      CALL FAUC()
      CALL NVWRMF($CF)

C   rel-x = 0 and rel-y = 0.

      CALL NVBC8($MAR, $1M0M0)
      CALL NVBCR1(1)
      CALL NVBCF($BF, 1.0D0)
      CALL NVRRMF($CF)
      CALL FAUC()
      CALL NVWRMF($CF)
```

```
C   Orientation constraint.

      CALL NVENBL()
      CALL NVBC8($MAR, $1OCS)
      CALL NVRRM1($B1)

      CALL NVBC8($MAR, $FPCS)
      CALL NVRRMF($BF)

      CALL NVMSH1($E)
      CALL NVMSHF($E)

C { (B1) = 1
C   if west neighbor PE is orientation constrained. }
C { BF contains p orientation constraint
C   of west neighbor PE.                           }

      CALL NVMOV1($C1, $A1)
      CALL NVAND1()
      CALL NVMOV1($C1, $EN1)

C { Only those PEs are enabled where three contiguous
C   horizontal PEs seen from rightmost PE are depth
C   continuous and left neighbor of rightmost PE is
C   orientation constrained.                           }

C   b vector.

      CALL NVBC8($MAR, $FF)
      CALL NVRRMF($CF)
      CALL NVBCF($AF, 2.0D0)
      CALL FMAUC()
      CALL NVWRMF($CF)

C   rel-x = -2 and rel-y = 0.

      CALL NVBC8($MAR, $FN2M0)
      CALL NVRRMF($CF)
      CALL NVBCF($BF, HINV)
      CALL FSUC()
      CALL NVWRMF($CF)

C   rel-x = 0 and rel-y = 0.

      CALL NVBC8($MAR, $FM0M0)
      CALL NVRRMF($CF)
      CALL NVBCF($BF, HINV)
      CALL FAUC()
      CALL NVWRMF($CF)

      RETURN
      END
```

Figure I-4: Pre-computation Stage (Computation of Matrix Coefficients)

```
C PRE  : { For every depth continuous PE at chosen level,
C            (IO1) = 1. }
C PRE  : { Depth continuous PEs at chosen level
C           are enabled. }
C IN   : { For every depth continuous PE at chosen level,
C           IOF contains current value of the vector v.  }
C POST : { Depth continuous PEs at chosen level
C           are enabled. }
C OUT  : { For every depth continuous PE at chosen level,
C           element value of the vector Gv is left in CF.}

      SUBROUTINE FCGVCR()


      ...


C { For every depth continuous PE at chosen level,
C   (IO1) = 1, (EN1) = 1, i.e., enabled, and
C   IOF contains its own depth value. }

C   Clear CF.
      CALL NVBCF($CF, 0.0D0)


C   Two horizontal left terms.

C { Depth continuous PEs at chosen level are enabled. }
C { CF contains intermediate sum. }

C   rel-x = -1, rel-y = 0

      CALL NVMOVF($IOF, $BF)
      CALL NVMSHF($E)
C { BF contains depth value of west neighbor PE. }

      CALL NVBC8($MAR, $1N1M0)
      CALL NVRRM1($EN1)
      CALL NVRRMF($AF)
C { AF contains locally contributed multiplication factor
C   for west neighbor PE. }

C   CF := (CF) + (AF) * (BF)
      CALL FMAUC()
C { CF contains updated intermediate sum. }
```

```
C   rel-x = -2, rel-y = 0

      CALL NVENBL()
      CALL NVMOV1($IO1, $EN1)
C { All depth continuous PEs at chosen level
C   are enabled again. }

      CALL NVMSHF($E)
C { BF now contains current depth value of
C   west neighbor of west neighbor PE. }

      CALL NVBC8($MAR, $1N2M0)
      CALL NVRRM1($EN1)
      CALL NVRRMF($AF)
C { AF contains locally contributed multiplication factor
C   for west neighbor of west neighbor PE. }

C   CF := (CF) - (AF) * (BF)
      CALL FMSUC()
C { CF contains updated intermediate sum. }


      ...


