FPGA PLATFORM FOR REAL-TIME SIMULATION OF TISSUE DEFORMATION

FPGA PLATFORM FOR REAL-TIME SIMULATION OF TISSUE DEFORMATION

By

SAMSON AJAGUNMO, B.ENG.MGNT. (McMASTER UNIVERSITY)

A Thesis

Submitted to the School of Graduate Studies

In Partial Fulfillment of the Requirements

For the Degree

Masters of Applied Science in Engineering

McMaster University

MASTER OF APPLIED SCIENCE (2007) **McMASTER UNIVERSITY**

(Electrical and Computer Engineering) Hamilton, Ontario, Canada

TITLE: FPGA Platform for Real-Time Simultion of Tissue Deformation

AUTHORS: Samson Ajagunmo

SUPERVISOR: Dr. Aleksandar Jeremic

NUMBER OF PAGES: X, 52

ABSTRACT

The simulation of soft tissue deformations has many practical uses in the medical field such as diagnosing medical conditions, training medical professionals and surgical planning. While there are many good computational models that are used in these simulations, carrying out the simulations is time consuming especially for large systems. This is because most simulators are based on software, which are run on general-purpose computers (GPC) that are not optimized to carry out the operations needed for simulation. In order to improve the performance of these simulators, field-programmable-gate-arrays (FPGA) based accelerators for carrying out Matrix-by-Vector multiplications (MVM) have been implemented by Ramachandran in 1998 and Zhuo et. al. in 2005. Zhuo et. al. also looked at the best ways to store a matrix in memory, and how this is affected by certain properties of the matrix.

A better approach is to implement an accelerator for carrying out all operations required for simulation on hardware. In this study we propose a hardware accelerator for simulating soft-tissue deformation using finite-difference approximation of elastodynamics equations based on conjugate-gradient inversion of sparse matrices. We designed and implemented the accelerator, which is optimized for use with sparse matrices, on FPGA. We also conducted performance and resource requirements analysis for the accelerator. Our results show this approach is capable of achieving sufficiently high computational rate for carrying out real-time simulation; even with large grids or meshes. Finally, we developed computational models for carrying out real-time simulation of tissue deformation.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Aleksandar Jeremic for his help in my research. Thank you for your advice, support, and most importantly for being very patient with me.

Mum, Dad, thank you for your sacrifice, support, and for believing in me. This would not have been possible without you. Ezekiel, for your support, I thank you.

Annette, you are the best. Thank you for your constant encouragement, for making the long hours seem shorter, and most of all for your unwavering belief in me. Harry, and Barbara thank you for your interest and encouragement throughout my research.

I dedicate this thesis to my Mum, Dad, Annette, and my brother Ezekiel.

NOMENCLATURE

u	-	*Change in Length of a spring or tissue.*
u^d	-	*Change in Length of a spring or tissue in direction* d.
$\tilde{k}$	-	*Spring or Tissue Elasticity.*
$\tilde{l}^d$	-	*Displacement in direction* d.
f^d	-	*Load acting at a point in direction* d.
$u^d_{i+l,j+m,k+n}$	-	*Change in Length of the spring connecting nodes i,j,k and i+l,j+m,k+n in direction* d.
$\tilde{k}_{i+l,j+m,k+n}$	-	*Elasticity of the spring connecting nodes i,j,k and i+l,j+m,k+n .*
$\tilde{l}^d_{i,j,k}$	-	*Displacement of node i,j,k in direction* d.
$f^d_{i,j,k}$	-	*Load at node i,j,k in direction* d.
$\tilde{\mathbf{l}}^d$	-	*Vector of nodal Displacements in direction* d.
$\mathbf{f}^d$	-	*Vector of Loads in direction* d.
$\mathbf{K}$	-	*Stiffness Matrix.*
Δ	-	*Finite-Difference Operator.*
c	-	*Condition Number of a Matrix.*
ρ	-	*Lamé constant*
E	-	*Elastic Modulus.*
Q	-	*Shear Modulus.*
FDM	-	*Finite Difference Method.*
CGM	-	*Conjugate Gradient Method.*

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Some of the most common procedures in clinical practice involve the insertion of subcutaneous needles into tissue for biopsy of deep-seated tumors. These procedures are extremely sensitive to path (trajectory) planning and initial placement of the needle. The initial placement and path of the needle are usually chosen so as to set up a straight-line intercept with the target tumors. If a needle fails to reach its target, it must be retracted and the procedure is repeated. It may require several attempts to achieve a precise needle placement [DiMaio3]. One of the current trends in the training of medical professionals on these procedures is the development, and use, of simulators for tissue deformation [DiMaio2]. These simulators have also become important tools in surgical planning. Realistic simulation of tissue deformation undergoing these procedures is the bottleneck of all simulators.

The deformation of soft tissue is determined by elastodynamic partial differential equations (PDEs) [Fung2], defined over irregular domains (human organs). Due to the irregular shapes of these domains, a solution cannot be obtained analytically. In order to solve these equations we need discretization techniques such as the finite-difference method (FDM) and the finite-element method (FEM). In both methods, the domain of interest is discretized and the corresponding PDEs are transformed into a new set of equations, usually linear equation. This set of equations can be represented using a matrix ("stiffness" matrix), a load vector, and a vector of unknowns. The resulting system is then solved using numerical algorithms such as Newton's method, conjugate-gradient method (CGM) etc. These numerical algorithms are usually run on a GPC. The main difficulty with this approach is the size of tissue/organ of interest since desired accuracy may require a large grid (number of nodes). Consequently the size of the corresponding "stiffness" matrices describing the resulting system may make obtaining a solution very computationally expensive. Since the GPCs are not optimized to carry out most operations that are used in the numerical methods, solving the model also becomes time consuming for large grids. A possible solution to this problem is to develop hardware that is optimized to perform the tasks needed in numerical algorithm. In this thesis we propose FPGA platform for solving elastodynamic systems using FDM approximation and CGM for solving corresponding linear system. We exploit the fact that the "stiffness" matrix that describes the resulting system is sparse and band-limited, and develop hardware that executes CGM.

1.1 Related Work

Most of the previous work done in speeding up numerical methods, using hardware, focused on the implementation of efficient matrix-by-vector multiplier units (MVU) on FPGA. In Ramachandran, 1998, the author investigated the performance effects of using an FPGA based MVU to carry out an MVM. The MVU implemented a column-based SAXPY (Scalar Alpha X Plus Y) multiplication. In other words, a linear combination of the column of the matrix was done. The MVU communicated with a host computer, which used its results in solving a linear system that was generated from the Finite-Element method with a performance of 36 MFLOPS.

In Zhuo et. al., 2005, the authors also developed an MVU for MVMs that involved sparse matrices. Their method involved storing only the nonzero elements of a matrix in FPGA memory. The column index of the nonzero element and the number of nonzero elements in each row of the matrix were also stored in memory. This information was used to determine the position of the nonzero elements of the matrix and to make sure that the correct values are used in the multiplications. Their design in attained a performance of 350 MFLOPS for all their test cases. This is a 900% increase in performance when compared with results in Ramachandran, 1998. Possible factors for such an improvement by Zhuo et. al. may include the higher clock frequencies of FPGAs as of 2005 and the elimination of unnecessary multiplications involving zeros in their design.

1.2 Research Objectives

The objectives of our research are outlined below

1. Implement an FPGA based accelerator that executes a full iterative method, in this case the CGM, using parallel computing for carrying out MVMs.
2. Further, we determine the performance and resource requirements of the implementation, and the feasibility and limitation of this approach to solving computational models.

1.3 Research Approach

We carried out our study using the following steps:

1. Develop 3D computational models, for tissue deformation using the Finite-Difference method.

2. Look at the properties of the stiffness matrix generated from the 3D computational models.
3. Test for the rates of convergence of the computational models from 1 using a numerical example.
4. Design and implement a CGM accelerator that takes advantage of the properties in 2 on an FPGA.
5. Finally, measure the performance and resource requirements of this design.

1.4 Thesis Layout

This thesis consists of 6 chapters. In Chapter 2 we will introduce the constitutive equations that describe the physical properties of deformable materials and numerical techniques that are used generate and solve the FDM computational models for tissue deformation. We also generate FDM computational models for the modeling of tissue deformation, look at the properties of these computational models, and perform a convergence analysis using MATLAB and a numerical example. In Chapter 3, we focus on the inverse models for estimating the elasticity of a tissue. Then we estimate the elasticity of a homogeneous tissue using a numerical example and MATLAB. The design and implementation of the CGM accelerator are discussed in Chapter 4. In Chapter 5 we look at the resource requirements, and performance analysis of the CGM accelerator. In Chapter 6 we conclude by discussing the limitations, and possible future work that can be done for improvement.

2 FORWARD MODEL

In this Chapter, we introduce equations that govern the deformation of tissues and give a brief description of the numerical tools used in this study, namely the finite difference method and the conjugate gradient method. Then we generate computational models, which are suitable for hardware use. We use the FDM because it produces a very sparse stiffness matrix. Solving the linear systems generated from the FDM usually involves the calculation of a matrix inverse. The calculation of this matrix inverse requires arithmetic operations of the order 2^n for a system of size n. This value gets very large for large n, thus rather than computing matrix inverse, numerical algorithms such as CGM, Newton's method, method of steepest descent, and others are used to solve the system. We use the CGM because it has certain properties that suit its application on an FPGA (see Section 2.4).

2.1 Constitutive Equations for Tissue Deformation

A deformable material can be described as a collection of a very large number of particles and as such, it is often considered as a continuum. The real number system in mathematics is a continuum. In this system, there is another real number between any two real numbers [Fung2]. Extending the same concept to a material continuum, between any two particles there is another particle. The constitutive equations that describe the physical properties of a material must be satisfied by each particle of the materials continuum. While there are as many constitutive equations that describe as many different materials, they all give stress-strain relationships. Thus, physical properties such as the deformation of a material can be described by the relationship between the external forces acting on the tissue and the resulting strain in the tissue.

***Strain*:** The deformation of an elastic material is characterized by the change in its lengths, which in turn can be characterized by strain. Strain, and consequently deformation, has two components: axial strain and shear strain. The shear strain in a material is a result of twisting, while stretching causes axial strain. Consider a sample of 3-dimensional elastic material with lengths L^d in each dimension d. When this material is deformed, it lengths change by u^d in each dimension d. The axial and shear strains in each dimension are given by (1).

$$\begin{aligned} &\textit{Axial Strains}:\ \varepsilon_{11}=\frac{u^1}{L^1},\ \varepsilon_{22}=\frac{u^2}{L^2},\ \varepsilon_{33}=\frac{u^3}{L^3} \\ &\textit{Shear Strains}:\ \varepsilon_{12}=\frac{u^1}{L^1}-\frac{u^2}{L^2},\ \varepsilon_{13}=\frac{u^1}{L^1}-\frac{u^3}{L^3},\ \varepsilon_{23}=\frac{u^2}{L^2}-\frac{u^3}{L^3} \end{aligned} \qquad (1)$$

***Stress*:** Stress is a result of forces acting on a deformable material. By definition, stress is a measure of the force per unit surface area. Like strain, stress can also be divided into two components: axial stress and shear stress, which cause corresponding axial and shear strains. The equations describing the relationship between stress and strain in a 3-dimensional material for stress values that do not exceed the elastic limit of the material are given in (2).

$$\begin{aligned} &\textit{Axial Stresses}: \quad \begin{aligned} \sigma_{11}&=\rho(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33})+2Q\varepsilon_{11} \\ \sigma_{22}&=\rho(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33})+2Q\varepsilon_{22} \\ \sigma_{33}&=\rho(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33})+2Q\varepsilon_{33} \end{aligned} \\ &\textit{Shear Stresses}: \quad \sigma_{12}=2Q\varepsilon_{12},\ \sigma_{13}=2Q\varepsilon_{13},\ \sigma_{23}=2Q\varepsilon_{23} \end{aligned} \qquad (2)$$

$$\upsilon=\frac{E}{2Q}-1,\ \rho=\frac{E\upsilon}{(1+\upsilon)(1-2\upsilon)}$$

Where E is the elastic modulus, Q is the shear modulus, ρ is a Lamé constant, and υ is Poisson's ratio.

These stresses must also satisfy the laws of conservation of mass and momentum in each element along with the required constraints and boundary conditions. These requirements are met when the stresses satisfy the set of equations in (3).

$$\begin{aligned}
&\frac{\partial \sigma_{11}}{\partial L^1} + \frac{\partial \sigma_{12}}{\partial L^2} + \frac{\partial \sigma_{13}}{\partial L^3} + b_1 = 0 \\
&\frac{\partial \sigma_{21}}{\partial L^1} + \frac{\partial \sigma_{22}}{\partial L^2} + \frac{\partial \sigma_{23}}{\partial L^3} + b_2 = 0 \\
&\frac{\partial \sigma_{31}}{\partial L^1} + \frac{\partial \sigma_{32}}{\partial L^2} + \frac{\partial \sigma_{33}}{\partial L^3} + b_3 = 0
\end{aligned} \tag{3}$$

Where b_1, b_2, and b_3 are the body forces per unit volume in the first, second, and third dimensions. For more detailed explanation on this subject matter the reader is referred to [Fung2].

While equations (1), (2), and (3) apply to most deformable materials, they do not accurately describe the deformation of soft tissues. Most soft tissues exhibit both elastic and viscous behavior, thus they are considered viscoelastic materials [Delingette]. A tissue can be described as a collection of particles (nodes) with adjacent nodes connected together by a spring and a damper. The Newtonian motion of any two connected nodes causes a change in the length of the connecting spring and consequently strain in the spring. Recall that the deformation of a tissue can be described by the relationship between the strain in the tissue and the forces acting on the tissue. Note that the strain of the tissue is now, given by the strain in the system of springs.

