# PARALLEL IMPLEMENTATION OF THE BOUNDARY ELEMENT METHOD FOR ELECTROMAGNETIC SOURCE IMAGING OF THE HUMAN BRAIN

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

YOLDAŞ ATASEVEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2005

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan ÖZGEN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof.Dr.İsmet ERKMEN
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof.Dr.Nevzat Güneri GENÇER
Supervisor

Examining Committee Members

| | | |
|---|---|---|
| Prof.Dr.Mustafa Kuzuoğlu | (METU EEE DEPT) | |
| Prof.Dr.Nevzat Güneri Gençer | (METU, EEE DEPT) | |
| Prof.Dr.Kemal Leblebicioğlu | (METU, EEE DEPT) | |
| Assist.Prof.Dr.Yeşim Serinağaoğlu | (METU, EEE DEPT) | |
| Dr.Can Erkin Acar | (Pro-G Ltd. Şti.) | |

**I hearby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required, I have fully cited and referenced all material and results that are not original to this work.**

**Name Lastname :** Yoldaş Ataseven

**Signature :**

# ABSTRACT

## PARALLEL IMPLEMENTATION OF THE BOUNDARY ELEMENT METHOD FOR ELECTROMAGNETIC SOURCE IMAGING OF THE HUMAN BRAIN

Ataseven, Yoldaş
B.Sc., Department of Electrical and Electronics Engineering
Supervisor: Prof.Dr.Nevzat Güneri Gençer

September, 2005, 103 pages

Human brain functions are based on the electrochemical activity and interaction of the neurons constituting the brain. Some brain diseases are characterized by abnormalities of this activity. Detection of the location and orientation of this electrical activity is called electro-magnetic source imaging (EMSI) and is of significant importance since it promises to serve as a powerful tool for neuroscience. Boundary Element Method (BEM) is a method applicable for EMSI on realistic head geometries that generates large systems of linear equations with dense matrices. Generation and solution of these matrix equations are time and memory consuming due to the size of the matrices and high computational complexity of direct methods. This study presents a relatively cheap and effective solution the this problem and reduces the processing times to clinically acceptable values using parallel cluster of personal computers on a local area network. For this purpose, a cluster of 8 workstations is used. A parallel BEM solver is implemented that distributes the model efficiently to the processors. The parallel solver for BEM is developed using the PETSc library. The performance of the

solver is evaluated in terms of CPU and memory usage for different number of processors. For a 15011 node mesh, a speed-up efficiency of 97.5% is observed when computing transfer matrices. Individual solutions can be obtained in 520 ms on 8 processors with 94.2% parallellization efficiency. It was observed that workstation clusters is a cost effective tool for solving complex BEM models in clinically acceptable time. Effect of parallelization on inverse problem is also demonstrated by a genetic algorithm and very similar speed-up is obtained.

Keywords: Boundary Element Method, Electromagnetic Source Imaging, Brain, Parallel Processing.

# ÖZ

## İNSAN BEYNİNİN ELEKTROMANYETİK KAYNAK GÖRÜNTÜLEMESİNDE SINIR ELEMANLARI YÖNTEMİNİN PARALEL UYGULAMASI

Ataseven, Yoldaş
Lisans, Elektrik ve Elektronik Mhendisliği Bölümü
Tez Yöneticisi: Prof.Dr.Nevzat Güneri Gençer

Eylül 2005, 103 sayfa

İnsan beyninin faaliyetleri, beyni oluşturan sinir hücrelerinin etkileşimi ve elektro-kimyasal aktivitelerine dayanır. Beynideki bazı bozukluklar ve hastalıklar da bu aktivitelerin bozukluğu ile karakterize edilir. Bu aktivitenin yerini ve yönünü belirlemeye elektromanyetik kaynak görüntüleme (EKG) denir ve sinirbilim için güçlü bir araç olacağından oldukça önemlidir. Sınır Elemanları Yöntemi (SEY), EKG için uygulanabilen, büyük ve yoğun denklem takımları yaratan bir yöntemdir. Bu denklem takımlarının oluşturulması ve çözülmesi, denklem takımlarının büyüklüğü ve doğrudan yolların işlem gücü gereksinimi nedeniyle uzun işlem süreleri ve büyük bellekler gerektirir. Bu çalışma, mevcut probleme ucuz ve verimli bir çözum sunmakta ve işlem sürelerini klinik olarak kabul edilebilir değerlere indirmektedir. Bu amaçla, 8 bilgisayarlı bir bilgisayar kümesi kurulmuş ve kullanılmıştır. Kullanılan modeli işlemcilere verimli bir şekilde dağıtan bir paralel çözücü geliştirilmiştir. Çözücünün başarımı, çeşitli sayıda işlemciler için işlemci ve bellek kullanımı bakımından incelenmiştir. 15011 düğümlü bir ağda, 8 işlemci kullanıldığı zaman transfer matrislerinin hesaplanmasında 97.5% verimlilik

gözlenmşitir. Bir tek çözümü alma süresinde 8 işlemci ile 94.2% hızlanma verimi ile 520ms sürede çözüm alınması saglanmıştır. Bilgisayar kümelerinin karmaşık BEM modellerinde çözüm almak için ucuz ve etkili bir arac oldugu gözlenmiştir. Paralelleştirmenin geri problem üzerindeki etkisi bir genetik algoritma ile gösterilmiş ve ileri problem çözümleri için çok benzer hızlanma sonuçları alınmıştır.