      RETURN
      END
```

Figure I-5: Iteration Stage (Computation of Matrix-Vector Multiplication)

I.3 Supplementary Numerical Results

I.3.1 Other Numerical Values

We show other numerical values of single-grid algorithm execution result for the plane example discussed in section 5.3.1.1.

In Table I-1 we have the results from each iterative method for three different densities of the depth constraints. In Table I-2 we have shown another result from the adaptive Chebyshev acceleration method when more accurate initial estimates were used. The measures listed are the number of iterations; the minimum, the maximum, and the average of the depth values; and the L_2-norms of the error vector. Furthermore, we have listed $\|A^{1/2}(x^{(i)} - \alpha)\|$ for the conjugate gradient method and the final estimate of the largest eigenvalue, M_E, for the adaptive Chebyshev acceleration method.

When we examine the final depth values, they show another aspect of the depth interpolation problem becoming harder as the depth constraints become sparser. With comparable or sometimes better average values, the minimum and the maximum values deviate further from the solution. This phenomenon is common to all three iterative methods. For example, for the conjugate gradient method, we have smaller values of $\|A^{1/2}(x^{(i)} - \alpha)\|$, the quantity being minimized in this method, as the depth constraints become sparser. Nevertheless, we have comparable L_2-norms of the error vector and the average values, and worse minimum and maximum values.

Density of the Depth Constraints = 50%

Results from the Conjugate Gradient Method

i	$x^{(i)}_{min}$	$x^{(i)}_{max}$	$x^{(i)}_{avg}$	$\|\|x^{(i)} - \alpha\|\|$	$\|\|A^{1/2}(x^{(i)} - \alpha)\|\|$
21	.457691	1.250388	.969833	12.433526	1.5220272
46	.899396	1.024408	.999615	1.229187	.1426707
71	.984603	1.002782	.999994	.126194	.0143727
96	.998077	1.000396	.999999	.013349	.0014978

Results from the Adaptive Chebyshev Acceleration Method (*initial* $m_E = -3.0, M_E = 0.0$)

i	$x^{(i)}_{min}$	$x^{(i)}_{max}$	$x^{(i)}_{avg}$	$\|\|x^{(i)} - \alpha\|\|$	M_E
56	.574881	1.027588	.915067	12.906060	.988772
95	.907512	1.005085	.994306	1.296641	.990385
136	.985364	1.000801	.999687	.124797	.991694
170	.997973	1.000107	.999982	.012647	.991694

Results from the Gauss-Seidel Method

i	$x^{(i)}_{min}$	$x^{(i)}_{max}$	$x^{(i)}_{avg}$	$\|\|x^{(i)} - \alpha\|\|$
107	.637535	1.034809	.910398	12.746128
229	.927154	1.003870	.992740	1.291107
365	.988639	1.000352	.999474	.127688
509	.998467	1.000085	.999962	.012823

(Continued)

Density of the Depth Constraints = 30%

Results from the Conjugate Gradient Method

i	$x^{(i)}_{min}$	$x^{(i)}_{max}$	$x^{(i)}_{avg}$	$\|\|x^{(i)} - \alpha\|\|$	$\|\|A^{1/2}(x^{(i)} - \alpha)\|\|$
32	.361134	1.213464	.970978	12.562866	1.0888625
70	.844512	1.019000	.999614	1.268632	.0957004
110	.980571	1.003009	.999988	.130898	.0096369
146	.998821	1.000854	1.000000	.010304	.0010545

Results from the Adaptive Chebyshev Acceleration Method (*initial* $m_E = -3.0, M_E = 0.0$)

i	$x^{(i)}_{min}$	$x^{(i)}_{max}$	$x^{(i)}_{avg}$	$\|\|x^{(i)} - \alpha\|\|$	M_E
80	.453362	1.038548	.922690	12.645528	.993695
137	.855935	1.005517	.995953	1.284419	.994744
208	.979400	1.000410	.999826	.126349	.996296
271	.997705	1.000047	.999991	.012930	.996805

Results from the Gauss-Seidel Method

i	$x^{(i)}_{min}$	$x^{(i)}_{max}$	$x^{(i)}_{avg}$	$\|\|x^{(i)} - \alpha\|\|$
196	.510884	1.026782	.915904	12.750857
445	.879459	1.003267	.994288	1.279349
752	.981034	1.000412	.999687	.128378
1103	.997807	1.000048	.999983	.012805

(Continued)

Density of the Depth Constraints = 15%

Results from the Conjugate Gradient Method

i	$x^{(i)}_{min}$	$x^{(i)}_{max}$	$x^{(i)}_{avg}$	$\|x^{(i)} - \alpha\|$	$\|A^{1/2}(x^{(i)} - \alpha)\|$
55	.567370	1.171957	.973058	12.643796	.6655899
130	.755494	1.015766	.999641	1.282472	.0487649
200	.979287	1.007600	.999997	.125836	.0053095
267	.998963	1.001350	1.000001	.012700	.0005662

Results from the Adaptive Chebyshev Acceleration Method (*initial* $m_E = -3.0, M_E = 0.0$)

i	$x^{(i)}_{min}$	$x^{(i)}_{max}$	$x^{(i)}_{avg}$	$\|x^{(i)} - \alpha\|$	M_E
129	.139001	1.039064	.931920	12.797598	.997510
251	.766146	1.010805	.997725	1.286709	.998322
384	.971733	1.001309	.999929	.127170	.998905
495	.997079	1.000135	.999993	.012906	.999064

Results from the Gauss-Seidel Method

i	$x^{(i)}_{min}$	$x^{(i)}_{max}$	$x^{(i)}_{avg}$	$\|x^{(i)} - \alpha\|$
460	.088604	1.030712	.925998	12.793736
1231	.766415	1.011704	.997037	1.278771
2394	.971534	1.001434	.999900	.127979
3650	.997069	1.000148	.999993	.012799

Table I-1: Other Results (plane)

Results from the Adaptive Chebyshev Acceleration Method

Density of the Depth Constraints = 50% (*initial* $m_E = -2.3$, $M_E = 0.99$)

i	$x^{(i)}_{min}$	$x^{(i)}_{max}$	$x^{(i)}_{avg}$	$\|\|x^{(i)} - \alpha\|\|$	M_E
26	.485533	1.158198	.944240	12.718857	.99
57	.890348	1.011170	.997374	1.297171	.99
97	.983999	1.000908	.999732	.127034	.99
142	.998015	1.000105	.999979	.012739	.992610

Density of the Depth Constraints = 30% (*initial* $m_E = -2.3$, $M_E = 0.993$)

i	$x^{(i)}_{min}$	$x^{(i)}_{max}$	$x^{(i)}_{avg}$	$\|\|x^{(i)} - \alpha\|\|$	M_E
45	.392290	1.093854	.934956	12.558977	.993
110	.856176	1.005725	.995760	1.300641	.995482
169	.979009	1.000427	.999845	.127135	.996110
224	.997772	1.000046	.999991	.012565	.996747

Density of the Depth Constraints = 15% (*initial* $m_E = -2.3$, $M_E = 0.997$)

i	$x^{(i)}_{min}$	$x^{(i)}_{max}$	$x^{(i)}_{avg}$	$\|\|x^{(i)} - \alpha\|\|$	M_E
70	.591026	1.079271	.947534	12.793268	.997
197	.765680	1.010865	.997963	1.268198	.998358
315	.971242	1.001331	.999939	.128831	.998937
422	.997084	1.000135	.999994	.012879	.999069

Table I-2: Other Results (plane : with more accurate initial estimates of m_E and M_E)

I.3.2 Sample Traces of Multigrid Algorithm (Fixed Scheme)

We show further details of multigrid algorithm execution result for the cylinder example discussed in section 5.3.2.1.

For the multigrid methods, we use the discrete L_2-norm of the error vector (at the nth iteration step), $\varepsilon^{(n)} \equiv x^{(n)} - \alpha$, which is defined as follows [Brig 87, p. 62]:

$$\|\varepsilon^{(n)}\|_{h_l} = (h_l \sum_{j=1}^{n_l} [x_j^{(n)} - \alpha_j]^2)^{1/2}, \qquad (74)$$