The force of interaction, f_s, between two connected nodes is given by (4), the sum of the forces in the spring and damper connecting the two nodes.

$$f_s = \beta \frac{\partial u}{\partial t} + \tilde{k}\, u \qquad (4)$$

Where u is the change in length of the spring, $\tilde{k}$ is elasticity of the spring, and β is the damping constant of the damper. The Newtonian motion of each node can be expressed in terms of its nodal acceleration, external forces, interaction forces between the connected nodes, the change in length of the connecting spring, and the rate at which the length of the connecting spring changes with respect to time, as seen in (5).

$$\begin{aligned} f &= \mathbf{m}a + f_s \\ f &= \mathbf{m}a + \beta \frac{\partial u}{\partial t} + \tilde{k}\, u \end{aligned} \qquad (5)$$

Where $\mathbf{m}$ is the nodal mass value, a nodal acceleration, and f is the external load acting at a node. Since each node will have more than one connected adjacent node, the Newtonian motion of each node is given by (6), which takes into account the sum of interaction forces with all the connected nodes.

$$\begin{aligned} f &= \mathbf{m}a + \sum_i f_{s_i} \\ f &= \mathbf{m}a + \sum_i (\beta_i \frac{\partial u_i}{\partial t} + \tilde{k}_i\, u_i) \end{aligned} \qquad (6)$$

Consider the node with mass $\mathbf{m}_1$ in the system shown in Figure 2-1. The motion of this node, relative to the second node and the wall, is described by (7).

$$\mathbf{m}_1 a + \beta_2 \frac{\partial u_2}{\partial t} - \beta_1 \frac{\partial u_1}{\partial t} + \tilde{k}_2\, u_2 - \tilde{k}_1\, u_1 = 0 \qquad (7)$$

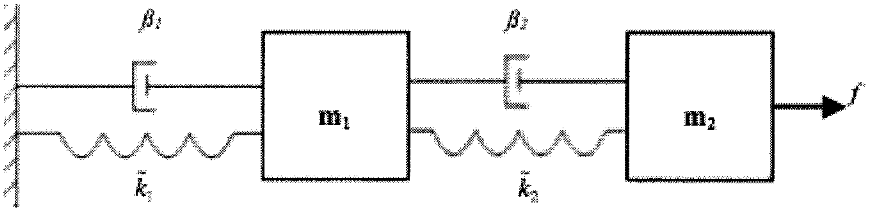


Figure 2-1: Spring-Damper System Model

When an external axial force acts on a node, it causes a change in the position of the node. This change in position in turn causes an opposing (restoration) force in the springs connected to the node, which attempts to restore the node to its original position. This change in position, due to the external force, continues until an equal restoration force is generated by the springs, thus keeping the system in equilibrium. At this point, (5), (6) and (7) are reduced to the hookean equations shown in (8), (9) and (10). Thus, at

$$f = \tilde{k}\, u \tag{8}$$

$$f = \sum_i \tilde{k}_i\, u_i \tag{9}$$

$$\tilde{k}_2\, u_2 - \tilde{k}_1\, u_1 = 0 \tag{10}$$

equilibrium, a tissue can be described as a collection of nodes with adjacent nodes connected together by a spring. Consequently, at equilibrium the system in Figure 2-1 can be replaced by the system in Figure 2-2.

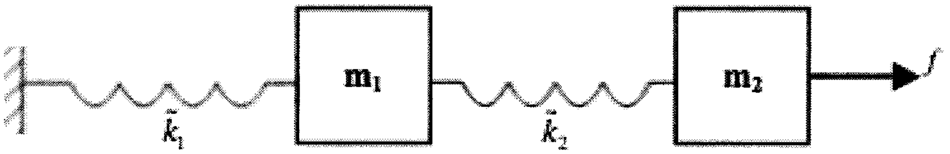


Figure 2-2: Spring-Damper System Model at Equlibrium

2.2 Finite-Difference Method

The Finite-Difference method is a very popular numerical tool that is used to create computational models for many large systems. The technique is used to discretize the constitutive equations that describe the physical properties of the system in question. As seen in the previous section, these equations are either polynomials or partial-differential equations. The finite-difference method involves dividing the solution domain into a grid of discrete points. The constitutive equations are then rewritten at each of these nodes, such that the terms of the constitutive equations are replaced with the equivalent finite-difference or finite-divided difference equations. Each node has global indexes, which identifies its position within the grid.

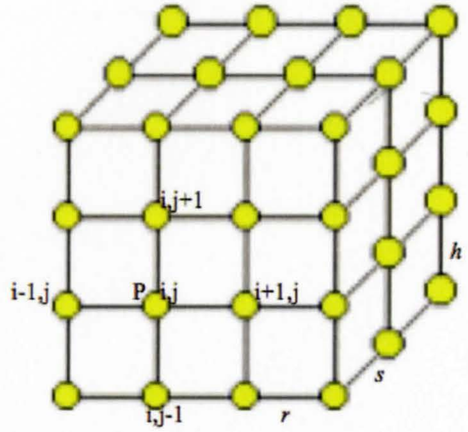


Figure 2-3: Finite Difference Node Grid

We use the three dimensional grid in Figure 2-3 to illustrate an example of global node indexing. Note that the nodes have an equal spacing of dx = r, dy = s, and dz = h in the x-, y-, and z- dimensions (or directions) respectively.

Using this indexing, a function f of three independent variables is evaluated at a point P as follows:

$$f(P) = f(x_p, y_p, z_p) = f(ir, js, kh) = f_{i,j,k} \qquad (11)$$

Using the notation in (11); (12) - (14) show the three forms of obtaining difference equations. These finite-differences are taken with respect to x at point P.

Forward Difference: $\Delta f(x_p, y_p, z_p)_x = f(x_p + r, y_p, z_p) - f(x_p, y_p, z_p) = f_{i+1,j,k} - f_{i,j,k}$ (12)

Backward Difference: $\Delta f(x_p,y_p,z_p)_x = f(x_p,y_p,z_p) - f(x_p - r,y_p,z_p) = f_{i,j,k} - f_{i-1,j,k}$ (13)
Central Difference: $\Delta f(x_p,y_p,z_p)_x = f(x_p + r,y_p,z_p) - f(x_p - r,y_p,z_p) = f_{i+1,j,k} - f_{i-1,j,k}$ (14)

Where Δ is the difference operator. The nth order difference equations have the general forms given in (15), (16), and (17).

$$\text{Forward Difference}: \; \Delta^n f(x_p,y_p,z_p)_x = \sum_m^n {}^nC_m(-1)^m f_{i+n-m,j,k} \tag{15}$$

$$\text{Backward Difference}: \; \Delta^n f(x_p,y_p,z_p)_x = \sum_m^n {}^nC_m(-1)^m f_{i-m,j,k} \tag{16}$$

$$\text{Central Difference}: \; \Delta^n f(x_p,y_p,z_p)_x = \sum_m^{n-1} {}^nC_m(-1)^m (f_{i+n-m,j,k} - f_{i-m,j,k}) \tag{17}$$

The nth partial derivative, with respect to *x*, of function *f* at point P is calculated by using the nth finite difference as shown in (18) and (19). These equations are called finite divided difference. The central difference, forward difference, and backward difference forms in (15), (16), and (17) are used to calculate the central divided-difference, forward divided-difference, and backward divided-difference respectively. This process can be

$$\text{Forward and Backward Difference}: \frac{\partial^n f(x_p,y_p,z_p)}{\partial x^n} = \frac{\Delta^n f(x_p,y_p,z_p)_x}{r^n} \tag{18}$$

$$\text{Central Difference}: \frac{\partial^n f(x_p,y_p,z_p)}{\partial x^n} = \frac{\Delta^n f(x_p,y_p,z_p)_x}{(2r)^n} \tag{19}$$

carried out for each direction of the problem as needed. Based on the discretization described above, the nodal equation at node P expresses the unknown nodal value at node

$$f_{i,j,k} = \sum_n \sum_m \sum_l c_{n,m,l} f_{n,m,l} \tag{20}$$

$$c_{n,m,l} = 0 \;\text{ when }\; n = i,\; m = j,\; \text{and}\; l = k$$

P as a linear combination of other unknown nodal values. These nodal equations are then assembled to give a linear system that can be easily solved algebraically. Furthermore, the boundary conditions and other constraints must be specified so as to get a unique solution to the system. Note that the number of nodes used in the grid affects both the accuracy and computation time of the solution.

2.3 Conjugate-Gradient Method

The CGM is an iterative numerical algorithm commonly used to solve linear systems. Unlike some other numerical methods, the starting point (initial solution) of the CGM does not need to be close to the actual solution to guarantee or gain convergence.

$$c = {}^{\lambda_{\max}}\!/_{\lambda_{\min}} \qquad (21)$$

The condition number c (the ratio of the largest to smallest eigen value of a matrix) also affects the rate of convergence of any numerical method to the final solution. For a badly conditioned matrix (i.e. c is very large or $\lambda_{\min}$ is very close to zero) convergence to a solution is very slow and not always guaranteed for most numerical methods. For a badly conditioned matrix, the CGM will reach a solution in at most n steps for a system of order n. By preconditioning the matrix, we can reduce the number of steps required to get to the final solution.

```
$\mathbf{r}_0 = \mathbf{b} - \mathbf{K}\mathbf{x}_0$                 Residual error based on initial guess $\mathbf{x} = \mathbf{x}_0$
$\mathbf{p}_0 = \mathbf{r}_0$                         Initial Direction
for $n = 1:N$
    $\alpha = \mathbf{r}_{n-1}^T\mathbf{r}_{n-1} / \mathbf{p}_{n-1}^T\mathbf{K}\mathbf{p}_{n-1}$     Calculate New Step Size for $\mathbf{x}$ and $\mathbf{r}$
    $\mathbf{x}_n = \mathbf{x}_{n-1} + \alpha * \mathbf{p}_{n-1}$         Calculate New $\mathbf{x}$
    $\mathbf{r}_n = \mathbf{r}_{n-1} - \alpha * \mathbf{K}\mathbf{p}_{n-1}$        Calculate New Residual $\mathbf{r}$
    $\beta = \mathbf{r}_n^T\mathbf{r}_n / \mathbf{r}_{n-1}^T\mathbf{r}_{n-1}$           Calculate New Step Size for $\mathbf{p}$
    $\mathbf{p}_n = \mathbf{r}_n + \beta * \mathbf{p}_{n-1}$            Calculate New Search direction $\mathbf{p}$
*end*
$\mathbf{x} = \mathbf{x}_N$                           Final Result
```

A numerical method usually consists of a series of one or more MVM, vector-inner products (VIP), vector-scalar products (VSP), and scalar operations (SO). For simplicity VIP, VSP, and SO will be referred to as vector-scalar operation (VSO). Since MVMs are more computationally intensive than VSOs, the effective bottleneck of a numerical method is the MVMs. Most numerical methods have two or more MVMs, however, as can be seen above, the CGM has only one MVM. Since the other operations are VSOs, this makes the MVM the single computational bottleneck of the CGM. Speeding up the MVM effectively speeds the CGM.

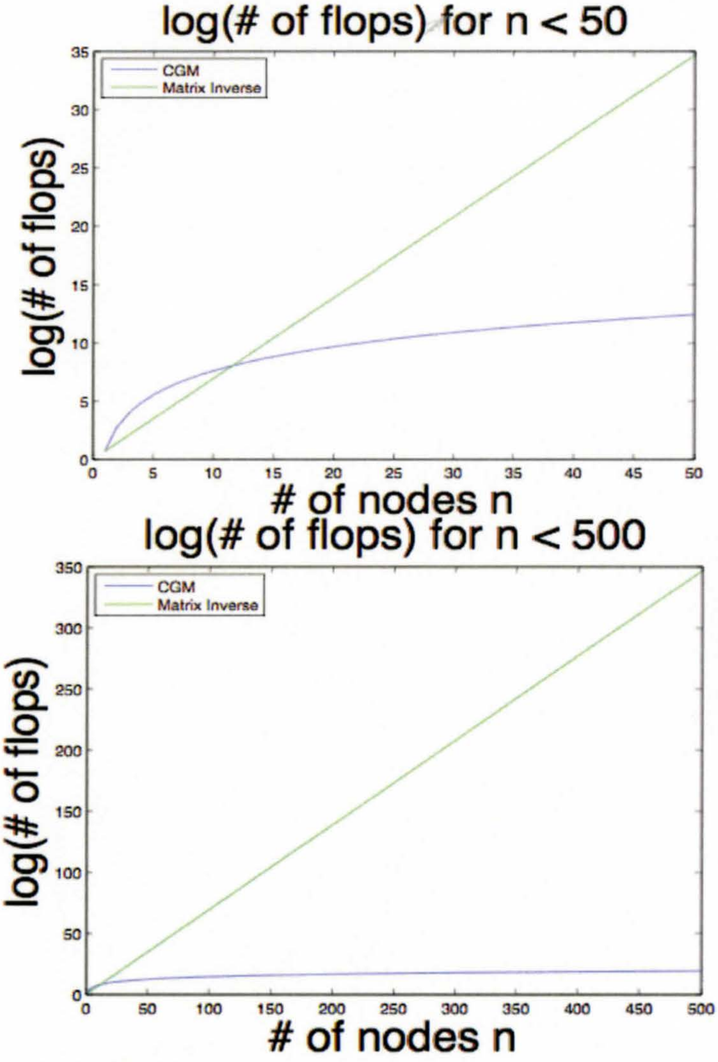
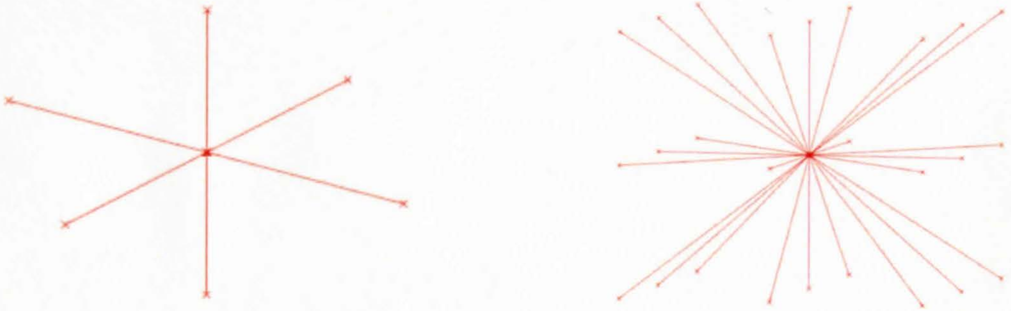


Figure 2-4: Number of flops for CGM and K^{-1}

As mentioned earlier, convergence of the CGM does not depend on the initial guess or solution. This eliminates the computational requirement for choosing a suitable starting point, as would have been the case for some other methods. In the worst-case scenario, the CGM will require $2n^3$ flops. This is very small when compared with the 2^n flops required for calculating a matrix inverse for large n (see Figure 2-4).

2.4 Computational Model

Solving constitutive equations over irregularly shaped materials usually requires the utilization of discretization techniques like the FEM and FDM. Though the FEM is very accurate, it is rather computationally expensive. The FDM on the other hand, is not as accurate but may still give the desired accuracy. In addition to producing a sparser stiffness matrix than the FEM, the FDM is also not as computationally expensive as the FEM. For these reasons we utilize the FDM to generate computational models for the modeling of tissue deformation.



Axial Model (A)

Axial-Shear Model (B)

Figure 2-5: Connection Models

We modeled the tissue as a uniform 3D grid of nodes connected together by a system of springs. The connection among the nodes was modeled with two connection models, which are both shown in Figure 2-5. The first connection model accounts for axial forces and strains, while the second accounts for both axial and shear forces and strains. Recall from Section 2.1 that when an external force acts on a certain node, it causes a change in the length of the springs connected to that node. This in turn creates opposing forces in these springs, which keeps the system of connected springs in equilibrium.