Anahtar sözcükler: Sınır Elemanları Yöntemi, Elektromanyetik Kaynak Görüntüleme, Beyin, Paralel İşleme.

To my parents and sister

# ACKNOWLEDGMENTS

The author wishes to express deepest gratitude to his supervisor, Prof. Dr. Nevzat G. Gençer, for his guidance, and insight throughout the research.

The author also wishes to thank Dr. Zeynep Akalın-Acar for the data, software, explanations and help provided by her; and Dr. Can Acar for his help and comments on parallelization and Unix.

The author also would like to thank Prof. Dr. Kemal Leblebicioğlu, for his suggestions and comments on the inverse problem.

Finally, the author thanks to his colleagues in the biomedical research family for their friendship and old friends Tolga Köktürk, Çağrı Karapıçak, and Ayhan Yılmaz for their existence, which gives him strength.

This study was supported by Grant No: BAP-2003-07-02-00-43 from Middle East Technical University Research Program.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| ATLAS | Automatically Tuned Linear Algebra Software |
| atol | Absolute Tolerance (used in KSMs) |
| BEM | Boundary Element Method |
| BiCG | Bi-conjugate Gradients |
| BiCGSTAB | Bi-conjugate Gradients Stabilized |
| CSF | Cerebrospinal Fluid |
| CGNE | Conjugate Gradients on Normalized Equations |
| CR | Conjugate Residuals |
| CGS | Conjugate Gradients Squared |
| CISC | Complex Instruction Set Computing |
| CPU | Central Processing Unit |
| BLAS | Basic Linear Algebra Subprograms |
| EEG | Electroencephalography |
| EMSI | Electromagnetic Source Imaging |
| FDM | Finite Difference Method |
| FEM | Finite Element Method |
| FP | Forward Problem |
| FSB | Front Side Bus |
| GMRES | Generalized Minimal Residual |
| I/O | Input-Output |
| IP | Inverse Problem |

| | |
|---|---|
| IPA | Isolated Problem Approach |
| KSM | Krylov Subspace Method |
| LAN | Local Area Network |
| LAPACK | Linear Algebra Package |
| MEG | Magnetoencephalography |
| MPI | Message Passing Interface |
| NMR | Nuclear Magnetic Resonance |
| OS | Operating System |
| PC | Personal Computer |
| PETSc | Portable, Extensible Toolkit for Scientific Computation |
| PVM | Parallel Virtual Machine |
| RAM | Random Access Memory |
| RDM | Relative Difference Measure |
| RISC | Reduced Instruction Set Computing |
| rsh | Remote Shell |
| rtol | Relative Tolerance (used in KSMs) |
| SIMD | Single Instruction, Multiple Data |
| SNR | Singal-To-Noise Ratio |
| TFQMR | Transpose-free Quasi-minimal Residual |

# CHAPTER 1

# INTRODUCTION

## 1.1 General Overview

The electrical activities in the brain generate electrical potentials on the scalp surface and magnetic fields near the scalp. Electroencephalography (EEG) and Magnetoencephalography (MEG) use electrodes and SQUID magnetometers to sense these fields. In Electro-magnetic Source Imaging (EMSI), the ultimate goal is to determine the distribution of these activities using the EEG and MEG measurements [1], [2], [3], [4]. The *forward problem* (FP) of EMSI is defined as calculating the electric potentials/magnetic fields on/near the scalp surface, given the electrical activities in the brain [5]; and the *inverse problem* (IP) is defined as finding the electrical activities from these measurements. It is apparent that, accurate results require accurate modeling [6], [7], [8]. Since analytical solutions for the FP are not available for realistic head models, numerical methods are used. Boundary Element Method (BEM) is a numerical method that solves the electric/magnetic field on the boundaries of different conductivity regions [8], [9], [7], [6], [10].