where n_l represents the number of the depth continuous nodes in the region and h_l the length of the sides of the tessellation square elements[91] at level l.

The L_2-norm of the error vector is not available in most problems and a more practical measure of convergence is the discrete L_2-norm of the residual vector, $r^{h_l} \equiv g^{h_l} - A^{h_l} x^{h_l}$, similarly defined by

$$\|r^{(n)}\|_{h_l} = (h_l \sum_{j=1}^{n_l} [g_j^{h_l} - (A^{h_l} x^{(n)})_j]^2)^{1/2}. \qquad (75)$$

Figure I-6 shows the scheduling of grids for the fixed scheme with $n_2 = 1$. [For the fixed scheme, see the program listing in Figure 4-2.]

As explained already, the value of t_0 was chosen to achieve the desired accuracy at the coarsest level. Though we did not run extensive experiments, the choice of $n_2 = 1$ and $n_1 = n_3$ seems to be simple and good enough.

The set of s_0, n_1, and n_3 were chosen as the possible, but not always, minimal values where the final RMSE value for every level is smaller than the initial RMSE value when each level is first entered. For the coarsest level, the final RMSE value is ensured to be smaller than the desired accuracy obtained after t_0 iterations. For the other levels, the final RMSE values are ensured to be smaller than the initial values generated through the prolongation process in the procedure FMRA before invoking the procedure FMC.

[91]The actual lengths of the sides from coarsest to finest respectively were $h_1 = 0.8$, $h_2 = 0.4$, $h_3 = 0.2$, and $h_4 = 0.1$.

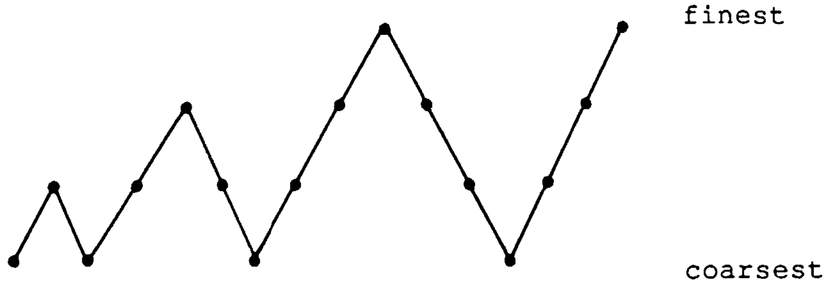

Figure I-6: Schedule of Grids on Four Levels[92]

As the first trace example of the scheduling, we show the result of the conjugate gradient method. The parameter values are $t_0 = 15$, $s_0 = 4$, $n_1 = 6$, $n_2 = 1$, and $n_3 = 6$.

We trace the change of the RMSE values as the grids are visited. We trace also the discrete L_2-norms of the residual vector but the values of the residual norms are enclosed in parentheses.

After initial 15 iterations ($n = 15$), we have the RMSE value of .1011137 (.12556313) at the coarsest level $l = 1$.

The prolongation in the procedure FMRA provides the initial guess for level 2 and the initial RMSE value at this level is .1005757 (.07040571). The procedure FMC is invoked for the first time and after 6 iterations on level 2, we have the restriction back to level 1. The reduced RMSE value at the coarsest level is .0872281 (.02906745). After 4 iterations it is reduced further to .0795454 (.03060642) and we have the prolongation to level 2. After 6 iterations on level 2, we have the RMSE value of .0727709 (.02480185) and the first call of FMC is finished.

The prolongation in the procedure FMRA provides the initial guess for level 3 and the initial RMSE value is .0731687 (.02369524). The procedure FMC is now invoked for the second time

[92]taken from [Brig 87, p. 47]

and after 6 iterations on level 3, we have the restriction back to level 2. The reduced RMSE value at level 2 is .0709065 (.01163662) and after 6 iterations on level 2, we have the restriction back to level 1. The reduced RMSE value at the coarsest level is .0685103 (.00639124). After 4 iterations it is reduced further to .0682280 (.00522123) and we have the prolongation to level 2. After 6 iterations on level 2, we have the RMSE value of .0691420 (.00539452) and we have the prolongation to level 3. After 6 iterations on level 3, we have the RMSE value of .0693508 (.00605052) and the second call of FMC is finished.

The prolongation in the procedure FMRA provides the initial guess for the finest level and the initial RMSE value is .0696286 (.00991621). The procedure FMC is now invoked for the last time and after 6 iterations on level 4, we have the restriction back to level 3. The reduced RMSE value at level 3 is .0691617 (.00425162) and after 6 iterations on level 3, we have the restriction back to level 2. The reduced RMSE value at level 2 is .0682678 (.00336418) and after 6 iterations on level 2, we have the restriction back to level 1. The reduced RMSE value at the coarsest level is .0665816 (.00173344).

Now, starting at the coarsest level, we have a final series of iterations and prolongations down to the finest level. After 4 iterations (n = 81), the RMSE value is reduced further to final .0664448 (.00191015) and we have the prolongation to level 2. After 6 iterations on level 2 (n = 87), we have the final value of .0675243 (.00228414) and we have the prolongation to level 3. After 6 iterations on level 3 (n = 93), we have the final value of .0680527 (.00363177) and we have the prolongation to level 4. After 6 iterations on the finest level (n = 99), we have the final value of .0683314 (.00244497) and the whole procedure is finished.

As the second trace example of the scheduling, we show the result of the adaptive Chebyshev acceleration method. The parameter values are $t_0 = 23$, $s_0 = 4$, $n_1 = 7$, $n_2 = 1$, and $n_3 = 7$. We trace again the change of the RMSE values and the residual norms as the grids are visited. The values of the residual norms are enclosed in parentheses.

After initial 23 iterations (n = 23), we have the RMSE value of .0955011 (.02904537) at the coarsest level $l = 1$.

The prolongation in the procedure FMRA provides the initial guess for level 2 and the initial RMSE value at this level is .0951464 (.05456750). The procedure FMC is invoked for the first time and after 7 iterations on level 2, we have the restriction back to level 1. The reduced RMSE value at the coarsest level is .0810578 (.01965193). After 4 iterations it is reduced further to .0779575 (.01120022) and we have the prolongation to level 2. After 7 iterations on level 2, we have the RMSE value of .0685260 (.01813959) and the first call of FMC is finished.

The prolongation in the procedure FMRA provides the initial guess for level 3 and the initial RMSE value is .0684786 (.01851928). The procedure FMC is now invoked for the second time and after 7 iterations on level 3, we have the restriction back to level 2. The reduced RMSE value at level 2 is .0669836 (.00918446) and after 7 iterations on level 2, we have the restriction back to level 1. The reduced RMSE value at the coarsest level is .0654301 (.00367233). After 4 iterations it is reduced further to .0647048 (.00225785) and we have the prolongation to level 2. After 7 iterations on level 2, we have the RMSE value of .0634130 (.00356676) and we have the prolongation to level 3. After 7 iterations on level 3, we have the RMSE value of .0625448 (.00975163) and the second call of FMC is finished.