Using (9), the equation governing the relationship among this system of springs, for a given dimension (direction) d, is given in (22).

$$f_{i,j,k}^{d} = \sum_{n=-1}^{1}\sum_{m=-1}^{1}\sum_{l=-1}^{1}\tilde{k}_{i+l,j+m,k+n}u_{i+l,j+m,k+n}^{d} \qquad (22)$$

Where $\tilde{k}_{i+l,j+m,k+n}$ is the elasticity of the spring connecting nodes *i,j,k* and *i+l,j+m,k+n*, $u_{i+l,j+m,k+n}^{d}$ is the change in length of the connecting spring, and $f_{i,j,k}^{d}$ is load acting at node i,j,k in direction d. By replacing $u_{i+l,j+m,k+n}^{d}$ with the equivalent finite-difference equations, shown in (23), which is the difference between displacements of node *i,j,k* and node *i+l,j+m,k+n*, in direction d, (22) can now be written as shown in (24).

$$u_{i+l,j+m,k+n}^{d} = \Delta\tilde{l}_{i+l,j+m,k+n}^{d} = \tilde{l}_{i+l,j+m,k+n}^{d} - \tilde{l}_{i,j,k}^{d} \qquad (23)$$

$$f_{i,j,k}^{d} = \sum_{n=-1}^{1}\sum_{m=-1}^{1}\sum_{l=-1}^{1}\tilde{k}_{i+l,j+m,k+n}\Delta\tilde{l}_{i+l,j+m,k+n}^{d} \qquad (24)$$

Following from (24), (25) and (26) show the nodal equations for connection models A and B in Figure 2-5.

Model A

$$f_{i,j,k}^{d} = \sum_{n=-1}^{1}\sum_{m=-1}^{1}\sum_{l=-1}^{1}\tilde{k}_{i+l,j+m,k+n}\Delta\tilde{l}_{i+l,j+m,k+n}^{d} \qquad (25)$$

$$|l| \neq |m|,\ |l| \neq |n|,\ |n| \neq |m| \quad l,m,n \neq 0$$

Model B

$$f_{i,j,k}^{d} = \sum_{n=-1}^{1}\sum_{m=-1}^{1}\sum_{l=-1}^{1}\tilde{k}_{i+l,j+m,k+n}\Delta\tilde{l}_{i+l,j+m,k+n}^{d} \tag{26}$$

$$l,m,n \neq 0$$

Both (25) and (26) can be written in a simpler form as the inner product of two vectors, $\tilde{\mathbf{k}}_{i,j,k}$ and $\Delta\tilde{\mathbf{l}}^{d}$, shown in (27). Assembling these nodal equations for every node yields

$$f_{i,j,k}^{d} = \tilde{\mathbf{k}}_{i,j,k}{}^{T}\Delta\tilde{\mathbf{l}}^{d} \tag{27}$$

$$\Delta\tilde{\mathbf{l}}^{d} = \left[\Delta\tilde{l}_{1,1,1}^{d},\Delta\tilde{l}_{2,1,1}^{d},\Delta\tilde{l}_{3,1,1}^{d},\Delta\tilde{l}_{4,1,1}^{d},..,\Delta\tilde{l}_{N,1,1}^{d},\Delta\tilde{l}_{1,2,1}^{d},\Delta\tilde{l}_{2,2,1}^{d},\Delta\tilde{l}_{3,2,1}^{d},..\right]^{T}$$

a set of linear simultaneous equations that describes the system in direction d. These equations can be represented in matrix form as shown in (28).

$$\mathbf{f}^{d} = \tilde{\mathbf{K}}\,\Delta\tilde{\mathbf{l}}^{d} \tag{28}$$

$$\tilde{\mathbf{K}}_{i,j,k} = \left[\tilde{\mathbf{k}}_{1,1,1},\tilde{\mathbf{k}}_{2,1,1},\tilde{\mathbf{k}}_{3,1,1},\tilde{\mathbf{k}}_{4,1,1},....,\tilde{\mathbf{k}}_{N,1,1},\tilde{\mathbf{k}}_{1,2,1},\tilde{\mathbf{k}}_{2,2,1},\tilde{\mathbf{k}}_{3,2,1},.....\right]^{T}$$
$$\mathbf{f}^{d} = \left[f_{1,1,1}^{d},f_{2,1,1}^{d},f_{3,1,1}^{d},f_{4,1,1}^{d},....,f_{N,1,1}^{d},f_{1,2,1}^{d},f_{2,2,1}^{d},f_{3,2,1}^{d},.....\right]^{T}$$

We can write $\Delta\tilde{l}_{i+l,j+m,k+n}^{d}$ as the inner product of two vectors $\tilde{\mathbf{p}}_{i+l,j+m,k+n}$ and $\tilde{\mathbf{l}}^{d}$, where $\tilde{\mathbf{l}}^{d}$ is the displacement vector and $\tilde{l}_{i,j,k}^{d}$ and $\tilde{l}_{i+l,j+m,k+n}^{d}$ are elements of $\tilde{\mathbf{l}}^{d}$ that correspond to nodes *i,j,k* and node *i+l,j+m,k+n* respectively. Vector $\tilde{\mathbf{p}}_{i+l,j+m,k+n}$ consists of the coefficients required to determine $\Delta\tilde{l}_{i+l,j+m,k+n}^{d}$ for the spring connecting nodes *i,j,k* and node *i+l,j+m,k+n*.

$$\Delta\tilde{\mathbf{l}}^d = \tilde{\mathbf{P}}\,\tilde{\mathbf{l}}^d \tag{29}$$

$$\tilde{\mathbf{l}}^d = \left[\tilde{l}^d_{1,1,1}, \tilde{l}^d_{2,1,1}, \tilde{l}^d_{3,1,1}, \tilde{l}^d_{4,1,1}, \ldots, \tilde{l}^d_{N,1,1}, \tilde{l}^d_{1,2,1}, \tilde{l}^d_{2,2,1}, \tilde{l}^d_{3,2,1}, \ldots\right]^T$$
$$\tilde{\mathbf{P}} = \left[\tilde{\mathbf{p}}_{1,1,1}, \tilde{\mathbf{p}}_{2,1,1}, \tilde{\mathbf{p}}_{3,1,1}, \tilde{\mathbf{p}}_{4,1,1}, \ldots., \tilde{\mathbf{p}}_{N,1,1}, \tilde{\mathbf{p}}_{1,2,1}, \tilde{\mathbf{p}}_{2,2,1}, \tilde{\mathbf{p}}_{3,2,1}, \ldots..\right]^T$$

The combination of these inner products for the system of springs will yield a system of linear equations that can be represented in matrix form as shown in (29).

$$\begin{aligned}\mathbf{f}^d &= \tilde{\mathbf{K}}\,\tilde{\mathbf{P}}\,\tilde{\mathbf{l}}^d \\ &= \mathbf{K}\tilde{\mathbf{l}}^d \end{aligned} \tag{30}$$

$$\mathbf{K} = \tilde{\mathbf{K}}\,\tilde{\mathbf{P}}$$

The combination of (28) and (29) produces the equation shown in (30), where $\mathbf{K}$ is the stiffness matrix that describes the system of connected nodes and springs. For a homogeneous tissue, the same elasticity $\tilde{k}$ is used for the springs in the connection models A and B. Note that (27), (28) and (30) can be written as (31), (32) and (33) respectively.

$$f^d_{i,j,k} = \tilde{k}(\mathbf{v}_{i,j,k})^T \Delta\tilde{\mathbf{l}}^d \tag{31}$$

$$\tilde{\mathbf{k}}_{i,j,k} = \tilde{k}\,\mathbf{v}_{i,j,k}$$

$$\mathbf{f}^d = \tilde{k}\,\tilde{\mathbf{A}}\,\Delta\tilde{\mathbf{l}}^d \tag{32}$$

$$\tilde{\mathbf{A}} = \left[\mathbf{v}_{1,1,1}, \mathbf{v}_{2,1,1}, \mathbf{v}_{3,1,1}, \mathbf{v}_{4,1,1}, \ldots., \mathbf{v}_{N,1,1}, \mathbf{v}_{1,2,1}, \mathbf{v}_{2,2,1}, \mathbf{v}_{3,2,1}, \ldots..\right]^T$$
$$\tilde{\mathbf{K}} = \tilde{k}\,\tilde{\mathbf{A}}$$

$$\mathbf{f}^d = \tilde{k}\,\mathbf{A}\,\tilde{\mathbf{l}}^d \tag{33}$$

$$\mathbf{A} = \tilde{\mathbf{A}}\,\tilde{\mathbf{P}}$$
$$\mathbf{K} = \tilde{k}\,\mathbf{A}$$

2.4.1 Linear Model

In this section, we look at a computational model to determine the deformation of a tissue with known elasticity, when a load is acting on it. The deformation of a tissue is a measure of the change in the position, or displacements $\tilde{l}^d$, of the nodes in the grid used to model the tissue. Thus, given the required boundary conditions, the deformation of the tissue in question is given by the displacement values that solve the system in (30) in each direction d. In other words $\tilde{\mathbf{l}}^d$ that minimizes Π^d, in (34), must be determined in

$$\Pi^d = \mathbf{K}\,\tilde{\mathbf{l}}^d - \mathbf{f}^d \qquad (34)$$

each direction d, for specified constraints and boundary conditions, in order to get the required deformations for the whole system. The required minimization can be carried out using the CGM.

2.4.2 Time Dependent Model

In this section, we develop a time dependent computational model for carrying out real-time modeling for the deformation of a tissue with known elasticity. To do this, we use a quasi-static approach in combination with a variation of the system shown in (34), while $\mathbf{f}^d$ changes over time. In the quasi-static approach, we take a sample, $\mathbf{f}_t^d$, of $\mathbf{f}^d$ at certain time intervals (or time-steps), then use this sampled value to solve for the corresponding displacement values, $\tilde{\mathbf{l}}_t^d$. This process is repeated at every time step.

Recall that the stiffness matrix $\mathbf{K}$ depends on the elasticities of the springs used to model the tissue. As such, $\mathbf{K}$ will depend on $\tilde{\mathbf{l}}^d$ when these elasticities are dependent on the $\tilde{\mathbf{l}}^d$ (displacement dependent model). Obviously this not the case when the spring elasticities are constant (displacement independent model). This difference also extends to the time dependent computational models developed from both models.

Displacement Dependent Model

When $\mathbf{K}$ depends on $\tilde{\mathbf{l}}^d$, the dependence means that $\mathbf{K}$ changes at each time step, thus the computational models that is used must update $\mathbf{K}$ at each time step using the most recent displacement values $\tilde{\mathbf{l}}_{t-1}^d$ (i.e. the solution to the previous time step). The computational model, in this case, is to minimize Π_t^d given in (35) at each time step

$$\Pi_t^d = \mathbf{K}_t \Delta \tilde{\mathbf{l}}_t^d - \Delta \mathbf{f}_t^d \tag{35}$$

for a given set of constraints and boundary conditions, provided the relationship between $\mathbf{K}$ and $\tilde{\mathbf{l}}^d$ is known. Note that $\Delta\tilde{\mathbf{l}}_t^d$ and $\Delta\mathbf{f}_t^d$, the changes in $\tilde{\mathbf{l}}^d$ and $\mathbf{f}^d$ from values at the previous step, along with the updated stiffness matrix, $\mathbf{K}_t$, are used in (35). Rewriting

$$\begin{aligned}
\Pi_t^d &= \mathbf{K}_t(\tilde{\mathbf{l}}_t^d - \tilde{\mathbf{l}}_{t-1}^d) - (\mathbf{f}_t^d - \mathbf{f}_{t-1}^d) \\
\Pi_t^d &= \mathbf{K}_t\tilde{\mathbf{l}}_t^d - \mathbf{K}_t\tilde{\mathbf{l}}_{t-1}^d - \mathbf{f}_t^d + \mathbf{f}_{t-1}^d \\
\Pi_t^d &= \mathbf{K}_t\tilde{\mathbf{l}}_t^d - \mathbf{h}_t^d \qquad (36)
\end{aligned}$$

$$\mathbf{h}_t^d = \mathbf{K}_t\tilde{\mathbf{l}}_{t-1}^d + \mathbf{f}_t^d - \mathbf{f}_{t-1}^d$$

(35) in the expanded form seen in (36) shows that the sampled value $\mathbf{f}_{t-1}^d$ and solution $\tilde{\mathbf{l}}_{t-1}^d$ at the previous time step need to be stored.

Displacement Independent Model

When $\mathbf{K}$ does not depend on $\tilde{\mathbf{l}}^d$, the independence eliminates the need to update $\mathbf{K}$ as it is constant. Given the required set of constraints and boundary conditions, the computational model in this case is to minimize (36) at each time step using the new value, $\mathbf{f}_t^d$, of $\mathbf{f}^d$ and a constant stiffness matrix. In other words, $\mathbf{K}_t$ never changes. Further, the solution, $\tilde{\mathbf{l}}_{t-1}^d$, at the previous time step is used as the initial solution for the current time step.

Of the two models described above, the later is a more efficient way of carrying out real-time modeling. For example, take the updating of $\mathbf{K}$ in solving (35). Though $\mathbf{K}$ may be a very sparse matrix, the required update of $\mathbf{K}$ is inefficient given that the computational model based on (34) will not require any update of $\mathbf{K}$. In addition to this, the use of, $\tilde{\mathbf{I}}^d_{t-1}$, the solution to the previous time step by the second model as the initial solution to the current time step reduces the time and number of iterations needed to converge to the new solution, as $\tilde{\mathbf{I}}^d_{t-1}$ is the closest known value to the solution, $\tilde{\mathbf{I}}^d_t$, at time *t*. For these reasons the displacement independent model, in combination with the CGM, is the better choice for carrying out the real-time modeling of tissue deformation in each direction given the necessary boundary conditions.

```
$t = 1$
$\tilde{\mathbf{I}}^d_{0,N} = \tilde{\mathbf{I}}^d_I$
while $t \geq 1$
    $\tilde{\mathbf{I}}^d_{t,0} = \tilde{\mathbf{I}}^d_{t-1,N}$
    $\mathbf{r}_0 = \mathbf{h}^d_t - \mathbf{K}\,\tilde{\mathbf{I}}^d_{t,0}$                Residual error based on initial guess $\tilde{\mathbf{I}}^d_{t,0}$
    $\mathbf{p}_0 = \mathbf{r}_0$                                  Initial Direction
    for $n = 1:N$
        $\alpha = \mathbf{r}^T_{n-1}\mathbf{r}_{n-1} / \mathbf{p}^T_{n-1}\mathbf{K}\mathbf{p}_{n-1}$    Calculate New Step Size for d and r
        $\tilde{\mathbf{I}}^d_{t,n} = \tilde{\mathbf{I}}^d_{t,n-1} + \alpha * \mathbf{p}_{n-1}$        Calculate New Displacement $\tilde{\mathbf{I}}^d$
        $\mathbf{r}_n = \mathbf{r}_{n-1} - \alpha * \mathbf{K}\mathbf{p}_{n-1}$        Calculate New Residual r
        $\beta = \mathbf{r}^T_n\mathbf{r}_n / \mathbf{r}^T_{n-1}\mathbf{r}_{n-1}$             Calculate New Step Size for p
        $\mathbf{p}_n = \mathbf{r}_n + \beta * \mathbf{p}_{n-1}$             Calculate New Search direction p
    end
    $\tilde{\mathbf{I}}^d = \tilde{\mathbf{I}}^d_{t,N}$                                 Final Result at time step $t$
    $t = t + 1$
end
```

This combination gives the algorithm, a modified version of the CGM, shown above. Note that the feasibility of this algorithm, for its real-time modeling purpose, requires that the CGM must be executed fast enough to meet the time requirements. In order to meet the time requirements for real-time modeling, the CGM must be accelerated. Our approach to CGM acceleration is discussed in Chapter 4.