The BEM discretizes the corresponding integral equations on the conductivity boundaries to form a linear system of equations. The system matrix for the potential solutions is a dense matrix with no positive-definiteness or symmetry. When a detailed realistic head model is used, the dimensions of the resulting system matrix increase. Therefore, application of computationally expensive direct matrix solution methods is not feasible for solving the BEM matrix equations. Furthermore, only a limited set of iterative solution techniques can be applied due to the lack of positive definiteness and sym-

metry. If the number of unknowns (nodes) is increased in order to improve the accuracy, computation time and memory requirement increase quadratically. Therefore, computational complexity is a limiting factor in the forward problem solutions using large BEM meshes. In order to avoid long processing times and to prevent running out of memory, even the recent studies to use coarse meshes for realistic models [11], [12], [13]. The purpose of this study is to present a scalable parallelization scheme using a Beowulf cluster in both the system matrix generation and the FP solutions. The advantages of such a system are twofold: speeding up the FP calculations and elimination of memory limitations.

The IP is an optimization problem to be solved for optimum localization of the source, based on successive FP solutions. Thus, the solution time for the FP is the determining factor for IP solution time. For faster FP solutions, Akalın-Acar and Gençer used an accelerated BEM formulation, which calculates only the specific rows of the system matrix inverse that correspond to measurement nodes [12]. This approach reduces the FP solution to a matrix-vector multiplication for a specific source configuration. Even for the accelerated BEM, the computation times required for generation of coefficient and transfer matrices are long. The studies on speeding up EMSI are limited and parallel processing was attempted only for the Finite Element Method (FEM) implementations [14], [15], [16]. Although parallel implementation of the BEM was performed for various engineering problems [17], [18], [19], it has not been applied for EMSI.

Memory is another factor that prevents the use of detailed BEM head models. An increase in the number of nodes in a BEM mesh will increase the system matrix size exponentially. Consequently, the number of nodes is limited by the available memory of the computer used. If a realistic head model with high number of nodes will be used, required memory may exceed the available system resources. Parallelization solves this problem since the available memory can be extended by a factor of the number of personal computers (PCs) in a cluster.

Parallel FP solution can be implemented using direct (Gauss elimination based) or iterative approaches. The direct implementations are generally based on matrix factorization [17]. Iterative methods are divided into two main groups: stationary methods and Krylov subspace methods (KSMs) [20]. It has been shown that, for the BEM, there are efficient robust iterative KSMs which are better than Gauss-based methods in complexity [21] and thus in processing time. They are also superior to stationary methods in convergence rate. For this purpose, basic KSMs sacrifice applicability to all kinds of matrices. Yet, there are modified KSMs that are applicable to all types of matrices and much faster than stationary methods. In this study, the performance of these methods are tested and the most efficient method, namely the Generalized Minimal Residual (GMRES) method is used for the BEM system solutions.

To solve the system of equations, the Portable, Extensible Toolkit for Scientific Computation (PETSc) [22] is employed which allows the use of almost all KSMs in the literature. PETSc is a high performance scientific computations library that uses the Message Passing Interface (MPI) framework [23]. While PETSc is focused on sparse matrix operations, solvers for dense matrices are also available.

As the computational platform, a workstation cluster of 8 computers is used. Such a cluster of workstations, communicating over a local area network (LAN) is called a Beowulf cluster. Beowulf clusters are cheap and easy to build. They are constructed using readily available off-the-shelf PC hardware and are flexible for future modifications. [24].

This work introduces parallel processing for the forward and inverse problems of electromagnetic source imaging with BEM and reports the improvement in the performance with parallelization for 3-layer spherical and realistic head models (meshes). A parallel PC cluster of 8 computers is constructed over Gigabit Ethernet and various scientific computation and parallel processing libraries are installed to construct a parallel processing environment. The code is developed for forward and inverse problems using

a high-level library (PETSc) and C++ programming language. A simple genetic algorithm with real valued chromosomes is used for inverse problem solutions and tested on spherical and realistic head models.

## 1.2 Literature Summary

There are a number of approaches to solve the forward problem of BEM. The simplest one is to model the head as a sphere [25], [26],[27] or as concentric spherical shells [28]. Improvements to this approach also exist [7]. However, it is apparent that accurate results necessitate accurate (i.e. realistic) modeling [6], [8]. Such models are patient dependent and disable generic analytical solutions. Therefore, the studies that use realistic head models are generally based on more complex numerical methods [12], [10]. Spherical models are used for testing proposed methods since analytical solutions are available for such models [26].

The geometry of the head can be approximated by a patient-independent model [29], [30] or extracted from the output of an imaging device such as MRI or CT [12], [31], [32], [33]. The resultant numerical model is a mesh composed of nodes and their connection information (elements). Different element types [12], [11] are possible to describe the mesh. For acceptable accuracy in the results, the mesh is required to have sufficient number of nodes [12], [34]. In the BEM, the EMSI problem is reduced to a linear system of equations defined by an integral equation [25]. The corresponding system matrices are general dense matrices. Consequently, the studies mentioned thus far -including the recent ones- generally use moderate number of nodes (less than 10000) in their BEM models.