The prolongation in the procedure FMRA provides the initial guess for the finest level and the initial RMSE value is .0625041 (.00758814). The procedure FMC is now invoked for the last time and after 7 iterations on level 4, we have the restriction back to level 3. The reduced RMSE value at level 3 is .0624155 (.00400121) and after 7 iterations on level 3, we have the restriction back to level 2. The reduced RMSE value at level 2 is .0624187 (.00231163) and after 7 iterations on level 2, we have the restriction back to level 1. The reduced RMSE value at the coarsest level is .0627756 (.00092782).

Now, starting at the coarsest level, we have a final series of iterations and prolongations down to the finest level. After 4 iterations (n = 98), the RMSE value is reduced further to final .0627607 (.00052839) and we have the prolongation to level 2. After 7 iterations on level 2 (n = 105), we have the final value of .0624723 (.00089367) and we have the prolongation to level 3. After 7 iterations on level 3 (n = 112), we have the final value of .0621976 (.00224931) and we have the prolongation to level 4. After 7 iterations on the finest level (n = 119), we have the final value of

.0620595 (.00436349) and the whole procedure is finished.

As the last trace example of the scheduling, we show the result of the Gauss-Seidel method. The parameter values are $t_0 = 47$, $s_0 = 6$, $n_1 = 8$, $n_2 = 1$, and $n_3 = 8$. We trace again the change of the RMSE values and the residual norms as the grids are visited. The values of the residual norms are enclosed in parentheses.

After initial 47 iterations ($n = 47$), we have the RMSE value of .0967473 (.02969730) at the coarsest level $l = 1$.

The prolongation in the procedure FMRA provides the initial guess for level 2 and the initial RMSE value at this level is .0962959 (.05760785). The procedure FMC is invoked for the first time and after 8 iterations on level 2, we have the restriction back to level 1. The reduced RMSE value at the coarsest level is .0860415 (.01137371). After 6 iterations it is reduced further to .0800077 (.00710874) and we have the prolongation to level 2. After 8 iterations on level 2, we have the RMSE value of .0729618 (.00927485) and the first call of FMC is finished.

The prolongation in the procedure FMRA provides the initial guess for level 3 and the initial RMSE value is .0727732 (.01824689). The procedure FMC is now invoked for the second time and after 8 iterations on level 3, we have the restriction back to level 2. The reduced RMSE value at level 2 is .0716791 (.00394246) and after 8 iterations on level 2, we have the restriction back to level 1. The reduced RMSE value at the coarsest level is .0710175 (.00136784). After 6 iterations it is reduced further to .0703222 (.00089113) and we have the prolongation to level 2. After 8 iterations on level 2, we have the RMSE value of .0692155 (.00125151) and we have the prolongation to level 3. After 8 iterations on level 3, we have the RMSE value of .0683705 (.00293137) and the second call of FMC is finished.

The prolongation in the procedure FMRA provides the initial guess for the finest level and the initial RMSE value is .0682978 (.00733104). The procedure FMC is now invoked for the last time and after 8 iterations on level 4, we have the restriction back to level 3. The reduced RMSE value at level 3 is .0682557 (.00150354) and after 8 iterations on level 3, we have the restriction back to level 2. The reduced RMSE value at level 2 is .0683084 (.00051254) and after 8 iterations on level 2, we have the restriction back to level 1. The reduced RMSE value at the coarsest level is .0684145 (.00017451).

Now, starting at the coarsest level, we have a final series of iterations and prolongations down to the finest level. After 6 iterations (n = 137), the RMSE value is reduced further to final .0683368 (.00010632) and we have the prolongation to level 2. After 8 iterations on level 2 (n = 145), we have the final value of .0680508 (.00015623) and we have the prolongation to level 3. After 8 iterations on level 3 (n = 153), we have the final value of .0678371 (.00040012) and we have the prolongation to level 4. After 8 iterations on the finest level (n = 161), we have the final value of .0677086 (.00105797) and the whole procedure is finished.

We show other numerical results of multigrid execution in Table I-3, I-5, and I-7 for the conjugate gradient, the adaptive Chebyshev acceleration, and the Gauss-Seidel methods, respectively. We start with the accuracy achieved at the coarsest level after initial iterations on that level. We show then the effects of the prolongation operation to level 2, iterations on that level, and the restriction operation back to the coarsest level. We show next the initial guesses at level 3 and 4 after the prolongation operations. Lastly, we show the values at final iterations on each level and the effects of the prolongation operations to adjacent finer levels.

We show also numerical results for the iterations on a single level, i.e., on the finest level only, in Table I-4 and I-6, for the conjugate gradient and the adaptive Chebyshev acceleration methods, respectively.

The measures listed are the number of iteration steps n and the level l; the discrete L_2-norms of the residual vector; the minimum, the maximum, and the average of the depth values; the minimum, the maximum, and the average of the relative depth values where the computed depth values are divided by the ideal synthetic depth values; the discrete L_2-norms of the error vector and the RMSE values. Furthermore, we have listed $||A^{1/2}(x^{(n)} - \alpha)||$ for the conjugate gradient method, the estimate of the largest eigenvalue $M_E^{h_l}$ and the L_∞-norms of the pseudoresidual vector for the adaptive Chebyshev acceleration method, and the accumulated work units for the Gauss-Seidel method.