2.5 Properties of the Stiffness Matrix

In section 2.4, we generated a linear model and a time-dependent model for modeling tissue deformation, using a system with constant stiffness matrix $\mathbf{K}$. The properties of $\mathbf{K}$ are discussed in this section. These properties are as follows:

i) The stiffness matrix generated from the method above is always a sparse square band diagonal matrix, as shown below in Figure 2-6A.

ii) The Band of $\mathbf{K}$ changes with the connection model in Figure 2-5 that is used.

iii) The model being used also determines the number of nodes that are connected together and the sparsity of the stiffness matrix $\mathbf{K}$. The sparsity of $\mathbf{K}$ defines the number of zero elements in $\mathbf{K}$. It is calculated as a percentage of the total number elements in matrix $\mathbf{K}$. This value decreases as the number of nodes, n, increases since the models that are used limits and fixes the maximum

$$\begin{aligned} \mathit{Spar}(\mathbf{K}) &= 1-(\mathit{total\ \#\ of\ non-zero\ elements/total\ \#\ of\ \ elements\ in}\ \mathbf{K}) \\ &\approx 1-(\mathit{maximum\ \#\ of\ non-zero\ elements\ per\ row/\#\ of\ \ columns\ in}\ \mathbf{K}) \\ &\approx 1-(\mathit{maximum\ \#\ of\ non-zero\ elements\ per\ row}/\mathrm{n}) \qquad (37) \end{aligned}$$

number of nonzero elements per row (via the connection between nodes). On the other hand, the sparcity of $\mathbf{K}$ will decrease by changing from model A to model B. For example, if n = 500 the sparcity of $\mathbf{K}$ will decrease from 98.6% to 94.6%. This is simply because more adjacent nodes are connected together while the total number of nodes is fixed.

iv) Recall from section 2.4, when the same elasticity value is used for all the springs, in the connection models, the stiffness matrix, **K**, can be factorized as shown in (38). Therefore, properties i) to iv) are effectively those of matrix A (see (33)).

$$\mathbf{K} = \tilde{k}\,\mathbf{A} \qquad (38)$$

v) Also, **K** that describe an isotropic system is a symmetric matrix

vi) When a uniform grid with an even number of nodes is used, the stiffness matrix **K** for a homogeneous system is not only symmetric; it also has the structure shown in figure 2-6B. Matrix **B** is a non-square-symmetric band diagonal matrix, while **B*** is the resulting matrix from rotating **B** 180 degrees. We call this conjugate symmetry.

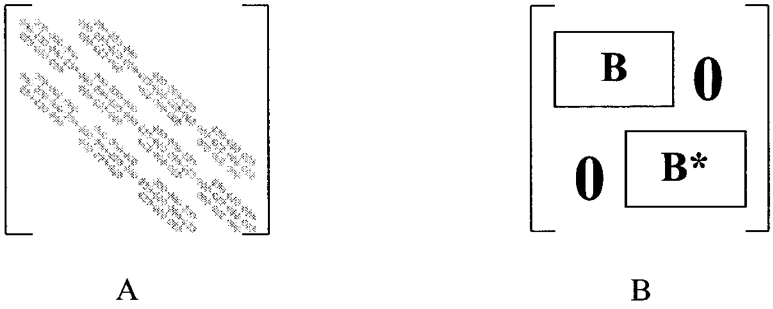
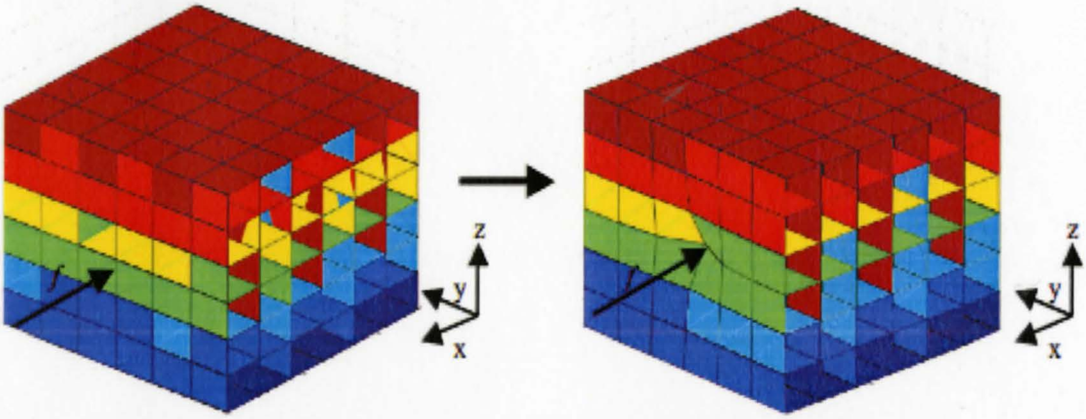


Figure 2-6: Stiffness Matrix Structure

2.6 Convergence Analysis

The ability to carry out real-time modeling is also affected by the rate of convergence of the computational model being used. A computational model with a fast rate of convergence will require fewer iterations to converge to a solution, thus improving the ability to meet the time requirements for real-time modeling. In this section, we focus on the computational models that require the minimization of (34). We use a numerical example to show the rate of convergence of these computational models. We use the CGM to minimize (34) in order to determine the deformation of a cube of tissue. We assume that the eight corners of the tissue do not move. The elasticity values in the x-, y-, and z- directions are set to 1 N/m, 2 N/m, and 0.5 N/m respectively. A force of 1N is then applied to center of the y-z surface of the tissue. The force is applied perpendicularly to this surface (x-direction). We used MATLAB to solve for the nodal displacements in x-direction for systems generated from connection models A and B in Figure 2-5.



Undeformed tissue

Deformed tissue

Figure 2-7: Deformed Tissue

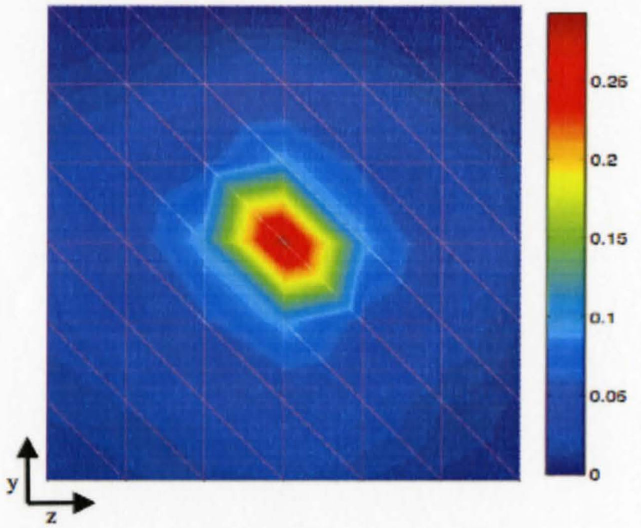
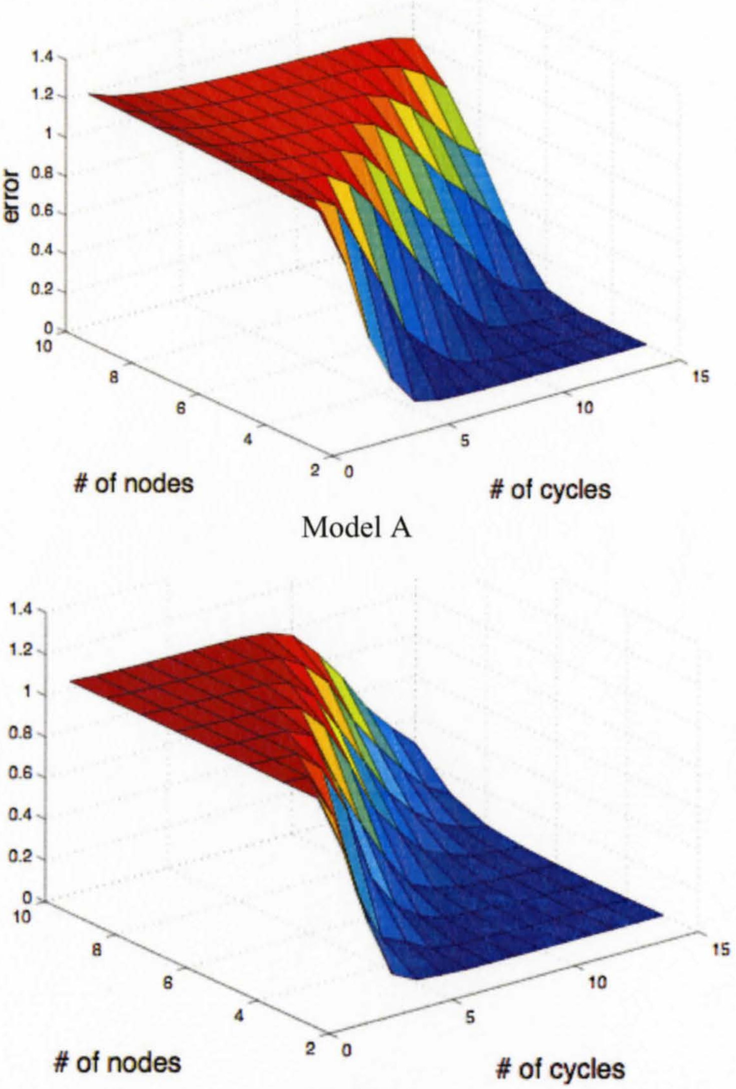


Figure 2-8: Displacements of boundary nodes on y-z plane

Figure 2-7 shows the deformation of a tissue before and after the force is applied. Figure 2-8 shows the displacements in the x-direction for the boundary nodes on the y-z plane where the force is applied.



Model A

Model B

Figure 2-9: CGM Error for connection Models A and B

We took the deviation of the solution from the actual displacement values in the x-direction at each node. The error in the solution is then calculated by taking the ∞-norm of the resulting deviation vector. The graph of error vs. system size vs. number of CGM cycles was plotted for both connection models (see Figure 2-9). As seen in Figure 2-9, model B converges faster than model A. Recall from Section 2.3 that the rate of convergence of the CGM depends on **K**. Thus, a better rate of convergence is also achieved by model B in the y-, and z-directions.

3 INVERSE MODEL

In Chapter 2, we derived computational models for determining the deformation of a tissue with known elasticity. In reality, the tissue elasticity is not always known. Thus an estimated value, calculated using estimation techniques such as least-squares (LS) estimate or maximum-likelihood (ML) estimate, is usually used. In this Chapter, we look at parametric and non-parametric computational models for estimating the elasticity of a tissue using a LS estimate. In each case, we look for elasticity values that provide the best fit for to a set of observed deformation and load values, which are measured using a uniform grid of sensors placed at certain nodes on the tissue.

3.1 Least-Squares Estimate

This is a very popular estimation technique that uses a set of observed physical quantity q to estimate a set of parameters θ, which in turn are used by a predicting model $\tilde{q}(x,\theta)$ to predict q. The LS estimate attempts to minimizing the distance between these observed values of q and its model predicted values, as these observed values have certain errors attached to them. In other words, the LS estimate provides the parameter estimate that best fits the observed values of q. Unlike most estimation techniques where information, usually assumptions, about the distribution family of q is needed, the LS estimate does not require any information about the distribution family of q.

Let q_i and x_i denote the ith observation of q and observation point respectively. Also, let $\tilde{q}_i(\theta)$ represent the model predicted value at observation point x_i. In other words, $\tilde{q}_i(\theta)$ is equivalent to $\tilde{q}(x_i,\theta)$. Let $\mathbf{q}$ and $\tilde{\mathbf{q}}(\theta)$ be the corresponding set (vectors) of observed and model predicted values q_i and $\tilde{q}_i(\theta)$ respectively. The LS estimate, $\hat{\theta}$, minimizes the squared 2-norm of the difference (or error) between the vectors $\mathbf{q}$ and $\tilde{\mathbf{q}}(\theta)$ as shown in (39). To minimize $\|\mathbf{q} - \tilde{\mathbf{q}}(\theta)\|_2^2$, its derivative is taken with respect to θ, the set of

$$\hat{\theta} = \arg\min_{\theta} \|\mathbf{q} - \tilde{\mathbf{q}}(\theta)\|_2^2 \tag{39}$$

parameters that are to be estimated, and equated to 0 as shown in (40). This gives a set of equations, one for each element of θ, that can be solved through the use of numerical techniques such as Newton's method, conjugate gradient method, or others.

$$\frac{\partial \|\mathbf{q} - \tilde{\mathbf{q}}(\theta)\|_2^2}{\partial \theta} = \frac{\partial ((\mathbf{q} - \tilde{\mathbf{q}}(\theta))^T (\mathbf{q} - \tilde{\mathbf{q}}(\theta)))}{\partial \theta} = 0 \tag{40}$$

The LS estimate, $\hat{\theta}$, discussed so far is based on one set of observed values. In the general case, where there are n sets of observed values, for $n \geq 1$, $\hat{\theta}$ is gotten by minimizing the summation of the squared 2-norm of the difference between the vectors $\mathbf{q}_k$ and $\tilde{\mathbf{q}}(\theta)$. $\mathbf{q}_k$ is the corresponding vector to the kth set of observed values q_i. Hence, $\hat{\theta}$ is given by the solution to (41).

$$\begin{aligned} \frac{\partial \sum_{k=1}^{n} \|\mathbf{q}_k - \tilde{\mathbf{q}}(\theta)\|_2^2}{\partial \theta} &= \frac{\partial \sum_{k=1}^{n} ((\mathbf{q}_k - \tilde{\mathbf{q}}(\theta))^T (\mathbf{q}_k - \tilde{\mathbf{q}}(\theta)))}{\partial \theta} \\ &= \sum_{k=1}^{n} \frac{\partial ((\mathbf{q}_k - \tilde{\mathbf{q}}(\theta))^T (\mathbf{q}_k - \tilde{\mathbf{q}}(\theta)))}{\partial \theta} \end{aligned} \tag{41}$$

3.2 Estimating Elasticity of a Homogeneous Tissue

In this section, we discuss computational models for estimating tissue elasticity. Recall that the tissue is modeled as a grid of nodes connected together by a system of springs.