There are two main groups of methods for the solution of the BEM equations: direct (Gaussian elimination based) methods and iterative methods. Direct methods are generally based on matrix factorization [17]. Iterative methods are; old-fashion stationary methods (Jacobi, SOR, etc) and well-known Krylov Subspace Methods (KSMs) [20].

In the literature, KSMs are used for EMSI with BEM [35], [36]. It is known that, for BEM, there are efficient robust iterative methods, which are better than the Gauss-based methods in complexity, and so the processing time [36], [21]. KSMs are also superior to stationary methods in convergence rate [20]. However, basic KSMs sacrifice applicability to all kinds of matrices. Yet, there are modified KSMs (such as GMRES, BiCGSTAB, CGS, CR, etc.) applicable to general matrices and much faster than stationary methods. Even with KSMs, solutions are time consuming due to large system matrices and the need for many matrix-vector products.

Akalın and Gençer [12] proposed an accelerated BEM, which calculates only the specific rows of the system matrix inverse (with KSMs) that correspond to the measurement points (EEG/MEG). Such an approach will reduce the forward problem solution to matrix-vector multiplications, reducing the inverse problem solution time significantly. This method is based on constructing the selected rows of the system matrix inverse by solving the forward problem for a fixed number (number of electrodes) of times.

Even for accelerated BEM, matrix fill, transfer matrix calculation and forward problem solution time on a fast computer for a moderate number of nodes is large [12]. If the number of nodes is increased due to accuracy considerations, computation times increase exponentially. Therefore, computational complexity is a limiting factor for BEM mesh sizes.

Parallelization of the BEM is a way to reduce computation time, which is quite long due to computational complexity of the methods and size of the problems. Parallel solution can implement a direct (Gauss-based) or indirect (iterative) approach. Parallelization of the BEM has examples in various engineering problems [19], [17], [18]. However, it is not used for the EMSI of the human brain except for generating head models [37] and some FEM applications [15].

There are various widely used open source high performance scientific computation libraries that can be accessed through The Internet. These libraries range from low level (inter-processor communication) [23], [38], to

high level (distributed data types, parallel solvers, etc.) [22], [39], [40] and built on top of each other (a higher level library uses the routines of the lower level library routines to perform lower level tasks). Although most of them are developed using FORTRAN, C interfaces are also provided for high-level libraries (with little loss in performance).

When FP solution is available, the next step is the solution of the IP. IP of EMSI with BEM cannot be solved by deterministic optimization methods since many local minimums are possible due to limited number of detectors. There are various studies on the IP which are focused on different topics of IP [41], [27], [10]. However, they ignore the need for a global optimization method except [42]. Genetic algorithm is used in [42] for EMSI with BEM and reported to provide excellent localization accuracy.

Evolutionary (genetic) algorithms are widely used in many engineering problems that necessitate obtaining the global minimum in the existence of many possible local minimums [43], which is the case for IP of EMSI with BEM [42]. For that, it is classified to be a global optimization method.

## 1.3 Objective of the Study

This study aims at:

1. Constructing a high performance scientific computation platform using a Beowulf class computer cluster with personal computers.

2. Implementing the accelerated BEM approach in the Beowulf cluster to speed up the forward problem of the EMSI.

3. Implementing a real coded Genetic Algorithm in the Beowulf cluster to speed up the inverse problem of the EMSI.

## 1.4 Significance of the Study

This work contributes to two major aspects of the EMSI: forward and inverse problems. A parallelization scheme is proposed to solve the FP of EMSI using the accelerated BEM. The proposed scheme is scalable and promises to be faster for increased number of processors. It is effective in matrix storage and filling, calculation of the inverse matrix's selected rows and individual solutions (matrix-vector products). Consequently, it provides faster solutions for the forward problem of the BEM. A global optimization algorithm, namely the Genetic algorithm is adopted to solve inverse problem in the parallel platform. The parallel forward problem solver is directly used for fitness function calculations. Thus, the speed up in the forward problem calculations is directly reflected to the inverse problem calculations. The proposed parallelization scheme allows the usage of very accurate head models in clinical applications.

## 1.5 Outline of the Thesis

In this thesis study, a general overview of the forward problem with the accelerated BEM approach will be presented in Chapter 2. The proposed parallelization scheme is described in Chapter 3. In Chapter 4, Genetic Algorithm adopted for the inverse problem is introduced. The results of the parallelization for both forward and inverse problem solutions are reported in Chapter 5.

# CHAPTER 2

# FORWARD PROBLEMS OF EMSI

Brain functions are generated by regional collective activity of neurons. Although the activity of a single neuron is not likely to be detected outside the head, the neural activity within a small region can be distinguished in EEG/MEG recordings due to the current density created by neurons in that region. The current density $\vec{J}_p$ within a differential volume element $dv$ can be modeled as a current dipole $\vec{p}$ ($\vec{p} = \vec{J}^p dv$). The activity within larger regions can also be modeled as current dipoles [5].