After the initial 15 iterations on level 1 :

$n = 15,\ l = 1$
$\|r^{(n)}\|_{h_1} = .12556313$

	$x^{(n)}_{min} = .684947,$	$x^{(n)}_{max} = 1.127059,$	$x^{(n)}_{avg} = .934484$
rel.	$x^{(n)}_{min} = .704688,$	$x^{(n)}_{max} = 1.148635,$	$x^{(n)}_{avg} = .978427$

$\|\varepsilon^{(n)}\|_{h_1} = 1.447022,$ $RMSE^{(n)} = .1011137,$ $\|A^{1/2}(x^{(n)} - \alpha)\| = .2746703$

After prolongation operation $[I_{1\to 2}\, x^{h_1}]$ in FMRA :

$\|r^{(n)}\|_{h_2} = .07040571$

	$x^{(n)}_{min} = .684947,$	$x^{(n)}_{max} = 1.127059,$	$x^{(n)}_{avg} = .935062$
rel.	$x^{(n)}_{min} = .703046,$	$x^{(n)}_{max} = 1.148635,$	$x^{(n)}_{avg} = .978854$

$\|\varepsilon^{(n)}\|_{h_2} = 2.035510,$ $RMSE^{(n)} = .1005757$

After 6 iterations on level 2 :

$n = 21,\ l = 2$
$\|r^{(n)}\|_{h_2} = .04193274$

	$x^{(n)}_{min} = .687771,$	$x^{(n)}_{max} = 1.147881,$	$x^{(n)}_{avg} = .936341$
rel.	$x^{(n)}_{min} = .703572,$	$x^{(n)}_{max} = 1.166643,$	$x^{(n)}_{avg} = .980099$

$\|\varepsilon^{(n)}\|_{h_2} = 1.787390,$ $RMSE^{(n)} = .0883160,$ $\|A^{1/2}(x^{(n)} - \alpha)\| = .2316637$

After restriction operation $[I_{2\to 1}\, x^{h_2}$ and $A^{h_1} x^{h_1} + I_{2\to 1}(g^{h_2} - A^{h_2} x^{h_2})]$ in FMC :

$\|r^{(n)}\|_{h_1} = .02906745$

	$x^{(n)}_{min} = .688887,$	$x^{(n)}_{max} = 1.094860,$	$x^{(n)}_{avg} = .935296$
rel.	$x^{(n)}_{min} = .703572,$	$x^{(n)}_{max} = 1.121478,$	$x^{(n)}_{avg} = .979236$

$\|\varepsilon^{(n)}\|_{h_1} = 1.248306,$ $RMSE^{(n)} = .0872281$

After prolongation operation $[I_{2\to 3}\, x^{h_2}]$ in FMRA :

$\|r^{(n)}\|_{h_3} = .02369524$

	$x^{(n)}_{min} = .717967,$	$x^{(n)}_{max} = 1.121397,$	$x^{(n)}_{avg} = .939824$
rel.	$x^{(n)}_{min} = .741299,$	$x^{(n)}_{max} = 1.144437,$	$x^{(n)}_{avg} = .983641$

$\|\varepsilon^{(n)}\|_{h_3} = 2.094210,$ $RMSE^{(n)} = .0731687$

(Continued)

After prolongation operation $[I_{3\to 4}\, x^{h_3}]$ in FMRA :

$\|r^{(n)}\|_{h_4} = .00991621$

$x^{(n)}_{min} = .721806$, $x^{(n)}_{max} = 1.123942$, $x^{(n)}_{avg} = .939447$

rel. $x^{(n)}_{min} = .747774$, $x^{(n)}_{max} = 1.151309$, $x^{(n)}_{avg} = .983218$

$\|\varepsilon^{(n)}\|_{h_4} = 2.818367$, $RMSE^{(n)} = .0696286$

After final iteration on level 1 :

$n = 81,\ l = 1$

$\|r^{(n)}\|_{h_1} = .00191015$

$x^{(n)}_{min} = .728396$, $x^{(n)}_{max} = 1.048217$, $x^{(n)}_{avg} = .936728$

rel. $x^{(n)}_{min} = .751891$, $x^{(n)}_{max} = 1.085186$, $x^{(n)}_{avg} = .980599$

$\|\varepsilon^{(n)}\|_{h_1} = .950881$, $RMSE^{(n)} = .0664448$, $\|A^{1/2}(x^{(n)} - \alpha)\| = .2465730$

After prolongation operation $[x^{h_2} + I_{1\to 2}(x^{h_1} - I_{2\to 1}x^{h_2})]$ in FMC :

$\|r^{(n)}\|_{h_2} = .00305485$

$x^{(n)}_{min} = .726623$, $x^{(n)}_{max} = 1.080351$, $x^{(n)}_{avg} = .938187$

rel. $x^{(n)}_{min} = .751891$, $x^{(n)}_{max} = 1.146863$, $x^{(n)}_{avg} = .981922$

$\|\varepsilon^{(n)}\|_{h_2} = 1.372128$, $RMSE^{(n)} = .0677977$

After final iteration on level 2 :

$n = 87,\ l = 2$

$\|r^{(n)}\|_{h_2} = .00228414$

$x^{(n)}_{min} = .726589$, $x^{(n)}_{max} = 1.073957$, $x^{(n)}_{avg} = .938208$

rel. $x^{(n)}_{min} = .751728$, $x^{(n)}_{max} = 1.141830$, $x^{(n)}_{avg} = .981953$

$\|\varepsilon^{(n)}\|_{h_2} = 1.366596$, $RMSE^{(n)} = .0675243$, $\|A^{1/2}(x^{(n)} - \alpha)\| = .1937609$

After prolongation operation $[x^{h_3} + I_{2\to 3}(x^{h_2} - I_{3\to 2}x^{h_3})]$ in FMC :

$\|r^{(n)}\|_{h_3} = .00554816$

$x^{(n)}_{min} = .718667$, $x^{(n)}_{max} = 1.105213$, $x^{(n)}_{avg} = .938600$

rel. $x^{(n)}_{min} = .748165$, $x^{(n)}_{max} = 1.184091$, $x^{(n)}_{avg} = .982343$

$\|\varepsilon^{(n)}\|_{h_3} = 1.951834$, $RMSE^{(n)} = .0681943$

(Continued)

After final iteration on level 3 :

$n = 93,\ l = 3$
$\|r^{(n)}\|_{h_3} = .00363177$

	$x^{(n)}_{min} = .714002,$	$x^{(n)}_{max} = 1.099287,$	$x^{(n)}_{avg} = .938572$
rel.	$x^{(n)}_{min} = .748673,$	$x^{(n)}_{max} = 1.170988,$	$x^{(n)}_{avg} = .982313$

$\|\varepsilon^{(n)}\|_{h_3} = 1.947783,$ $RMSE^{(n)} = .0680527,$ $\|A^{1/2}(x^{(n)} - \alpha)\| = .1039976$

After prolongation operation $[x^{h_4} + I_{3\to4}(x^{h_3} - I_{4\to3}x^{h_4})]$ in FMC :

$\|r^{(n)}\|_{h_4} = .00713094$

	$x^{(n)}_{min} = .707536,$	$x^{(n)}_{max} = 1.113764,$	$x^{(n)}_{avg} = .938798$
rel.	$x^{(n)}_{min} = .748534,$	$x^{(n)}_{max} = 1.184069,$	$x^{(n)}_{avg} = .982558$

$\|\varepsilon^{(n)}\|_{h_4} = 2.767711,$ $RMSE^{(n)} = .0683771$

After final iteration on level 4 :

$n = 99,\ l = 4$
$\|r^{(n)}\|_{h_4} = .00244497$

	$x^{(n)}_{min} = .704425,$	$x^{(n)}_{max} = 1.112690,$	$x^{(n)}_{avg} = .938812$
rel.	$x^{(n)}_{min} = .748437,$	$x^{(n)}_{max} = 1.185867,$	$x^{(n)}_{avg} = .982572$

$\|\varepsilon^{(n)}\|_{h_4} = 2.765860,$ $RMSE^{(n)} = .0683314,$ $\|A^{1/2}(x^{(n)} - \alpha)\| = .0562665$

Table I-3: Trace of Multigrid Algorithm (Conjugate Gradient Method)

After 683 iterations (on level 4) :

$n = 683,\ l = 4$
$\|r^{(n)}\|_{h_4} = .00075454$

	$x^{(n)}_{min} = .737644,$	$x^{(n)}_{max} = 1.265321,$	$x^{(n)}_{avg} = .953792$
rel.	$x^{(n)}_{min} = .851758,$	$x^{(n)}_{max} = 1.268927,$	$x^{(n)}_{avg} = .997791$

$\|\varepsilon^{(n)}\|_{h_4} = 2.762807,$ $RMSE^{(n)} = .0682560,$ $\|A^{1/2}(x^{(n)} - \alpha)\| = .0340234$

Table I-4: Result of Iterations on the Finest Level Only (Conjugate Gradient Method)

After the initial 23 iterations on level 1 :

$n = 23$, $l = 1$, $M_E^{h_1} = .97$
$||r^{(n)}||_{h_1} = .02904537$

	$x^{(n)}_{min} = .705024$,	$x^{(n)}_{max} = 1.041640$,	$x^{(n)}_{avg} = .905493$
rel.	$x^{(n)}_{min} = .722250$,	$x^{(n)}_{max} = 1.050381$,	$x^{(n)}_{avg} = .948284$
$||\varepsilon^{(n)}||_{h_1} = 1.366701$,		$RMSE^{(n)} = .0955011$,	$||\delta^{(n)}||_\infty = .00753925$

After prolongation operation $[I_{1\to 2}\, x^{h_1}]$ in FMRA :

$||r^{(n)}||_{h_2} = .05456750$

	$x^{(n)}_{min} = .705024$,	$x^{(n)}_{max} = 1.041640$,	$x^{(n)}_{avg} = .905679$
rel.	$x^{(n)}_{min} = .722250$,	$x^{(n)}_{max} = 1.068944$,	$x^{(n)}_{avg} = .948291$
$||\varepsilon^{(n)}||_{h_2} = 1.925627$,		$RMSE^{(n)} = .0951464$	

After 7 iterations on level 2 :

$n = 30$, $l = 2$, $M_E^{h_2} = .99$
$||r^{(n)}||_{h_2} = .02926091$

	$x^{(n)}_{min} = .701193$,	$x^{(n)}_{max} = 1.048129$,	$x^{(n)}_{avg} = .916837$
rel.	$x^{(n)}_{min} = .719006$,	$x^{(n)}_{max} = 1.053860$,	$x^{(n)}_{avg} = .959766$