3.2.1 Non-Parametric Model

In the case where all the springs in connection models A and B have the same elasticity value, the problem is to estimate the scalar quantity $\tilde{k}$, which best fit the set of observed deformations and load values.

$$\Pi = \sum_{d=1}^{3} \left\| \mathbf{f}^{d} - \tilde{k}\mathbf{A}\,\tilde{\mathbf{l}}^{d} \right\|_2^2 \tag{42}$$

Recall from section 2.4 that the deformation is a measure of nodal displacements due to an acting load. Therefore, the displacement and load values are observed at each node in each direction. These values, for a direction d, are given by displacement and load vectors $\tilde{\mathbf{l}}^{d}$ and $\mathbf{f}^{d}$ respectively. The LS estimate $\hat{k}$, is the value of $\tilde{k}$ that minimizes (42).

$$\begin{aligned}
\frac{\partial \sum_{d=1}^{3} \left\| \mathbf{f}^{d} - \tilde{k}\mathbf{A}\,\tilde{\mathbf{l}}^{d} \right\|_2^2}{\partial \tilde{k}} &= \sum_{d=1}^{3} \frac{\partial((\mathbf{f}^{d} - \tilde{k}\mathbf{A}\,\tilde{\mathbf{l}}^{d})^{T}(\mathbf{f}^{d} - \tilde{k}\mathbf{A}\,\tilde{\mathbf{l}}^{d}))}{\partial \tilde{k}} \\
&= \sum_{d=1}^{3} \frac{\partial(\mathbf{f}^{d^T}\mathbf{f}^{d} - 2\tilde{k}\mathbf{f}^{d^T}\mathbf{A}\,\tilde{\mathbf{l}}^{d} + \tilde{k}^{2}\tilde{\mathbf{l}}^{d^T}\mathbf{A}^{T}\mathbf{A}\,\tilde{\mathbf{l}}^{d})}{\partial \tilde{k}} \\
&= \sum_{d=1}^{3} 2(\tilde{\mathbf{l}}^{d^T}\mathbf{A}^{T}\mathbf{A}\,\tilde{\mathbf{l}}^{d})\tilde{k} - 2\mathbf{f}^{d^T}\mathbf{A}\,\tilde{\mathbf{l}}^{d} \\
&= \sum_{d=1}^{3} 2\tilde{a}^{d}\tilde{k} - 2\,\tilde{f}^{d}
\end{aligned} \tag{43}$$

$$\begin{aligned}
\tilde{a}^{d} &= \tilde{\mathbf{l}}^{d^T}\mathbf{A}^{T}\mathbf{A}\,\tilde{\mathbf{l}}^{d} \\
\tilde{f}^{d} &= \mathbf{f}^{d^T}\mathbf{A}\,\tilde{\mathbf{l}}^{d}
\end{aligned}$$

Hence, equating (43) to zero gives the required estimate, $\hat{k}$ (see (44)). $\sum_{d=1}^{3}$ represents the summation of values in the x-, y-, and z-directions, where the values 1, 2, and 3, for d, represent values in the x-, y-, and z-directions respectively.

$$\hat{k} = \frac{\sum_{d=1}^{3} \tilde{f}^{d}}{\sum_{d=1}^{3} \tilde{a}^{d}} \tag{44}$$

3.2.2 Parametric Model

In Section 3.2.1, we assumed that the elasticity values of the spring are constant. In this Section, we discuss a computational model that describes the relationship between deformation and elasticity of a spring. The elasticity of each spring, in the connection model, is given by $\tilde{k}_{i+l,j+m,k+n} = \alpha_0 + \sum_{d=1}^{3} \alpha_1 \Delta\tilde{l}^{d}_{i+l,j+m,k+n} + \sum_{d=1}^{3} \alpha_2 (\Delta\tilde{l}^{d}_{i+l,j+m,k+n})^2$. We look at a computational model that can be used to determine the elasticity parameters α_0, α_1, and α_2 from deformation values.

$$\tilde{k}_{i+l,j+m,k+n} = \mathbf{w}^{T}_{i+l,j+m,k+n}\mathbf{a} \tag{45}$$

$$\mathbf{a} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{bmatrix}, \ \mathbf{w}_{i+l,j+m,k+n} = \begin{bmatrix} 1 \\ \sum_{d=1}^{3} \Delta\tilde{l}^{d}_{i+l,j+m,k+n} \\ \sum_{d=1}^{3} (\Delta\tilde{l}^{d}_{i+l,j+m,k+n})^2 \end{bmatrix}$$

The elasticity of each spring can be written as the inner product of two vectors as shown in (45). Further, the elasticity values of the system of springs can be represented in the matrix form shown in (46).

$$\mathbf{k} = \mathbf{W}\mathbf{a} \tag{46}$$

$$\mathbf{k} = \left[\tilde{k}_{1,1,1}, \tilde{k}_{2,1,1}, \tilde{k}_{3,1,1}, \tilde{k}_{4,1,1}, \ldots, \tilde{k}_{N,1,1}, \tilde{k}_{1,2,1}, \tilde{k}_{2,2,1}, \tilde{k}_{3,2,1}, \ldots\right]^T$$
$$\mathbf{W} = \left[\mathbf{w}_{1,1,1}, \mathbf{w}_{2,1,1}, \mathbf{w}_{3,1,1}, \mathbf{w}_{4,1,1}, \ldots, \mathbf{w}_{N,1,1}, \mathbf{w}_{1,2,1}, \mathbf{w}_{2,2,1}, \mathbf{w}_{3,2,1}, \ldots\right]^T$$

We can rewrite $\tilde{\mathbf{k}}_{i,j,k}$ (from (27)) as the matrix-by-vector product between a square-symmetric matrix $\mathbf{B}_{i,j,k}$ and vector $\mathbf{k}$. Substituting this product into (27), along with (29), yields (47).

$$\begin{aligned} f^d_{i,j,k} &= (\mathbf{B}_{i,j,k}\mathbf{k})^T \Delta\tilde{\mathbf{I}}^d \\ &= (\mathbf{B}_{i,j,k}\mathbf{k})^T \tilde{\mathbf{P}}\,\tilde{\mathbf{I}}^d \\ &= \mathbf{k}^T\mathbf{B}^T_{i,j,k}\tilde{\mathbf{P}}\,\tilde{\mathbf{I}}^d \\ &= \mathbf{k}^T\tilde{\mathbf{C}}_{i,j,k}\tilde{\mathbf{I}}^d \end{aligned} \tag{47}$$

$$\tilde{\mathbf{C}}_{i,j,k} = \mathbf{B}^T_{i,j,k}\tilde{\mathbf{P}}$$

While (46) gives the relationship between $\mathbf{a}$ and $\mathbf{k}$, (47) gives the relationship between $f^d_{i,j,k}$ and $\mathbf{k}$. The combination of (46) and (47) yields (48), which gives the relationship between $\mathbf{a}$ and $f^d_{i,j,k}$.

$$f^d_{i,j,k} = \mathbf{a}^T\mathbf{C}_{i,j,k}\,\tilde{\mathbf{I}}^d \tag{48}$$

$$\mathbf{C}_{i,j,k} = \mathbf{W}^T\tilde{\mathbf{C}}_{i,j,k}$$

The operands, $f^d_{i,j,k}$, $\mathbf{C}_{i,j,k}$ and $\tilde{\mathbf{I}}^d$, are observed or calculated directly from the observed values, thus making $\mathbf{a}$ the only unknown in (48). We get the LS estimate, $\hat{\mathbf{a}}$, for $\mathbf{a}$, by minimizing (49) with respect to $\mathbf{a}$.

$$\Pi = \sum_{d=1}^{3}\sum_{k}\sum_{j}\sum_{i}(f^d_{i,j,k} - \mathbf{a}^T\mathbf{C}_{i,j,k}\,\tilde{\mathbf{I}}^d)^2 \tag{49}$$

The CGM can be used to obtain the solution to (50), thereby yielding the parameters $\hat{\mathbf{a}}$ that describe the relationship between deformation and elasticity of the springs used in modeling the tissue. Recall that $\tilde{\mathbf{l}}^d$ and $\mathbf{f}^d$ are displacement and load vectors of observed displacement and load values at each node in direction d.

$$\frac{\partial \sum_{d=1}^{3}\sum_{k}\sum_{j}\sum_{i}(f^d_{i,j,k} - \mathbf{a}^T\mathbf{C}_{i,j,k}\,\tilde{\mathbf{l}}^d)^2}{\partial \mathbf{a}} = \frac{\partial \sum_{d=1}^{3}\sum_{k}\sum_{j}\sum_{i}(f^d_{i,j,k})^2 - 2(\mathbf{a}^T\mathbf{C}_{i,j,k}\,\tilde{\mathbf{l}}^d)f^d_{i,j,k} + (\mathbf{a}^T\mathbf{C}_{i,j,k}\,\tilde{\mathbf{l}}^d)^2}{\partial \mathbf{a}}$$

$$= \frac{\partial \sum_{d=1}^{3}\sum_{k}\sum_{j}\sum_{i}(f^d_{i,j,k})^2 - 2(\mathbf{a}^T\mathbf{g}^d_{i,j,k})f^d_{i,j,k} + \mathbf{a}^T\mathbf{g}^d_{i,j,k}\mathbf{g}^{d^T}_{i,j,k}\mathbf{a}}{\partial \mathbf{a}}$$

$$= \sum_{d=1}^{3}\sum_{k}\sum_{j}\sum_{i}2\mathbf{g}^d_{i,j,k}\mathbf{g}^{d^T}_{i,j,k}\mathbf{a} - 2\mathbf{g}^d_{i,j,k}f^d_{i,j,k}$$

$$= \sum_{d=1}^{3}2\mathbf{G}^d\mathbf{G}^{d^T}\mathbf{a} - 2\mathbf{G}^d\,\mathbf{f}^d$$

$$= \mathbf{Ha} - \widehat{\mathbf{f}} \qquad (50)$$

$$\mathbf{g}^d_{i,j,k} = \mathbf{C}_{i,j,k}\,\tilde{\mathbf{l}}^d\;,\; \mathbf{G}^d = \left[\mathbf{g}^d_{1,1,1},\mathbf{g}^d_{2,1,1},\mathbf{g}^d_{3,1,1},\mathbf{g}^d_{4,1,1},....,\mathbf{g}^d_{N,1,1},\mathbf{g}^d_{1,2,1},\mathbf{g}^d_{2,2,1},\mathbf{g}^d_{3,2,1},.....\right]^T$$

$$\widehat{\mathbf{f}} = \sum_{d=1}^{3}2\mathbf{G}^d\mathbf{f}^d,\; \mathbf{H} = \sum_{d=1}^{3}2\mathbf{G}^d\mathbf{G}^{d^T}$$

Recall from section 3.2.1 that the values 1, 2, and 3, for d, represent values in the x-, y-, and z-directions respectively. Using the LS estimate $\hat{\mathbf{a}}$, the elasticity values of the system of springs are now given by the system in (51) or the parametric model in (52).

$$\mathbf{k} = \mathbf{W}\hat{\mathbf{a}} \qquad (51)$$

$$\tilde{k}_{i+l,j+m,k+n} = \hat{\alpha}_0 + \sum_{d=1}^{3}\hat{\alpha}_1\Delta\tilde{l}^d_{i+l,j+m,k+n} + \sum_{d=1}^{3}\hat{\alpha}_2\Delta(\tilde{l}^d_{i+l,j+m,k+n})^2 \qquad (52)$$

3.3 Estimating Elasticity of a Non-Homogeneous Tissue

The springs used in the connection models may not necessarily have the same elasticity values. In other to estimate these elasticity values, we use a computational model that is generated from (47). The aim is to estimate the values of $\mathbf{k}$, which provide the best fit to a set of observed deformation and load values by minimizing (53) with respect to $\mathbf{k}$.

$$\Pi = \sum_{d=1}^{3}\sum_{k}\sum_{j}\sum_{i}(f_{i,j,k}^{d} - \mathbf{k}^{T}\tilde{\mathbf{C}}_{i,j,k}\,\tilde{\mathbf{l}}^{d})^{2} \qquad (53)$$

Where $f_{i,j,k}^{d}$, $\tilde{\mathbf{C}}_{i,j,k}^{T}$ and $\tilde{\mathbf{l}}^{d}$, can be observed or calculated from the observed values. The LS estimate, $\hat{\mathbf{k}}$, are the values of $\mathbf{k}$ that minimizes (53).

The solution to (54) gives the required LS estimate, $\hat{\mathbf{k}}$'s that describe the elasticity values of the system of springs. Recall that $\tilde{\mathbf{l}}^{d}$ and $\mathbf{f}^{d}$ are displacement and load vectors

$$\begin{aligned}
\frac{\partial\sum_{d=1}^{3}\sum_{k}\sum_{j}\sum_{i}(f_{i,j,k}^{d} - \mathbf{k}^{T}\tilde{\mathbf{C}}_{i,j,k}\,\tilde{\mathbf{l}}^{d})^{2}}{\partial\mathbf{k}} &= \frac{\partial\sum_{d=1}^{3}\sum_{k}\sum_{j}\sum_{i}(f_{i,j,k}^{d})^{2} - 2(\mathbf{k}^{T}\tilde{\mathbf{C}}_{i,j,k}\,\tilde{\mathbf{l}}^{d})f_{i,j,k}^{d} + (\mathbf{k}^{T}\tilde{\mathbf{C}}_{i,j,k}\,\tilde{\mathbf{l}}^{d})^{2}}{\partial\mathbf{k}} \\
&= \frac{\partial\sum_{d=1}^{3}\sum_{k}\sum_{j}\sum_{i}(f_{i,j,k}^{d})^{2} - 2(\mathbf{k}^{T}\tilde{\mathbf{g}}_{i,j,k}^{d})f_{i,j,k}^{d} + \mathbf{k}^{T}\tilde{\mathbf{g}}_{i,j,k}^{d}\tilde{\mathbf{g}}_{i,j,k}^{d^{T}}\mathbf{k}}{\partial\mathbf{k}} \\
&= \sum_{d=1}^{3}\sum_{k}\sum_{j}\sum_{i}2\tilde{\mathbf{g}}_{i,j,k}^{d}\tilde{\mathbf{g}}_{i,j,k}^{d^{T}}\mathbf{k} - 2\tilde{\mathbf{g}}_{i,j,k}^{d}f_{i,j,k}^{d} \\
&= \sum_{d=1}^{3}2\tilde{\mathbf{G}}^{d}\tilde{\mathbf{G}}^{d^{T}}\mathbf{k} - 2\tilde{\mathbf{G}}^{d}\,\mathbf{f}^{d} \\
&= \sum_{d=1}^{3}2\tilde{\mathbf{H}}\mathbf{k} - 2\tilde{\mathbf{f}}^{d} \qquad (54)
\end{aligned}$$

$$\tilde{\mathbf{g}}_{i,j,k}^{d} = \tilde{\mathbf{C}}_{i,j,k}\,\tilde{\mathbf{l}}^{d}\ ,\ \tilde{\mathbf{G}}^{d} = \left[\tilde{\mathbf{g}}_{1,1,1}^{d},\tilde{\mathbf{g}}_{2,1,1}^{d},\tilde{\mathbf{g}}_{3,1,1}^{d},\tilde{\mathbf{g}}_{4,1,1}^{d},\ldots,\tilde{\mathbf{g}}_{N,1,1}^{d},\tilde{\mathbf{g}}_{1,2,1}^{d},\tilde{\mathbf{g}}_{2,2,1}^{d},\tilde{\mathbf{g}}_{3,2,1}^{d},\ldots\right]^{T}$$

$$\tilde{\mathbf{f}} = \sum_{d=1}^{3}2\tilde{\mathbf{G}}^{d}\,\mathbf{f}^{d},\ \tilde{\mathbf{H}} = \sum_{d=1}^{3}2\tilde{\mathbf{G}}^{d}\tilde{\mathbf{G}}^{d^{T}}$$

of observed displacement and load values at each node in direction d. Again, the values 1, 2, and 3, for d, represent values in the x-, y-, and z-directions respectively.