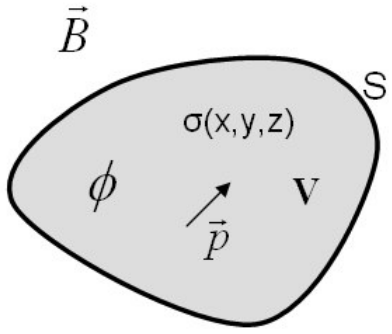


Figure 2.1: A current dipole p in an inhomogeneous volume conductor V bounded by surface S, $\sigma$(x,y,z) is the conductivity distribution function of the volume conductor, $\phi$ is the potential distribution in V and B is the magnetic field near S.

If the region of interest has a layered structure with different average conductivity values for each region, BEM is applicable for calculating the potential distribution $\phi$ on S.

## 2.1 Formulation

The BEM is a frequently employed numerical method for the solution of the forward problem of EMSI. It is based on finding the electric potentials on the tissue boundaries of head using the boundary conditions for the electric field. The sources inside the brain that generate these potentials are assumed to be dipolar [5]. The electric potential $\phi$ and the magnetic field $\vec{B}$ due to a dipole source $\vec{p}$ in a piecewise homogeneous volume conductor can be found using the following integral equations [44]:

$$\bar{\sigma}\phi(\vec{r}) = g(\vec{r}) + \frac{1}{4\pi}\sum_{k=1}^{L}\left(\frac{\sigma_k^- - \sigma_k^+}{\sigma_i^- + \sigma_i^+}\right)\int_{S_k}\phi(\vec{r}\,')\frac{\vec{R}}{R^3}\cdot d\vec{S}_k(\vec{r}\,'), \tag{2.1}$$

$$\vec{B}(\vec{r}) = \vec{B}_0(\vec{r}) + \frac{\mu_0}{4\pi}\sum_{k=1}^{L}\left(\sigma_k^- - \sigma_k^+\right)\int_{S_k}\phi(\vec{r}\,')\frac{\vec{R}}{R^3}\times d\vec{S}_k(\vec{r}\,'). \tag{2.2}$$

Here, $\vec{R} = \vec{r} - \vec{r}\,'$ is the displacement vector between the field point $\vec{r}$ and the source point $\vec{r}\,'$ and $R$ is the magnitude of $\vec{R}$. $S_k$ are the boundaries that lie between two different conductivity regions having conductivity values $\sigma_k^-$ for the inner layer and $\sigma_k^+$ for the outer layer. $\bar{\sigma}$ denotes the average conductivity of inner and outer layers of the field point. The potential at any point $\vec{r}$ on a boundary has two major parts: the field generated by the dipole (*primary source*) in an unbounded medium ($g$ and $\vec{B}_0$), and the field generated by surface currents (*secondary sources*) at the boundaries of layers having different conductivities. The terms $g$ and $\vec{B}_0$ are given as:

$$g(\vec{r}) = \frac{1}{4\pi\sigma_0}\frac{\vec{p}\cdot\vec{R}}{R^3}, \tag{2.3}$$

$$\vec{B_0}(\vec{r}) = \frac{\mu_0}{4\pi}\frac{\vec{p}\times\vec{R}}{R^3}, \tag{2.4}$$

where $\mu_0$ is the permeability of vacuum and $\sigma_0$ is unity.

Since the magnetic field can be obtained directly form the potential distribution, in the remaining part of this text, only potential distribution will be mentioned to avoid duplication. Magnetic field considerations will be mentioned wherever needed.

The above formulation is computationally suitable, since the surface integrals can be numerically approximated by discretizing the surfaces with elements and interpolation over each element.

## 2.2 Methodology

To compute the secondary terms in (2.1) and (2.2) numerically, the boundary surfaces are approximated by meshes composed of nodes and elements. The surface integrals are computed over these elements [45], [46], [47], [44].

### 2.2.1 Discretizations

Although analytical solutions exist for simple head models based on the concentric spheres model [26], [28], realistic models are superior in accuracy even if the model is slightly evolved to a realistic one [6],[7]. Complicated nature of the surfaces in the realistic head models makes the analytical calculation of the integrals in (2.1) and (2.2) impossible. To overcome this difficulty, numerical methods are widely used for the FP and the IP [25], [10], [11], [42].

Any finite surface integral can be represented by the sum of surface integrals on all surface patches constituting that surface. If that surface can be represented by a collection of surface elements and if the integration on each element is approximated by a finite sum, then the surface integral can

be computed on each element, and we can numerically solve (2.1) and (2.2) for $\Phi$ and $\vec{B}$ as long as we have the primary source (dipole) information [25].