$||\varepsilon^{(n)}||_{h_2} = 1.637754$,		$RMSE^{(n)} = .0809224$,	$||\delta^{(n)}||_\infty = .00564610$

After restriction operation $[I_{2\to 1}\, x^{h_2}$ and $A^{h_1} x^{h_1} + I_{2\to 1}(g^{h_2} - A^{h_2} x^{h_2})]$ in FMC :

$||r^{(n)}||_{h_1} = .01965193$

	$x^{(n)}_{min} = .701193$,	$x^{(n)}_{max} = 1.042649$,	$x^{(n)}_{avg} = .916537$
rel.	$x^{(n)}_{min} = .719006$,	$x^{(n)}_{max} = 1.049661$,	$x^{(n)}_{avg} = .959665$
$||\varepsilon^{(n)}||_{h_1} = 1.160005$,		$RMSE^{(n)} = .0810578$	

After prolongation operation $[I_{2\to 3}\, x^{h_2}]$ in FMRA :

$||r^{(n)}||_{h_3} = .01851928$

	$x^{(n)}_{min} = .745547$,	$x^{(n)}_{max} = 1.035361$,	$x^{(n)}_{avg} = .923353$
rel.	$x^{(n)}_{min} = .766526$,	$x^{(n)}_{max} = 1.049402$,	$x^{(n)}_{avg} = .966460$
$||\varepsilon^{(n)}||_{h_3} = 1.959972$,		$RMSE^{(n)} = .0684786$	

(Continued)

After prolongation operation $[I_{3\to4}\, x^{h_3}]$ in FMRA :

$\|r^{(n)}\|_{h_4} = .00758814$

	$x^{(n)}_{min} = .754358$,	$x^{(n)}_{max} = 1.040128$,	$x^{(n)}_{avg} = .928839$
rel.	$x^{(n)}_{min} = .779176$,	$x^{(n)}_{max} = 1.062586$,	$x^{(n)}_{avg} = .972136$

$\|\varepsilon^{(n)}\|_{h_4} = 2.529988$, $RMSE^{(n)} = .0625041$

After final iteration on level 1 :

$n = 98,\ l = 1,\ M_E^{h_1} = .97$

$\|r^{(n)}\|_{h_1} = .00052839$

	$x^{(n)}_{min} = .752202$,	$x^{(n)}_{max} = 1.041069$,	$x^{(n)}_{avg} = .928749$
rel.	$x^{(n)}_{min} = .781071$,	$x^{(n)}_{max} = 1.047655$,	$x^{(n)}_{avg} = .972245$

$\|\varepsilon^{(n)}\|_{h_1} = .898158$, $RMSE^{(n)} = .0627607$, $\|\delta^{(n)}\|_\infty = .00010293$

After prolongation operation $[x^{h_2} + I_{1\to2}(x^{h_1} - I_{2\to1}x^{h_2})]$ in FMC :

$\|r^{(n)}\|_{h_2} = .00143687$

	$x^{(n)}_{min} = .752202$,	$x^{(n)}_{max} = 1.041069$,	$x^{(n)}_{avg} = .929075$
rel.	$x^{(n)}_{min} = .781071$,	$x^{(n)}_{max} = 1.056168$,	$x^{(n)}_{avg} = .972389$

$\|\varepsilon^{(n)}\|_{h_2} = 1.262605$, $RMSE^{(n)} = .0623860$

After final iteration on level 2 :

$n = 105,\ l = 2,\ M_E^{h_2} = .99$

$\|r^{(n)}\|_{h_2} = .00089367$

	$x^{(n)}_{min} = .752694$,	$x^{(n)}_{max} = 1.041820$,	$x^{(n)}_{avg} = .928960$
rel.	$x^{(n)}_{min} = .780554$,	$x^{(n)}_{max} = 1.058650$,	$x^{(n)}_{avg} = .972272$

$\|\varepsilon^{(n)}\|_{h_2} = 1.264350$, $RMSE^{(n)} = .0624723$, $\|\delta^{(n)}\|_\infty = .00017401$

After prolongation operation $[x^{h_3} + I_{2\to3}(x^{h_2} - I_{3\to2}x^{h_3})]$ in FMC :

$\|r^{(n)}\|_{h_3} = .00285338$

	$x^{(n)}_{min} = .749737$,	$x^{(n)}_{max} = 1.041820$,	$x^{(n)}_{avg} = .928822$
rel.	$x^{(n)}_{min} = .777687$,	$x^{(n)}_{max} = 1.065809$,	$x^{(n)}_{avg} = .972111$

$\|\varepsilon^{(n)}\|_{h_3} = 1.783643$, $RMSE^{(n)} = .0623179$

(Continued)

After final iteration on level 3 :

$n = 112,\ l = 3,\ M_E^{h_3} = .999$

$\|r^{(n)}\|_{h_3} = .00224931$

	$x^{(n)}_{min} = .747626,$	$x^{(n)}_{max} = 1.042553,$	$x^{(n)}_{avg} = .928938$
rel.	$x^{(n)}_{min} = .780249,$	$x^{(n)}_{max} = 1.069988,$	$x^{(n)}_{avg} = .972230$
$\|\varepsilon^{(n)}\|_{h_3} = 1.780198,$		$RMSE^{(n)} = .0621976,$	$\|\delta^{(n)}\|_\infty = .00028079$

After prolongation operation $[x^{h_4} + I_{3\to4}(x^{h_3} - I_{4\to3}x^{h_4})]$ in FMC :

$\|r^{(n)}\|_{h_4} = .00526873$

	$x^{(n)}_{min} = .747597,$	$x^{(n)}_{max} = 1.042562,$	$x^{(n)}_{avg} = .928940$
rel.	$x^{(n)}_{min} = .779067,$	$x^{(n)}_{max} = 1.072773,$	$x^{(n)}_{avg} = .972243$
$\|\varepsilon^{(n)}\|_{h_4} = 2.514787,$		$RMSE^{(n)} = .0621286$	

After final iteration on level 4 :

$n = 119,\ l = 4,\ M_E^{h_4} = .99995$

$\|r^{(n)}\|_{h_4} = .00436349$

	$x^{(n)}_{min} = .747721,$	$x^{(n)}_{max} = 1.042188,$	$x^{(n)}_{avg} = .928940$
rel.	$x^{(n)}_{min} = .778485,$	$x^{(n)}_{max} = 1.073646,$	$x^{(n)}_{avg} = .972240$
$\|\varepsilon^{(n)}\|_{h_4} = 2.511991,$		$RMSE^{(n)} = .0620595,$	$\|\delta^{(n)}\|_\infty = .00064709$

Table I-5: Trace of Multigrid Algorithm (Adaptive Chebyshev Acceleration Method)

After 1174 iterations (on level 4) :

$n = 1174,\ l = 4,\ M_E^{h_4} = .99995$

$\|r^{(n)}\|_{h_4} = .00011623$

	$x^{(n)}_{min} = .734298,$	$x^{(n)}_{max} = 1.197495,$	$x^{(n)}_{avg} = .938368$
rel.	$x^{(n)}_{min} = .847392,$	$x^{(n)}_{max} = 1.206766,$	$x^{(n)}_{avg} = .981669$
$\|\varepsilon^{(n)}\|_{h_4} = 2.510787,$		$RMSE^{(n)} = .0620297,$	$\|\delta^{(n)}\|_\infty = .00001307$

Table I-6: Result of Iterations on the Finest Level Only (Adaptive Chebyshev Accel. Method)

After the initial 47 iterations on level 1 :

$n = 47$, $l = 1$, WU = .734375
$||r^{(n)}||_{h_1} = .02969730$
$x^{(n)}_{min} = .690886$, $x^{(n)}_{max} = .999192$, $x^{(n)}_{avg} = .882249$
rel. $x^{(n)}_{min} = .753693$, $x^{(n)}_{max} = 1.035565$, $x^{(n)}_{avg} = .923816$
$||\varepsilon^{(n)}||_{h_1} = 1.384535$, $RMSE^{(n)} = .0967473$

After prolongation operation $[I_{1\rightarrow 2}\, x^{h_1}]$ in FMRA :

$||r^{(n)}||_{h_2} = .05760785$
$x^{(n)}_{min} = .690886$, $x^{(n)}_{max} = .999192$, $x^{(n)}_{avg} = .883024$
rel. $x^{(n)}_{min} = .753693$, $x^{(n)}_{max} = 1.054821$, $x^{(n)}_{avg} = .924535$
$||\varepsilon^{(n)}||_{h_2} = 1.948891$, $RMSE^{(n)} = .0962959$

After 8 iterations on level 2 :

$n = 55$, $l = 2$, WU = 1.234375
$||r^{(n)}||_{h_2} = .01570253$
$x^{(n)}_{min} = .693834$, $x^{(n)}_{max} = 1.001389$, $x^{(n)}_{avg} = .893808$
rel. $x^{(n)}_{min} = .753563$, $x^{(n)}_{max} = 1.040087$, $x^{(n)}_{avg} = .935709$
$||\varepsilon^{(n)}||_{h_2} = 1.734804$, $RMSE^{(n)} = .0857177$

After restriction operation $[I_{2\rightarrow 1}\, x^{h_2}$ and $A^{h_1} x^{h_1} + I_{2\rightarrow 1}(g^{h_2} - A^{h_2} x^{h_2})]$ in FMC :

$||r^{(n)}||_{h_1} = .01137371$
$x^{(n)}_{min} = .693834$, $x^{(n)}_{max} = 1.001389$, $x^{(n)}_{avg} = .893065$
rel. $x^{(n)}_{min} = .753563$, $x^{(n)}_{max} = 1.035467$, $x^{(n)}_{avg} = .935051$
$||\varepsilon^{(n)}||_{h_1} = 1.231326$, $RMSE^{(n)} = .0860415$

After prolongation operation $[I_{2\rightarrow 3}\, x^{h_2}]$ in FMRA :

$||r^{(n)}||_{h_3} = .01824689$
$x^{(n)}_{min} = .708452$, $x^{(n)}_{max} = 1.008463$, $x^{(n)}_{avg} = .906173$
rel. $x^{(n)}_{min} = .775397$, $x^{(n)}_{max} = 1.038347$, $x^{(n)}_{avg} = .948537$
$||\varepsilon^{(n)}||_{h_3} = 2.082891$, $RMSE^{(n)} = .0727732$

(Continued)

After prolongation operation $[I_{3\to4}\, x^{h_3}]$ in FMRA :

$\|r^{(n)}\|_{h_4} = .00733104$