3.4 Numerical Example

In this section, we use a set of simulated measured nodal displacement values to estimate the elasticity of a cube of homogeneous tissue with stationary corners. These measurements, like any other measured quantity, include noise components. In this example, we assume white noise. . We use MATLAB to generate a set of 1000 measurements at each node for a system of 1000 nodes when a force of 1N is applied perpendicularly to the center node on the y-z surface of the tissue. We solve for the required elasticity using these measurement values at each node. We repeat the process using different variances for the noise component in the nodal displacement measurements. As seen in Table 3-1, the accuracy of the solution increases with a decrease in the variance. This is not unexpected, as better more accurate measurements will yield a more accurate estimate for $\tilde{k}$.

	Variance			
Actual $\tilde{k}$	***0.0001***	***0.001***	***0.01***	***0.1***
1	1.0001	1.0004	1.0007	0.9970

Table 3-1: Tissue Elasticity Estimates

4 HARDWARE IMPLEMENTATION

In this Chapter, we discuss the hardware that executes and accelerates the CGM. The acceleration of the CGM involves designing hardware optimized for carrying out operations needed by the CGM and the speeding up of MVMs. We discuss both of these here, starting with the speeding up MVMs. We later introduce the CGM accelerator.

4.1 MVM Speed-Up

As discussed in section 2.3, speeding up the MVM will effectively speed up the CGM. The approach used to achieve the increase in speed of completing a MVM was to divide the multiplying matrix and vector into smaller appropriately dimensioned sub-matrices and sub-vector. Each of these sub-matrices and sub-vectors are used by a series of MVUs working in parallel to carry out the required MVM. Given a stiffness matrix **K** for a homogeneous system, generated using the models discussed in previous Chapters, the conjugate symmetry of the matrix can be used to divide it in two smaller sub-matrices **B** and **B***. As **B** contains all the information needed to generate **K**, the memory required to store **K** is reduced by 50% by storing only **B**.

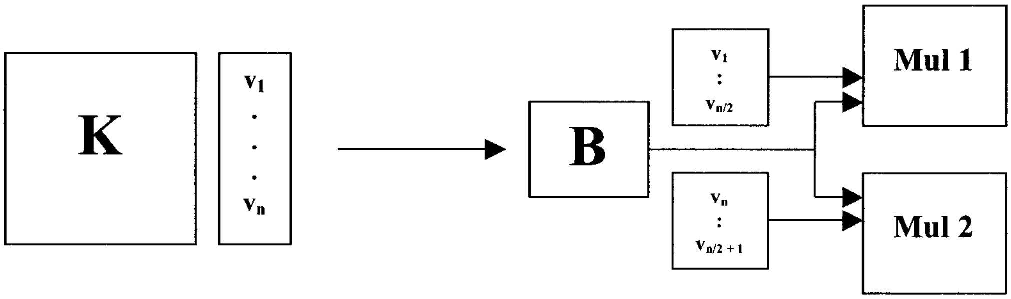


Figure 4-1: MVM using Conjugate Symmetry of K

Only one memory fetch will be needed to access two element of **K** at the same time. This in turn means that two MVUs can be used in parallel, as shown in Figure 4-1, to complete an MVM of **K** and **v** by using sub-matrix **B** and vector **v**. One thing of note is that the ordering of the elements in the vector used by Mul2 is inverted. The ordering of the elements of Mul2's results is inverted to get the bottom half of the final result. There is also an overlap between the vectors assigned to the multipliers. Additional speed up can be achieved by further dividing matrix **B** and the two corresponding vectors in Figure 4-1 into smaller sub-matrices and sub-vector. The two approaches we looked at for carrying out MVMs are discussed below.

4.1.1 MVM using Overlapping Sub-Matrices

The matrix is divided into smaller matrices that overlap each other as shown in Figure 4-2. Figure 4-2 shows a 6x6 matrix being divided into three 4x4 matrices. The nonzero elements of the original larger matrix are distributed as evenly as possible among these smaller sub-matrices. This distribution is, however, done such that,

i. Any shared nonzero elements among the sub-matrices appears in one, and only one, of the possible sub-matrices that it can possibly belong to.

ii. The position of this element in the original matrix is not compromised (i.e. must not change).

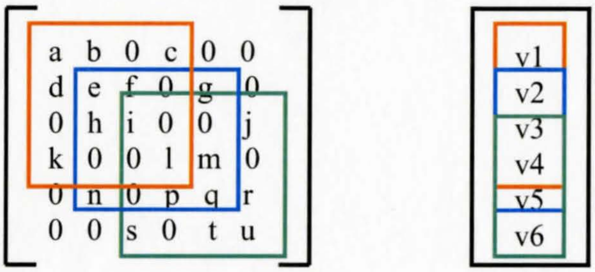


Figure 4-2: Matrix Division using Overlapping Sub-Matrices

Matrix A and its corresponding vector are given by the elements within the orange lines
Matrix B and its corresponding vector are given by the elements within the blue lines
Matrix C and its corresponding vector are given by the elements within the green lines

A MVM is carried, using these sub-matrices, as follows,

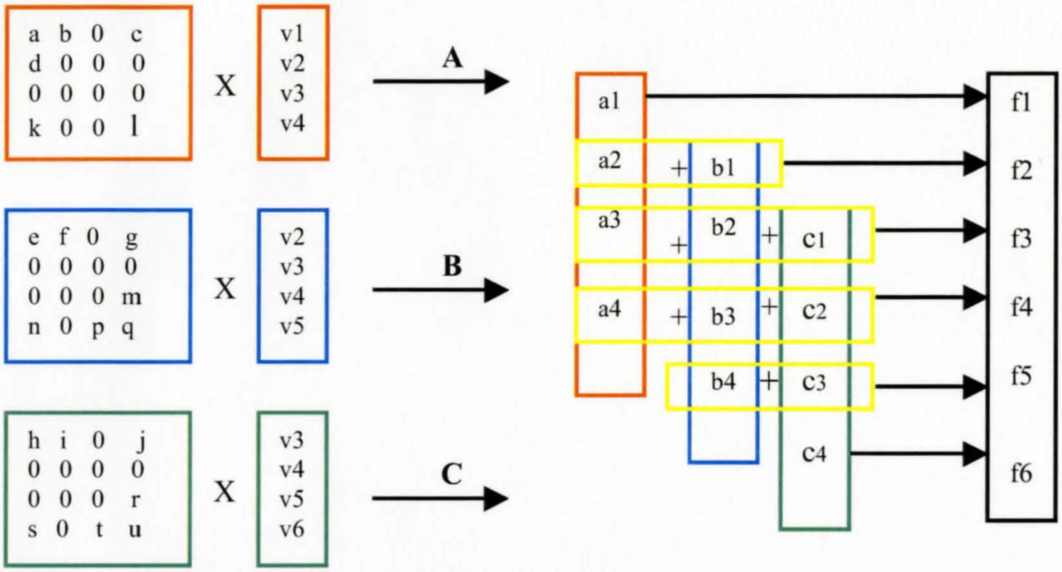


Figure 4-3: MVM using Overlapping Sub-Matrices

The elements of each of the smaller MVM results are matched to the corresponding element in the larger MVM result. In the case of an overlap, the sum of the corresponding overlapping elements (elements in the yellow boxes) is used.

4.1.2 MVM using Non-Overlapping Sub-Matrices

A better and more efficient way is to divide **B** into smaller matrices as shown in Figure 4-4. The nonzero elements of the original larger matrix are also distributed as evenly as possible among these smaller sub-matrices and vector **v** into smaller overlapping vectors as shown below. This is a special case of the approach discussed in section 4.1.1, where the nonzero elements of the original larger matrix are distributed among the smaller sub-matrices, with no overlap between any two sub-matrices. Due to the elimination of

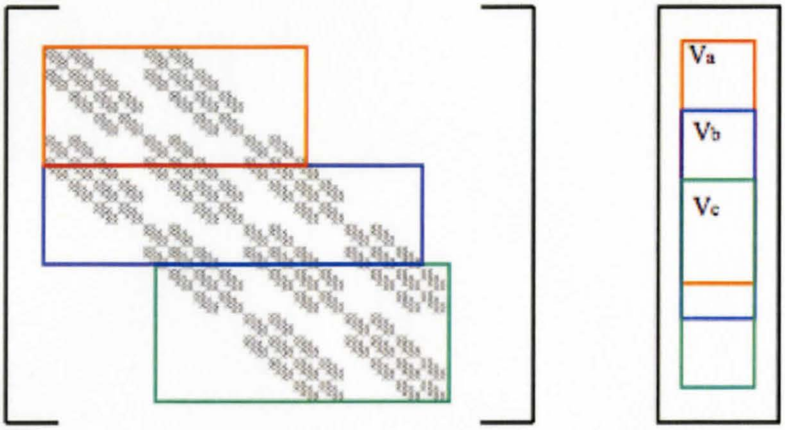


Figure 4-4: Matrix Division using Non-Overlapping Sub-Matrices

these overlaps, the elements of each of the smaller MVM results are matched to the corresponding element in the larger MVM result as shown in Figure 4-5.

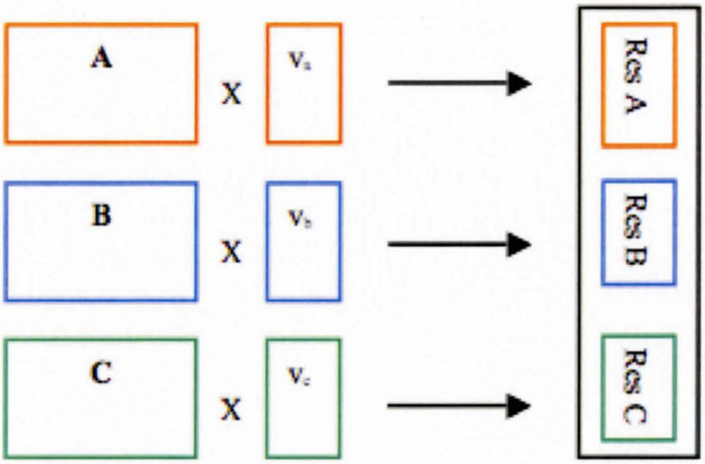


Figure 4-5: MVM using Non-Overlapping Sub-Matrices

In both approaches to MVM, the band of these sub-matrices is the same as that of **B**. Further, the completion time for a MVM will be 3 times faster for **B** and 6 times faster for **K**, since **B** was divided into 3 smaller sub-matrices in the examples above. In the general case, if **B** is divided into *M* smaller sub-matrices the MVM involving **K** will be completed 2*M* times faster. One point of note is that only one memory fetch can be done at any time. Earlier, we described the case when one memory fetch was used to feed two MVUs at the same time. The use of one memory fetch to feed two MVUs is mainly possible because of the conjugate symmetry of matrix **K**. This is, however, not the case for matrix **B**, thus each of the sub-matrices generated using the matrix division techniques discussed earlier must be stored in separate memory blocks, one for each of the MVUs that will be working in parallel. In systems where conjugate symmetry is not applicable, the matrix division techniques must be applied directly to **K**.

4.2 CGM Accelerator

The CGM accelerator consists of a series of MVU for carrying out MVMs and a Scalar-Vector Unit (SVU) for carrying out the remaining VSOs in the CGM.

4.2.1 Matrix-by-Vector Multiplier Unit

This MVU is designed for MVMs, of the form **Kp** and $\mathbf{p}^T\mathbf{Kp}$, which may involve sparse matrices. The design, shown in Figure 4-6, requires only the nonzero elements of the matrix to be stored in the memory. The nonzero elements are stored in memory as part of a simple 32-bit instruction format, shown below, that was designed for the MVU. Further, these nonzero elements are stored in memory using a fixed-point format.

***a*(1bit)**	***b*(1bit)**	***c*(9bits)**	***d*(21bits)**

a bit 31 determines end of matrix.
b bit 30 determines end of current row.
c bits 29 to 21 determine the column of the nonzero value.
d last 21 bits give the nonzero value.

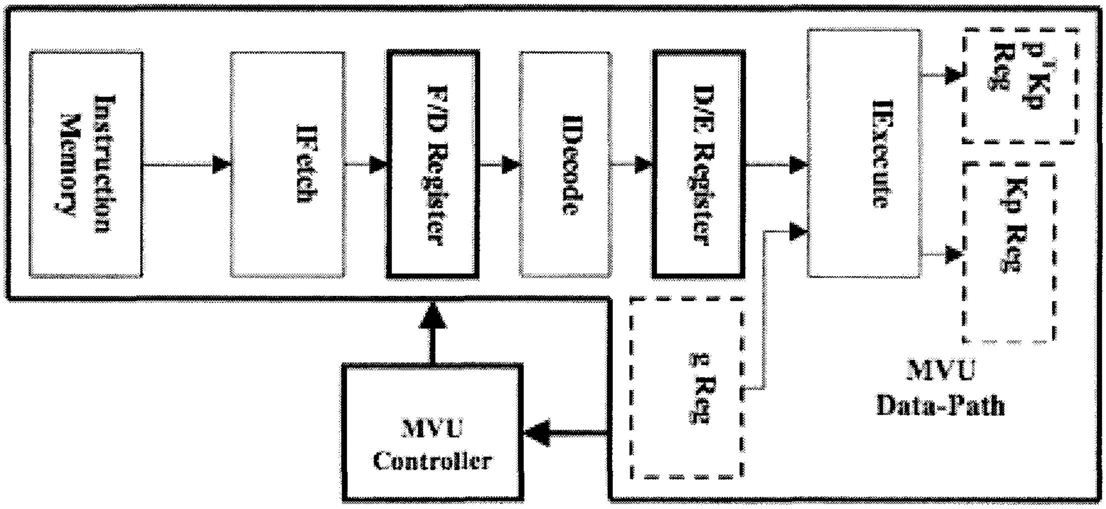


Figure 4-6: MVU Implementation

The MVU datapath is pipelined and divided into three modules

1. Instuction Fetch (module IFetch)
2. Instruction Decode (module IDecode)
3. Execute (module IExecute)

Module IFetch: This module fetches the next instruction from memory and forwards it to module IDecode through register **F/D Reg**. The instructions are read sequentially with addresses gotten from a sequential counter.