The most widely used surface element type is the triangle [9],[34],[10]. The surfaces are approximated by a mesh with triangular loops. Accuracy of such an approximation for representing the surface is strictly dependent on the surface shape and the number of elements used in the mesh (Figure 2.2)

The approximation of the surface can be done using higher order (quadratic, cubic) elements instead of triangular (first order $\equiv$ linear) elements. In these cases, the elements are again triangle-originated, but not necessarily planar (see the Appendix A for details). Higher order elements can provide better accuracy in the numerical calculations, especially for the cases with curved surfaces, even with less number of elements [25], [11].

The surface integral on each element also necessitates an approximation. Although, there are piecewise constant field approximations in the early literature, for the EMSI, the field distribution cannot be accepted constant over an element unless the element size is very small (i.e. number of elements is high) [15]. Generally, the most commonly used field distribution approximation is obtained by a linear interpolation, in which, the field at every point on the element is approximated by a linear combination of field values at the nodes (vertices) [6], [10], [48]. For higher order elements, higher order interpolations are possible [25], [11]. The higher order interpolations provide more accurate approximations for fast varying field patterns. For the interpolations of all degree, the field at any point on an element is represented in terms of the field values at nodes. For that, a local domain is defined, at least implicitly [25]. The node coordinates are fixed in the local domain and every point on the local domain element is mapped to the real domain by the interpolation (shape) functions, the local coordinates of that point and the real domain node coordinates.

In this study, the head is modeled using realistic and spherical three-layer BEM meshes representing brain, skull and scalp with second order elements. The BEM formulation used in this study is described by Tanzer and Gençer

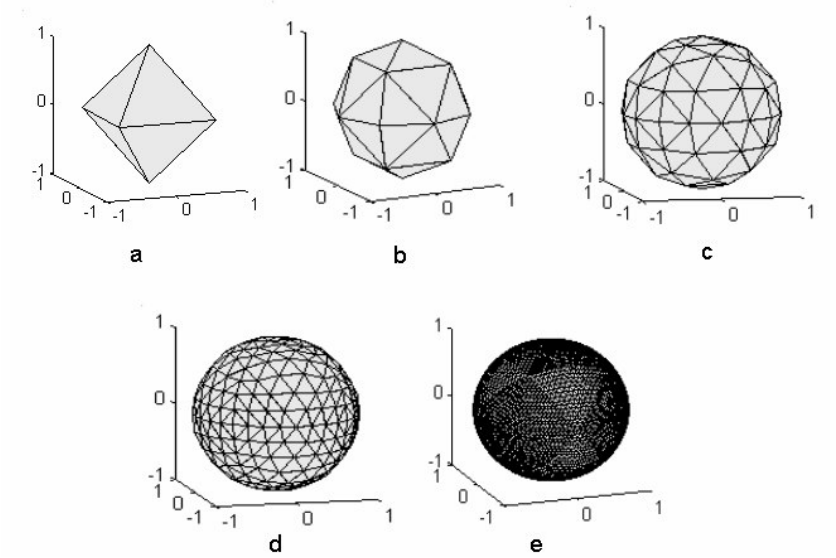


Figure 2.2: Spherical shell approximations using different number of linear elements. Surface modeling accurately increases from (a) to (e).

[25] and further refined by Akalın-Acar and Gençer [12]. The formulation uses triangular elements. The interpolation of coordinates and surface potentials over the element is performed using shape functions which map a triangular element defined in a local coordinate space to the actual element. Such a mapping is called *isoparametric* since both the position and the potentials are approximated using the same shape functions (see Appendix A). The order of the shape functions determine the order of the mesh. Linear, quadratic and cubic shape functions require three, six and ten nodes for each element, respectively. Quadratic elements provide a good balance between accuracy and mesh complexity [25].

After the interpolation functions are described, the surface integral over the element must be approximated by a finite sum. For that, Gauss-Legendre quadrature [49] with 13 points is used (see Appendix B).

Using the above approximations, we obtain the following discrete equation set [25]:

$$\phi_i = g_i + \frac{1}{2\pi} \sum_{k=1}^{K} \left\{ \left( \frac{\sigma_k^- - \sigma_k^+}{\sigma_i^- - \sigma_i^+} \right) \sum_{j=1}^{M_k} \left( \frac{1}{2} \sum_{l=1}^{gp} \phi_l \omega_l \frac{\vec{R}_l}{R^3} \cdot \vec{n}_l G_l \right) \right\} \tag{2.5}$$

This equation gives the potential at the $i^{th}$ node by summing the contributions of $K$ boundaries. Note that the $k^{th}$ boundary has $M_k$ patches and $M_k$ may be different for every boundary. Here, $g_i$ is the field created by the dipole in an unbounded case given by (2.3), $gp$ is the number of Gauss-Legendre quadrature points (13 in this study), $\phi_l$ is the potential at the $l^{th}$ Gauss-Legendre point, $\omega_l$ is the weight for that point, $n_l$ is the normal and $G_l$ is the Jacobian due to transformation from the local domain to the global domain. See the Appendix A and Appendix B for the details.