	$x^{(n)}_{min} = .711046$,	$x^{(n)}_{max} = 1.009373$,	$x^{(n)}_{avg} = .912234$
rel.	$x^{(n)}_{min} = .779310$,	$x^{(n)}_{max} = 1.038696$,	$x^{(n)}_{avg} = .954825$

$\|\varepsilon^{(n)}\|_{h_4} = 2.764502$, $RMSE^{(n)} = .0682978$

After final iteration on level 1 :

$n = 137,\ l = 1,\ \mathrm{WU} = 17.515625$

$\|r^{(n)}\|_{h_1} = .00010632$

	$x^{(n)}_{min} = .712825$,	$x^{(n)}_{max} = 1.009376$,	$x^{(n)}_{avg} = .912414$
rel.	$x^{(n)}_{min} = .782350$,	$x^{(n)}_{max} = 1.025581$,	$x^{(n)}_{avg} = .955122$

$\|\varepsilon^{(n)}\|_{h_1} = .977956$, $RMSE^{(n)} = .0683368$

After prolongation operation $[x^{h_2} + I_{1\to2}(x^{h_1} - I_{2\to1}x^{h_2})]$ in FMC :

$\|r^{(n)}\|_{h_2} = .00022244$

	$x^{(n)}_{min} = .712825$,	$x^{(n)}_{max} = 1.009376$,	$x^{(n)}_{avg} = .912975$
rel.	$x^{(n)}_{min} = .781922$,	$x^{(n)}_{max} = 1.029144$,	$x^{(n)}_{avg} = .955559$

$\|\varepsilon^{(n)}\|_{h_2} = 1.379051$, $RMSE^{(n)} = .0681397$

After final iteration on level 2 :

$n = 145,\ l = 2,\ \mathrm{WU} = 18.015625$

$\|r^{(n)}\|_{h_2} = .00015623$

	$x^{(n)}_{min} = .712866$,	$x^{(n)}_{max} = 1.009441$,	$x^{(n)}_{avg} = .913078$
rel.	$x^{(n)}_{min} = .782089$,	$x^{(n)}_{max} = 1.029073$,	$x^{(n)}_{avg} = .955665$

$\|\varepsilon^{(n)}\|_{h_2} = 1.377252$, $RMSE^{(n)} = .0680508$

After prolongation operation $[x^{h_3} + I_{2\to3}(x^{h_2} - I_{3\to2}x^{h_3})]$ in FMC :

$\|r^{(n)}\|_{h_3} = .00064137$

	$x^{(n)}_{min} = .711539$,	$x^{(n)}_{max} = 1.009441$,	$x^{(n)}_{avg} = .913123$
rel.	$x^{(n)}_{min} = .780050$,	$x^{(n)}_{max} = 1.033644$,	$x^{(n)}_{avg} = .955719$

$\|\varepsilon^{(n)}\|_{h_3} = 1.943675$, $RMSE^{(n)} = .0679092$

(Continued)

After final iteration on level 3 :

$n = 153,\ l = 3,\ \text{WU} = 20.015625$
$\|r^{(n)}\|_{h_3} = .00040012$

	$x^{(n)}_{min} = .711450,$	$x^{(n)}_{max} = 1.009423,$	$x^{(n)}_{avg} = .913260$
rel.	$x^{(n)}_{min} = .780049,$	$x^{(n)}_{max} = 1.033614,$	$x^{(n)}_{avg} = .955860$

$\|\varepsilon^{(n)}\|_{h_3} = 1.941612,$ $RMSE^{(n)} = .0678371$

After prolongation operation $[x^{h_4} + I_{3\rightarrow 4}(x^{h_3} - I_{4\rightarrow 3}x^{h_4})]$ in FMC :

$\|r^{(n)}\|_{h_4} = .00186325$

	$x^{(n)}_{min} = .711095,$	$x^{(n)}_{max} = 1.009423,$	$x^{(n)}_{avg} = .913338$
rel.	$x^{(n)}_{min} = .779602,$	$x^{(n)}_{max} = 1.038509,$	$x^{(n)}_{avg} = .955964$

$\|\varepsilon^{(n)}\|_{h_4} = 2.743285,$ $RMSE^{(n)} = .0677737$

After final iteration on level 4 :

$n = 161,\ l = 4,\ \text{WU} = 28.015625$
$\|r^{(n)}\|_{h_4} = .00105797$

	$x^{(n)}_{min} = .710992,$	$x^{(n)}_{max} = 1.009429,$	$x^{(n)}_{avg} = .913534$
rel.	$x^{(n)}_{min} = .779613,$	$x^{(n)}_{max} = 1.038426,$	$x^{(n)}_{avg} = .956166$

$\|\varepsilon^{(n)}\|_{h_4} = 2.740650,$ $RMSE^{(n)} = .0677086$

Table I-7: Trace of Multigrid Algorithm (Gauss-Seidel Method)

References

[Aspe 87] *An Introduction to the PIPE System*
Aspex Incorporated, 1987.

[Back 78] Backus, J.
Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs.
Communications of the ACM, 21(8):613-641, 1978.

[Batc 83] Batcher, K.
General Description of the MPP.
Technical Report GER-17140, Goodyear Aerospace Corp., April, 1983.

[Bawd 84] Bawden, A.
A Programming Language for Massively Parallel Computers.
Master's thesis, Massachusetts Institute of Technology, August, 1984.

[Blak 87] Blake, A. and Zisserman, A.
Visual Reconstruction.
MIT Press, 1987.

[Boul 86] Boult, T. E.
Information-based Complexity in Nonlinear Equations and Computer Vision.
PhD thesis, Columbia University, October, 1986.

[Bran 77] Brandt, A.
Multi-Level Adaptive Solutions to Boundary-Value Problems.
Mathematics of Computation, 31(138):333-390, April, 1977.

[Bran 82] Brandt, A.
Guide to Multigrid Development.
Multigrid Methods.
Springer-Verlag, 1982, pages 220-312.

[Brig 87] Briggs, W. L.
A Multigrid Tutorial.
Society for Industrial and Applied Mathematics, 1987.

[Brou 83] Brou, P.
Finding the Orientation of Objects in Vector Maps.
PhD thesis, Massachusetts Institute of Technology, July, 1983.

[Choi 85a] Choi, D. J. and Shaw, D. E.
The NON-VON 3 Simulator User's Guide.
1985

[Choi 85b] Choi, D. J. and Kender, J. R.
Solving the Depth Interpolation Problem with the Adaptive Chebyshev Acceleration Method on a Parallel Computer.
In *Image Understanding Workshop*, pages 219-223. 1985.

[Choi 87] Choi, D. J. and Kender, J. R.
Solving the Depth Interpolation Problem on a Parallel Architecture.
In *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, pages 107-114. October, 1987.

[Choi 88] Choi, D. J. and Kender, J. R.
Solving the Depth Interpolation Problem on a Parallel Architecture with a Multigrid Approach.
In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* June, 1988.
(To appear).

[Chri 84] Christman, D. P.
Programming the Connection Machine.
Master's thesis, Massachusetts Institute of Technology, January, 1984.

[Duff 83] Duff, M. J. B. (editor).
Computing Structures for Image Processing.
Academic Press, 1983.

[Duff 86] Duff, M. J. B. (editor).
Intermediate-level Image Processing.
Academic Press, 1986.

[Dyer 79] Dyer, C. R.
Augmented Cellular Automata for Image Analysis.
PhD thesis, University of Maryland, March, 1979.

[Dyer 81] Dyer, C. R.
A VLSI Pyramid Machine for Hierarchical Parallel Image Processing.
In *Proceedings of the IEEE Conference on Parallel Recognition and Image Processing*, pages 381-386. 1981.

[Gema 84] Geman, S. and Geman, D.
Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images.
IEEE Trans. Pattern Anal. Machine Intell., PAMI-6(6):721-741, November, 1984.

[Gilm 83] Gilmore, P. A.
The Massively Parallel Processor (MPP) : A Large Scale SIMD Processor.
In *Proceedings of SPIE*, pages 166-174. 1983.

[Golu 85] Golub, G. H. and Van Loan, C. F.
Matrix Computations.
Johns Hopkins University Press, 1985.

[Gott 86] Gottlieb, A.
An Overview of the NYU Ultracomputer Project.
Technical Report 100, NYU, July, 1986.
Ultracomputer Note.

[Graf 86] Graf, H.P., Jackel, L.D., et al.
VLSI Implementation of a Neural Network Memory with Several Hundreds of Neurons.
In *Neural Networks for Computing*, pages 182-187. American Inst. of Physics, 1986.

[Grim 81] Grimson, W. E. L.
From Images to Surfaces.
MIT Press, 1981.

[Grim 83] Grimson, W. E. L.
An Implementation of a Computational Theory of Visual Surface Interpolation.