Module IDecode: The instruction is decoded here using the format above. It is determined here if the end of the current row or (ER) or the end of matrix (EM) has been reached. These are determined by the first two bits. It also determines the matrix element and the address of the vector element needed for the next multiplication.

Module IExecute: This module performs the traditional MVM (i.e. taking the inner product of each row and the multiplying vector, starting with the first row) using a set of multipliers and an accumulator. The accumulator is reset to zero at the end of every row. The appropriate value in the accumulator is stored in the required location of the Final

Result register. The ER and EM bits are also forwarded to the appropriate registers. One thing of note is that the first element of the result of **Kp**, is available after the first inner product is completed, and the second is available after the second inner product is completed and so on. The availability of these values means that the calculation of $\mathbf{p}^T\mathbf{Kp}$ can be done concurrently with the calculation of **Kp**.

The MVU-Controller controls the flow of information among the registers and modules in the MVU data-path. There are three registers used to pass information between the MVU and the SVU, one for the vector used in the MVM (multiplying vector), while the others are for the MVU results. The multiplying vector register **g Reg** is used for receiving the direction vector from the SVU, while the result registers, $\mathbf{p}^T\mathbf{Kp}$ **Reg** and **Kp Reg,** are used for passing the MVU results ($\mathbf{p}^T\mathbf{Kp}$ and **Kp**) to the SVU. All three registers are updated once every iteration of the CGM. When the MVU has finished a MVM, it notifies the SVU. The SVU then becomes active and begins its operations. The time to complete an MVM depends solely on the number of nonzero elements of the matrix used in the operation. In other words, the time it takes the MVU to complete an MVM varies with matrix size and sparcity.

4.2.2 Scalar-Vector Unit

As mentioned earlier, the SVU carries out all the required VSOs in the CGM except for the MVM. Figure 4-7 shows the set-up that carries out these operations. The design provides some ease and flexibility in carrying out the initialization of the required parameter used in the CGM. Most of the operations in the CGMs main loop, shown below, are dependent on each other; hence they must be carried out sequentially in the order of dependence. For example α must be updated before **x** or **r** is updated, and **r** must be updated before β is updated.

```
for $n = 1:N$
$\alpha = \mathbf{r}_{n-1}^T \mathbf{r}_{n-1} / \mathbf{p}_{n-1}^T K \mathbf{p}_{n-1}$    Calculate New Step Size for x and r
$\mathbf{x}_n = \mathbf{x}_{n-1} + \alpha * \mathbf{p}_{n-1}$    Calculate New x
$\mathbf{r}_n = \mathbf{r}_{n-1} - \alpha * K\mathbf{p}_{n-1}$    Calculate New Residual r
$\beta = \mathbf{r}_n^T \mathbf{r}_n / \mathbf{r}_{n-1}^T \mathbf{r}_{n-1}$    Calculate New Step Size for p
$\mathbf{p}_n = \mathbf{r}_n + \beta * \mathbf{p}_{n-1}$    Calculate New Search direction p
end
```

On the other hand, the updating of **x** and **r** are independent of one another, so they can be updated simultaneously. However, updating **x** and **r** simultaneously is not necessary because the time that is saved, 1 clock cycle, is not justified when considering that the amount of resources that is required to update **x** and **r** will be doubled. Note that the use of a *for* loop keeps the completion time for the SVU fixed.

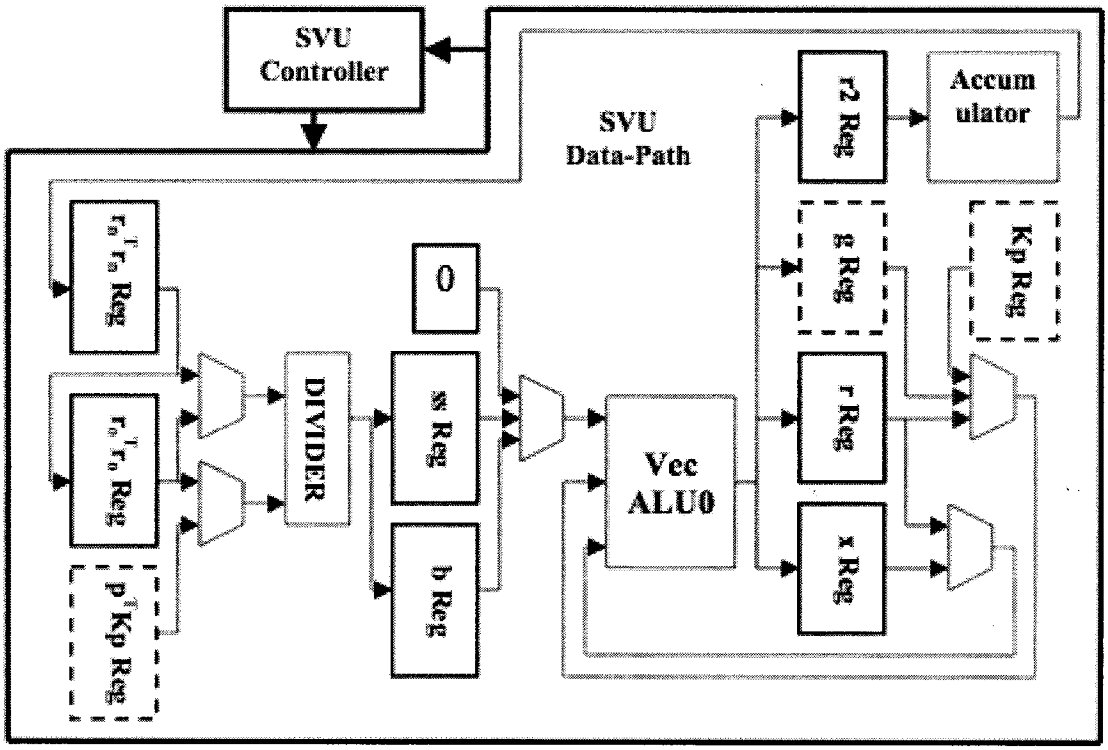


Figure 4-7: SVU Implementation

As seen in Figure 4-7, the three main modules used in the SVU data-path are a Divider, a Vector ALU (VecALU), and an Accumulator. The operations performed by these modules are described below.

Divider: This module is used to calculate α and β, which in turn are used by VecALU.

VecALU: This is an arithmetic logic unit (ALU) that specifically carries out vector-vector or vector-scalar operations. The new values of the residual **r**, direction **g** and variable **x** are calculated here. The module uses these values along with α and β to calculate the required values in the next iteration. The new value of **r**2 **Reg**, also calculated here, is passed to the Accumulator.

Accumulator: This module sums the elements of the register **r**2 **Reg**. The result of this summation is the 2-norm of vector **r**. Hence, each element of register **r**2 **Reg** is the square of the corresponding element of register **r** **Reg**. The Divider uses this 2-norm value in the calculation of α and β.

Like the MVU-Control, SVU-Control also controls the flow of information among the registers and modules in the SVU data-path. As discussed earlier, in section 4.2.1, the MVU result registers, and multiplying vector register are used for passing information between the SVU and MVU. Furthermore, when the SVU has finished its operations, it notifies each of the MVU to start the next MVM.

Our CGM accelerator has just one SVU and several MVUs, hence each of these MVUs must communicate with the SVU and vice-versa. This CGM accelerator is a simple example of how two or more processors can be used in the speed up of numerical algorithms, as the SVU and MVUs are essentially separate application specific processors. Figure 4-8 shows a CGM accelerator set-up when 5 MVUs work in parallel.

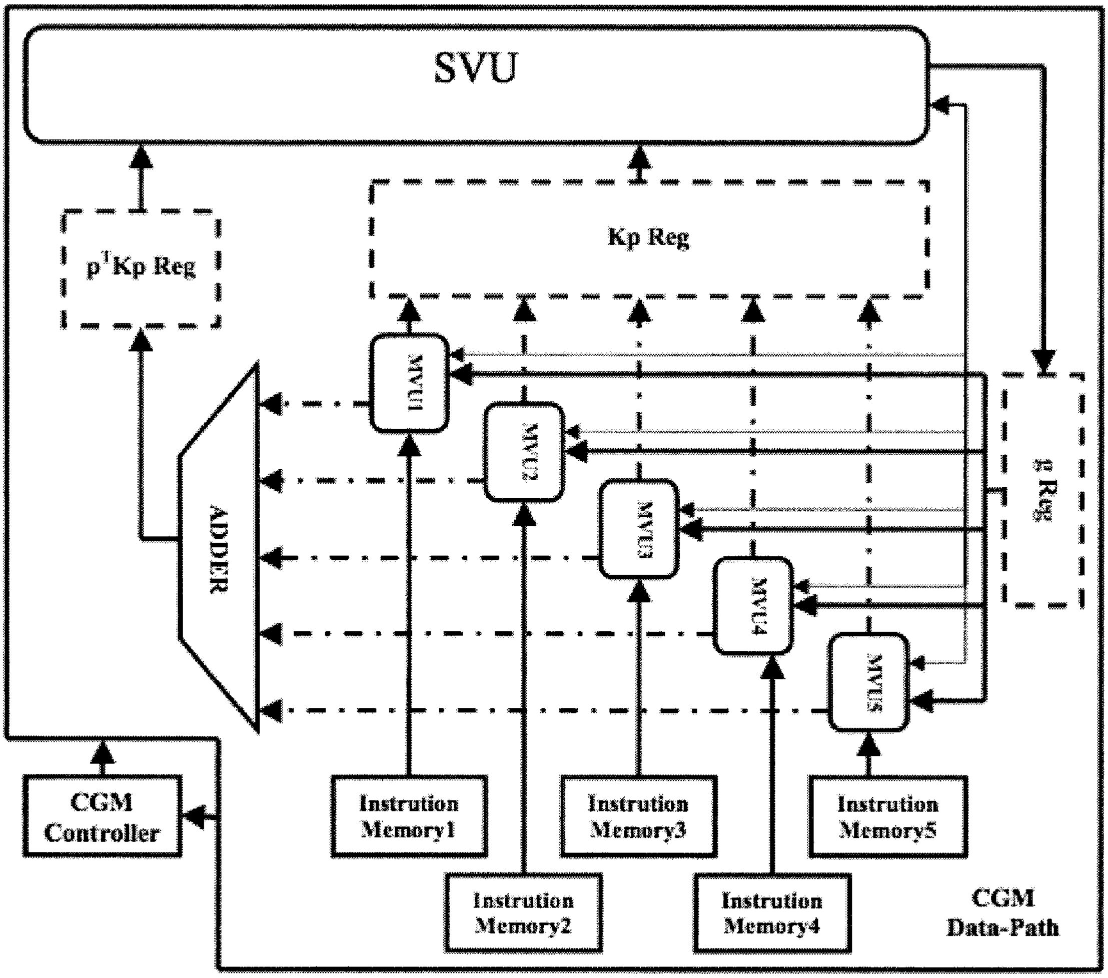


Figure 4-8: CGM Accelerator Set-Up using 5 MVUs

The CGM Controller determines when to start and stop the execution of the CGM. The CGM Controller also controls the number of iterations used in the CGM.

5 RESOURCE USAGE AND PERFORMANCE ANALYSIS

5.1 Resource Usage

FPGAs contains three main resources namely, multipliers, logic elements and registers. Of these three, the multipliers are of least abundance. This makes them the bottle-neck of any design for applications that are heavily dependent on the usage of multipliers. For this reason, the multiplier usage is the primary measure of our designs resource usage quantifier because it is the deciding factor in the maximum size of the system that can be solved on one FPGA. Figure 5-1 shows the multiplier usage of our CGM accelerators implementation for different number of MVUs and grid sizes n (number of nodes). We implemented the CGM accelerator on Altera's DE2 development board using the Quartus II software. The implementation can be clocked at speeds up to 133MHz.

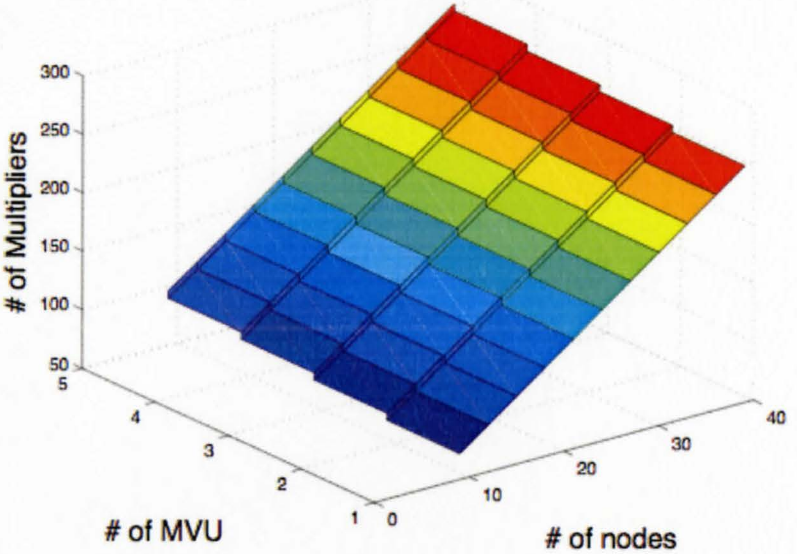


Figure 5-1: Multiplier Usage

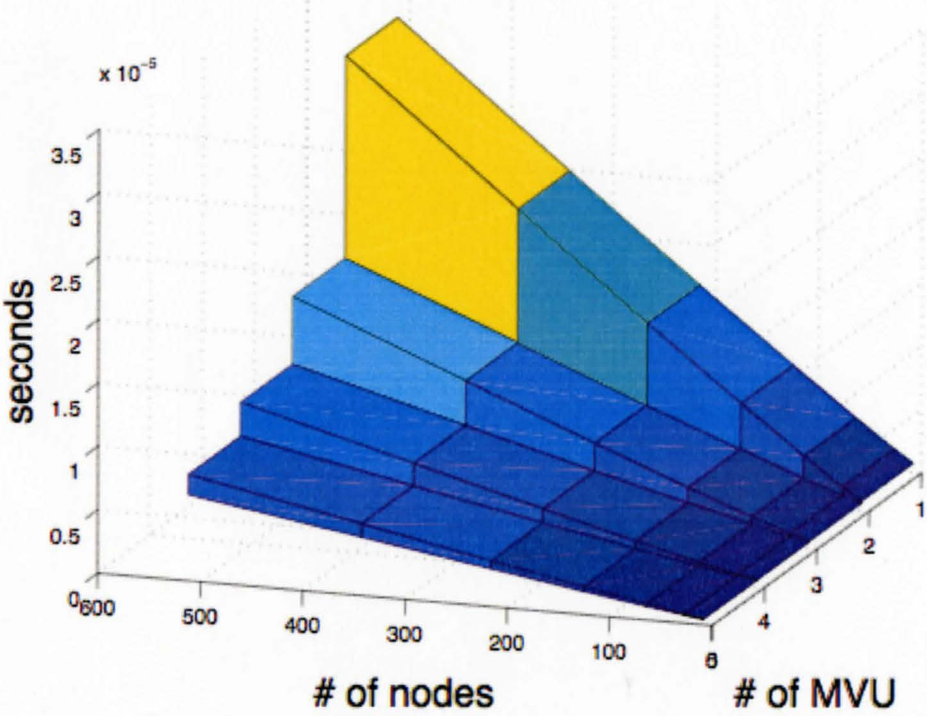
5.2 Performance Analysis

We used a two-pronged approach to carry out the performance analysis on the CGM accelerator. Firstly, we use Altera's simulation software to carry out a timing analysis on the CGM accelerator so as to determine it's timing performance. Secondly, we determine the MFLOPs performance of our CGM accelerator implementation. We repeated the both analyses for different grid sizes. These tests are done at 100MHz.