Computer Vision, Graphics, and Image Processing, 22:39-69, 1983.

[Hack 85] Hackbusch, W.
Multi-Grid Methods and Applications.
Springer-Verlag, 1985.

[Hack 86] Hackbusch, W. and Trottenberg, U. (editors).
Multigrid Methods II.
Springer-Verlag, 1986.

[Hans 86] Hansen, C. and Henderson, T.
UTAH Range Database.
Technical Report UUCS-86-113, University of Utah, April, 1986.

[Hill 81] Hillis, W. D.
The Connection Machine (Computer Architecture for the New Wave).
Technical Report 646, MIT, September, 1981.
A.I. Memo.

[Hill 86] Hillis, W. D.
The Connection Machine.
MIT Press, 1986.

[Hillyer 86] Hillyer, B. K.
On Applying Heterogeneous Parallelism to Elements of Knowledge-Based Data Management.
PhD thesis, Columbia University, 1986.

[Horn 81] Horn, B. K. P. and Schunck, B. G.
Determining Optical Flow.
Artificial Intelligence, 17:185-203, 1981.

[Humm 87] Hummel, R. and Zhang, K.
Dynamic Processor Allocation for Parallel Algorithms in Image Processing.
Technical Report 123, NYU, January, 1987.
Ultracomputer Note.

[Huss 84] Hussein, A. H. I.
Image Understanding Algorithms on Fine-Grained Tree-Structured SIMD Machines.
PhD thesis, Columbia University, October, 1984.

[IBM 84] *VS FORTRAN Language and Library Reference*
1st edition, IBM, 1984.
Release 4.0.

[Ikeu 81] Ikeuchi, K. and Horn, B. K. P.
Numerical Shape from Shading and Occluding Boundaries.
Artificial Intelligence, 17:141-184, 1981.

[Jack 86] Jackel, L.D., Denker, J.S., et al.
Electronic Neural Computing.
1986
AT&T Bell Laboratories.

[Koch 84]	Koch, C. and Poggio, T. *Biophysics of Computation: Neurons, Synapses and Membranes.* Technical Report 795, MIT, October, 1984. A.I. Memo.
[Lee 85]	Lee, D. *Contributions to Information-based Complexity, Image Understanding and Logic Circuit Design.* PhD thesis, Columbia University, October, 1985.
[Litt 86]	Little, J. J. *Parallel Algorithms for Computer Vision on the Connection Machine.* Technical Report 928, MIT, November, 1986. A.I. Memo.
[Litt 87]	Little, J. J., Blelloch, G., and Cass, T. How to Program the Connection Machine for Computer Vision. In *1987 Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, pages 11-18. 1987.
[Mann 74]	Manna, Z. *Mathematical Theory of Computation.* McGraw-Hill, 1974.
[Marr 82]	Marr, D. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* W. H. Freeman and Company, 1982.
[McCo 87]	McCormick, S. (editor). *Multigrid Methods.* Society for Industrial and Applied Mathematics, 1987.
[Mins 85]	Minsky, M. (editor). *Robotics.* Anchor Press/Doubleday, 1985.
[Mins 86]	Minsky, M. *The Society of Mind.* Simon and Schuster, 1986.
[Pott 85]	Potter, J. L. *The Massively Parallel Processor.* MIT Press, 1985.
[Rose 84]	Rosenfeld, A. (editor). *Multiresolution Image Processing and Analysis.* Springer-Verlag, 1984.
[Same 80]	Samet, H. Region Representation: Quadtrees from Binary Arrays. *Computer Graphics and Image Processing*, 13:88-93, 1980.
[Schu 83]	Schunck, B. G. *Motion Segmentation and Estimation.* PhD thesis, Massachusetts Institute of Technology, May, 1983.

[Seit 84] Seitz, C. L.
Concurrent VLSI Architectures.
IEEE Trans. Comput., C-33(12):1247-1265, December, 1984.

[Shaw 80] Shaw, D. E.
Knowledge-Based Retrieval on a Relational Database Machine.
PhD thesis, Stanford University, August, 1980.

[Shaw 82] Shaw, D. E.
The NON-VON Supercomputer.
Technical Report, Columbia University, August, 1982.

[Shaw 84a] Shaw, D. E. and Sabety, T. M.
An Eight-Processor Chip for a Massively Parallel Machine.
Technical Report, Columbia University, July, 1984.

[Shaw 84b] Shaw, D. E.
Organization and Operation of a Massively Parallel Machine.
Technical Report, Columbia University, October, 1984.

[Stew 86] Stewart C. V. and Dyer C. R.
Convolution Algorithms on the Pipelined Image-Processing Engine.
Technical Report, University of Wisconsin, May, 1986.

[Stol 87] Stolfo, S. J.
Initial Performance of the DADO2 Prototype.
Computer, :75-83, January, 1987.

[Terz 83] Terzopoulos, D.
Multilevel Computational Processes for Visual Surface Reconstruction.
Computer Vision, Graphics, and Image Processing, 24:52-96, 1983.

[Terz 84] Terzopoulos, D.
Multiresolution Computation of Visible-surface Representations.
PhD thesis, Massachusetts Institute of Technology, January, 1984.

[Terz 85a] Terzopoulos, D.
Concurrent Multilevel Relaxation.
In *Image Understanding Workshop*, pages 156-162. 1985.

[Terz 85b] Terzopoulos, D.
Computing Visible-surface Representations.
Technical Report 800, MIT, March, 1985.
A.I. Memo.

[Terz 86] Terzopoulos, D.
Image Analysis Using Multigrid Relaxation Methods.
IEEE Trans. Pattern Anal. Machine Intell., PAMI-8(2):129-139, March, 1986.

[Thom 85] Thompson, R. F.
The Brain: An Introduction to Neuroscience.
W. H. Freeman and Company, 1985.

[Tikh 77] Tikhonov, A. N. and Arsenin, V. Y.
Solutions of Ill-Posed Problems.
V. H. Winston & Sons, 1977.

[TMC 86] .
Introduction to Data Level Parallelism.
Technical Report, Thinking Machines Corporation, April, 1986.
Thinking Machines Technical Report 86.14.

[TMC 87a] .
Connection Machine Model CM-2 Technical Summary.
Technical Report, Thinking Machines Corporation, April, 1987.
Thinking Machines Technical Report HA87-4.

[TMC 87b] Lim, W., Agrawal, A., and Nekludova, L.
A Fast Parallel Algorithm for Labeling Connected Components in Image Arrays.
Technical Report, Thinking Machines Corporation, April, 1987.
Thinking Machines Technical Report NA86-2.

[Trau 84] Traub, J. F. and Wozniakowski, H.
On the Optimal Solution of Large Linear Systems.
Journal of the Association for Computing Machinery, 31(3):545-559, 1984.

[Utah 84] The Utah Symbolic Computation Group.
The Portable Standard LISP Users Manual
University of Utah, 1984.
Version 3.2.

[Vish 83] Vishkin, U.
Synchronous Parallel Computation: A Survey.
Technical Report 53, NYU, April, 1983.
Ultracomputer Note.

[Wozn 77] Wozniakowski, H.
Numerical Stability of the Chebyshev Method for the Solution of Large Linear Systems.
Numerische Mathematik, 28:191-209, 1977.

[Wozn 80] Wozniakowski, H.
Roundoff-Error Analysis of a New Class of Conjugate-Gradient Algorithms.
Linear Algebra and its Applications, 29:507-529, 1980.

[Youn 81] Young, D. M. and Hageman, L. A.
Applied Iterative Methods.
Academic Press, 1981.