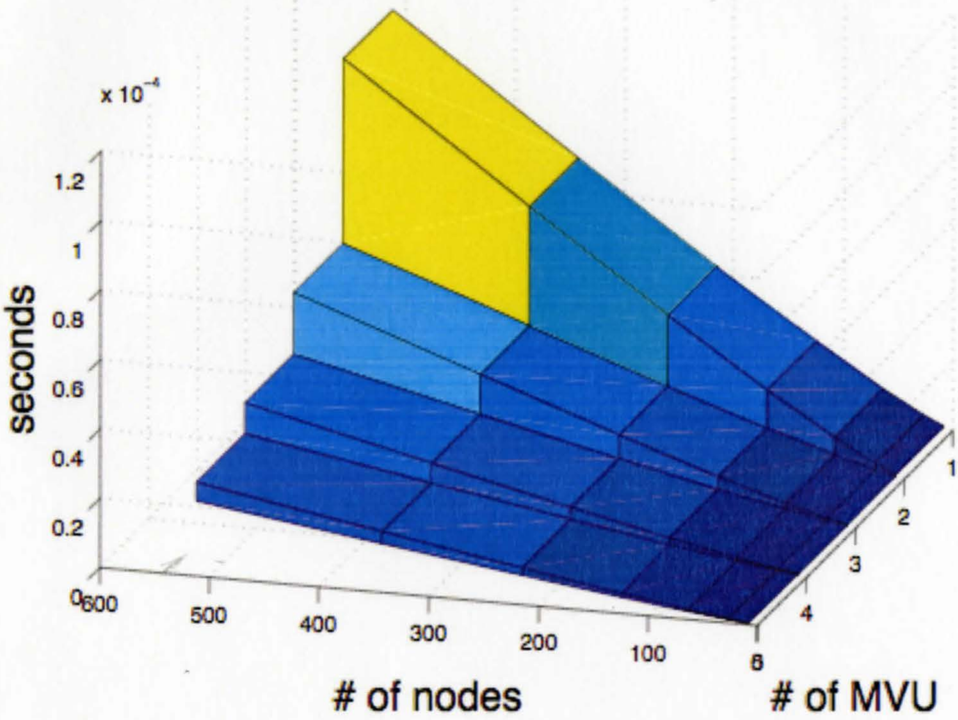
5.2.1 Timing Performance

The completion time, T, for one iteration of the CGM, given by (55), is the sum of the completion times for the SVU and MVU. We use T as a measure of timing performance for our CGM accelerator. Since the implementation of the SVU keeps T_{SVU} constant, this

$$T = T_{MVU} + T_{SVU} \tag{55}$$



Model A



Model B

Figure 5-2: Computation Time for connection Models A and B

makes T_{MVU} the only time that can be changed. T_{MVU} can be improved by using the techniques described in 4.1. Further, minimizing T_{MVU} effectively reduces the to time to carry out the CGM. In Figure 5-2 we show the computation time for one iteration of the CGM as a function of number of MVUs and grid size, for connection models A and B in Figure 2-5.

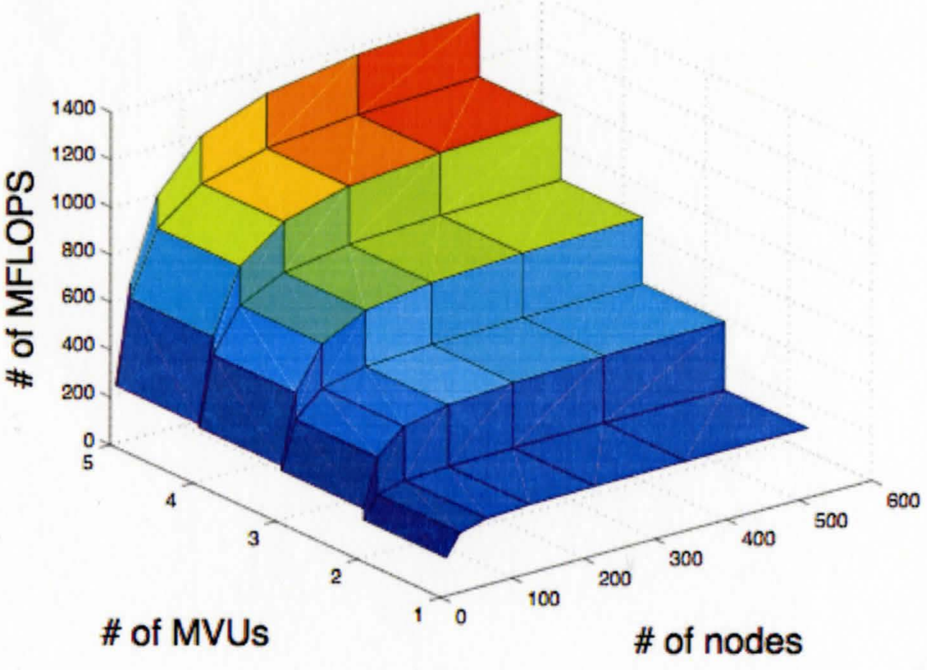
As seen in Figure 5-2, the computation time of systems generated from connection model A is less than that of systems generated from connection model B. This is not unexpected as the stiffness matrix generated using model A is sparser than the stiffness matrix generated using model B. Based on the values in Figure 5-2, in the worst case scenario, a solution to a system with a grid of 500 nodes can be arrived at in 16.6ms and 56.1ms for connection models A and B respectively when one MVU is used. Increasing the number of MVUs working in parallel will reduce computation time. However, the use of one or more MVUs in parallel means that fewer multipliers are available for use by the SVU, as the number of multipliers is fixed.

5.2.2 MFLOPS Performance

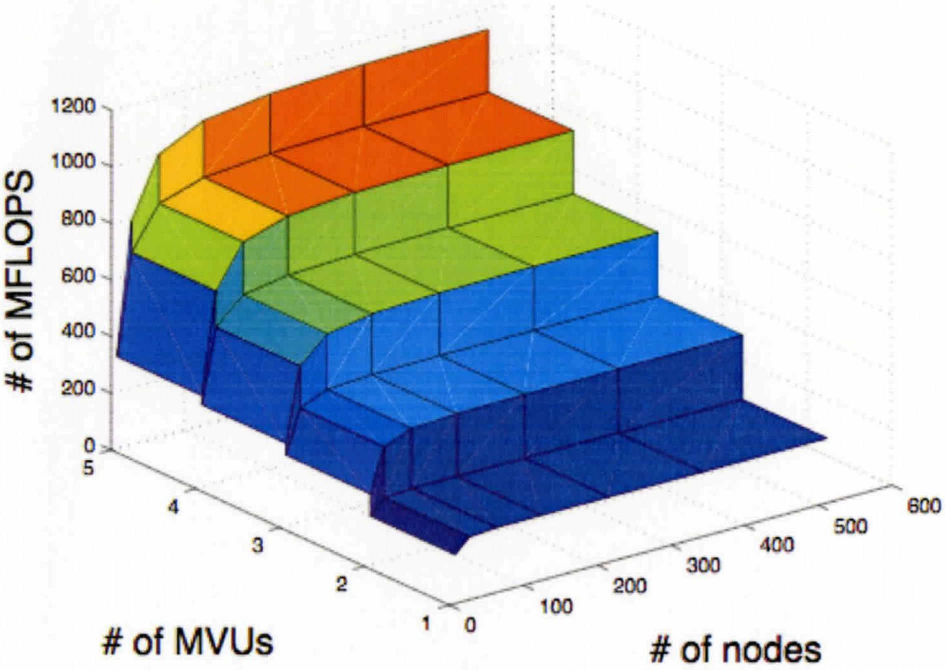
MFLOPS performance, given by (56), is a measure of the number of fixed-point operations (flops) per second.

$$\begin{aligned} MFLOPS &= \frac{Total\# \, of \, flops/iter.}{compute \; time/iter.} \\ &= \frac{2mn + 6n + 2}{T} \end{aligned} \tag{56}$$

Where n is the size of the system and m is the average number of nonzero elements per row. In Figure 5-3 we show the MFLOPS performance of the CGM as a function of number of MVUs and grid size, for connection models A and B in Figure 2-5.



Model A



Model B

Figure 5-3: MFLOPS Performance for connection Models A and B

Our CGM accelerator is able to achieve more than 1400 MFLOPS and 1100 MFLOPS for systems based on connection models A and B respectively. Both values are achieved with 5 MVUs working in parallel.

As you can see in Figure 5-3, the performance of the system plateaus as n gets very large. Figure 5-3 also shows a performance increase with an increase in the number of MVUs used in parallel. Hence, to further improve performance, we can use more MVUs in parallel. Further more, the amount of resources available determines the number of MVUs that can be used in parallel. Also note that while the computation times of systems generated from connection model A are less than those of systems generated from connection model B, the MFLOPS performance are in reverse order. In other words, the MFLOPS performance of systems generated from connection model A are higher than those of systems generated from connection model B. This can also be attributed to the same reason in 5.2.1, that is, the stiffness matrix generated using model A is sparser than the stiffness matrix generated using connection model B.

6 CONCLUSION

We proposed and implemented an FPGA based CGM accelerator for carrying out real-time simulation of tissue deformation. Our design takes the sparsity of the stiffness matrix into account. Further more, we discussed methods of improving the speed of MVMs using parallel computing. We then looked at the resource requirements and the performance of the CGM accelerator. Our work shows that developing FPGA based accelerators for use in real-time simulation is feasible. Furthermore, our results provide a benchmark for more complex, specific, and realistic models.

The convergence analysis, discussed in Section 2.6, showed that computational models generated using connection model B converges faster, while the timing analysis, discussed in Section 5.2.1 shows that computational models generated using connection model A have faster computation times. Further study may be carried out on the trade off between the convergence and timing properties of a computational model in relation to the connection model used in its generation. For example convergence and timing analysis may be done on computational models generated using of a combination of connection models A and B. Future work may also include using more specific and realistic computational models, as our CGM accelerator allows for the use of matrices with different sizes and sparsity. For example, a more specific and realistic computational model for needle insertion into a soft tissue may account for bending and force distribution along the needle shaft. These computational models may also be generated using a more accurate discretization technique like the finite-element method.

REFERENCES

[Chapra] Chapra, C. S., Canale, P. R., 2002. *Numerical Methods for Engineers*, McGraw Hill. NewYork, 4th edition. 820-871.

[Delingette] Delingette, H., 1998. *Towards Realistic Soft Tissue Modeling in Medical Simulation.* IEEE Special Issue on Surgery Simulation. 512-523.

[Devore] Devore, J. L., 2004. *Probabilty and Statistics for Engineering and the Sciences*, Brooks/Cole, Toronto, 6th edition. 272-277.

[DiMaio1] DiMaio, S. P., Salcudean , S. E., 2002. *Needle Insertion Modelling for the Interactive Simulation of Percutaneous Procedures.* In the 5th International Conference on Medical Image Computing and Computer-Assisted Intervention-Part II. Springer-Verlag 253-260.

[DiMaio2] DiMaio, S. P., Salcudean , S. E., 2003. *Needle Insertion Modelling and Simulation.* IEEE Transaction on Robotics and Automation. Volume 19, Issue 5 864-875.

[DiMaio3] DiMaio, S. P., Salcudean , S. E., 2005. *Needle Steering and Motion Planning in Soft Tissues.* IEEE Transaction on Biomedical Engineering. Volume 52, Issue 6 965-974.

[DiMaio4] DiMaio, S. P., Salcudean , S. E., 2005. *Interactive Simulation of Needle Insertion Models.* IEEE Transaction on Biomedical Engineering. Volume 52, Issue 7 1167-1179.

[Fung1] Fung, Y.C., 1990, *Biomechanics: Motion, Flow, Stress, and Growth*, Springer-Verlag, New York. 29-38.

[Fung2] Fung, Y.C., 1993, *Biomechanics: Mechanical Properties of Living Tissues*, Springer-Verlag, New York, 2nd edition. 23-40

[Goulb] Goulb, H. G., Van Loan, F. C., 1996. *Matrix Computations*, The John Hopkins University Press. London, 3rd edition. 490-493, 520-528.

[He] He, C., Qin, G., Zhao, W. *FPGA-Based High-Order Finite Difference Method for Linear Wave Modelling Problems.* Retrieved June 11, 2006, from http://lacsi.rice.edu/symposium/symposumdownloads.

[Hennessy] Hennessy, L. J., Patterson, A. D., 2003. *Computer Architecture: A Quatitative Approach*, Morgan Kaufmann Publishers. San Francisco, CA, 3rd edition.

[Nash] Nash, M., 1998. *Mechanical and Material Properties of the Heart using an Anatomically Accurate Mathematical Model.* Auckland, New Zealand. Department of Engineering Science, School of Engineering, The University of Auckland. 12-26.

[Ramachandran] Ramachandran, K., 1998. *Unstructured Finite Element Computations on Configurable Computers.* Blacksburg, Virginia: University Libraries, Virginia Polytechnic Institute and State University.

[Rocha] Rocha, K. M. C. *Numerical Techniques for Real Options.* Retrieved May 15, 2006, from http://www.puc-rio.br/marco.ind/katia-num.html.

[Stolle] Stolle, D., 2005. *An Introduction to the Finite Element Method.* Hamilton, Ontario. Department of Civil Engineering, McMaster University. 70-91.

[Zhuo] Zhuo, L., Prasanna, V. K., 2005. *Sparse Matrix-Vector Multiplication on FPGAs.* In Computation algorithms for FPGA, ACM/SIGDA 13th international symposium on Field programmable gate arrays. ACM Press. 63-74.