### 2.2.2 Linear Equations

As explained in section 2.2.1, the surface integral on each element is approximated as a weighted sum of potentials at sample points on the element. These sample points are selected in the local domain according to Gaussian quadrature [49] and the electric potential at each sample point is expressed in terms of node potentials using interpolation functions (see Appendix B). If the field point is close to an element, the gaussian integration over that element does not provide acceptable accuracy in the solutions. A recursive integration scheme, in which the element of integration is divided into subelements is implemented to reduce this effect [11]. After processing every element, equation (2.1) can be expressed as a system of linear equations with $N$ unknowns where $N$ is the number of nodes in the BEM mesh.

$$\Phi = \mathbf{g} + \mathbf{C}\Phi. \tag{2.6}$$

In this equation, $\mathbf{g}$ and $\Phi$ are $N \times 1$ vectors representing the electric potential due to primary sources and total electric potential at nodes, respectively.

Defining $\mathbf{A} = (\mathbf{I} - \mathbf{C})$, the matrix equation becomes,

$$\mathbf{A}\Phi = \mathbf{g}. \tag{2.7}$$

where $\mathbf{A}$ is an $N \times N$ matrix which represents the head geometry and conductivity distribution.

To eliminate the singularity in $\mathbf{A}$, a single deflation is performed on the matrix [50], [9]. The deflated $\mathbf{A}$ becomes:

$$\mathbf{A} = \mathbf{I} - \left(\mathbf{C} - \frac{1}{N}\mathbf{U}\right) \tag{2.8}$$

Here, $\mathbf{U}$ is an N×N matrix with all entries equal to unity. Deflation can be considered as taking the average of node potentials as the reference.

The low conductivity of the skull causes numerical errors in the forward problem solutions. The Isolated Problem Approach (IPA) is implemented to overcome this problem [9], [8]. In this approach, first the brain-skull interface is isolated from other layers (brain inside and air outside) and this problem is solved for the given dipole. Obtained potentials are used to modify the actual equation. This is done by replacing the right-hand-side of (2.7) with a corrected one that includes the effect of skull conductivity:

$$\mathbf{h} = \frac{\sigma_3^+}{\sigma_3^-}\mathbf{g} - \frac{2\sigma_3^+}{\sigma_3^+ + \sigma_3^-}\Phi_{IPA} \tag{2.9}$$

Here, $\Phi_{IPA}$ is a vector containing the IPA solutions for brain-skull interface nodes and zero elsewhere. $\sigma_3^+$ and $\sigma_3^-$ denote the skull and brain conductivities, respectively.

Therefore, two distinct matrices must be generated and two linear systems must be solved for forward problem solutions. IPA matrix (let us call it $\mathbf{A_s}$) could be extracted directly from $\mathbf{A}$. Since the outer conductivity of the brain-skull interface (conductivity of the skull) is replaced by air conductivity for the isolated problem, the entries of $\mathbf{A}$ matrix must be transferred to the $\mathbf{A_s}$ matrix by appropriate scaling.

Note that, both $\mathbf{A_s}$ and $\mathbf{A}$ remains the same for different dipole orientations. Changing dipole location and/or orientation changes only the right-hand-side of the equation system since the dipole information is inserted only to $\mathbf{g}$ as seen in (2.1) and (2.3).

For the magnetic field, equation (2.2) leads to another linear system of equations:

$$\mathbf{B} = \mathbf{B_0} + \mathbf{H}\Phi \tag{2.10}$$

where $\mathbf{B}$ is an $N \times 1$ vector keeping the radial component of the magnetic field. (MEG coils are assumed to be placed radially). $\mathbf{H}$ is a similar matrix generated from (2.2). Hence, the solution of the forward problem is reduced to solving a linear system of equations.

Inverting the system matrix $\mathbf{A}$ is a time-consuming task since $\mathbf{A}$ is dense and large. Furthermore, $\mathbf{A^{-1}}$ is not completely needed for the inverse problem. Rather, the rows of $\mathbf{A^{-1}}$ that correspond to electrode positions is sufficient. Therefore, when iterative methods are used, forward problem solution can be modified to compute the electric potential at the measurement nodes only [12]. For this purpose, the Accelerated BEM method developed by Akalın-Acar and Gençer is implemented [12]. The Accelerated BEM method computes transfer matrices for EEG and MEG and uses these matrices to solve the forward problems using simple matrix-vector multiplications. The method also includes formulations for using IPA when there are arbitrary number of layers inside and outside the skull compartment [13]. After computing the transfer matrices for the electric field ($\mathbf{E}$) and for the magnetic field ($\mathbf{M}$), the set of equations for the forward problem solution on a three-layer head model becomes:

$$\Phi_e = \mathbf{E}\left(\frac{1}{\sigma_3}\mathbf{g} - \frac{2\sigma_2}{\sigma_2+\sigma_3}\mathbf{A}_s^{-1}\mathbf{g}_3^0\right) \tag{2.11}$$

$$\mathbf{B} = \mathbf{B_0} + \mathbf{M}\left(\frac{1}{\sigma_3}\mathbf{g} - \frac{2\sigma_2}{\sigma_2+\sigma_3}\mathbf{A}_s^{-1}\mathbf{g}_3^0\right) + \mathbf{H}_3\mathbf{A}_s^{-1}\mathbf{g}_3^0 \tag{2.12}$$

where $\Phi_e$ represents the electrode potentials, $\sigma_2$ and $\sigma_3$ are the skull and brain conductivities, respectively. $\mathbf{A}_s^{-1}$ is the inverse of the coefficient matrix for the isolated model, $\mathbf{g}_3^0$ is the right hand side for the isolated model, $\mathbf{H}_3$ is the sub-matrix of $\mathbf{H}$ matrix corresponding to the nodes of the isolated model.

## 2.3 Solutions of the Linear Equations

There are various methods to solve linear systems of equations. They are classified using two main types, namely direct and iterative methods.

### 2.3.1 Direct Methods

The most straightforward methods for solving linear systems of equations are based on Gaussian elimination and are called direct methods. Direct methods are applicable to all kinds of linear systems. As long as a system of equations has a unique solution, direct methods guarantee finding that solution for exact arithmetic. However, direct methods are computationally very expensive. Gaussian elimination based Matrix inversion is $O(n^3)$.

If the system is to be solved successively for varying right-hand sides, more efficient direct methods are applicable. The most commonly used direct method is the LU decomposition [51], in which A is represented by a multiplication of an upper and a lower triangular matrix as:

$$\mathbf{A} = \mathbf{LU} \tag{2.13}$$

LU decomposition complexity is again $O(n^3)$. However, once the LU decomposition is obtained for a system matrix, system could be solved for varying right-hand-sides in $O(n^2)$ time by back-substitution ($\mathbf{Ly}=\mathbf{b}$, $\mathbf{Ux}=\mathbf{y}$). This is same with the solution complexity after matrix inversion (matrix-vector multiplication). Storage needs for LU decomposition is again same with matrix inversion. The major advantage for LU decomposition appears in finite precision arithmetic since back-substitution provides better accuracy

than multiplication with the system matrix inverse [52]. However, accelerated BEM is not applicable with LU decomposition, since the system solutions necessitate the entire decomposition.

For large BEM equations, direct methods are inappropriate due to their high computational complexity ($O(n^3)$). Furthermore, the entire matrix inverse is not needed. There is no way to construct only the selected rows of the inverse matrix by direct methods other than calculating the entire inverse or a very naive approach (Calculating LU decomposition in $O(n^3)$ time and then calculating $\mathbf{E}$ in $O(mn^2)$ time by solving $m$ equations using $\mathbf{LUx_i} = \mathbf{b_i}$).

At this point, as long as the matrix is well-conditioned, iterative methods have considerable advantage since $\mathbf{E}$ can be constructed in $O(mn^2)$ time without the need for a complete inversion or LU decomposition.

### 2.3.2 Iterative Methods

Iterative methods are based on making an initial guess for the solution and then improving this guess in successive steps. There are various iterative methods for the solution of linear equation systems.

One type of iterative methods covers *stationary iterative methods* [20]. These methods have a general form:

$$\mathbf{x_{k+1}} = \mathbf{G}\mathbf{x_k} + \mathbf{c} \tag{2.14}$$

Here, $\mathbf{x_{k+1}}$ is the next iterate (guess), $\mathbf{x_k}$ is the current guess, $\mathbf{c}$ and $\mathbf{G}$ are a constant vector and a matrix, respectively. $\mathbf{G}$ and $\mathbf{c}$ are chosen so that the function $g(\mathbf{x}) = \mathbf{G}\mathbf{x} + \mathbf{c}$ is stationary at the solution point $\mathbf{x}$. That is, if $\mathbf{x_i} = \mathbf{x}$, then $\mathbf{x_{i+1}} = \mathbf{x}$. The general approach is to split the system matrix $\mathbf{A}$ or its preconditioned version (the right hand side will also be modified in such a case). Common examples to this family of methods are:

1. The Jacobi Method:

$$\mathbf{x_{k+1}} = -\mathbf{D^{-1}}(\mathbf{L} + \mathbf{U})\mathbf{x_k} + \mathbf{D^{-1}}\mathbf{b} \tag{2.15}$$

2. The Gauss-Seidel method:

$$\mathbf{x_{k+1}} = -(\mathbf{D}+\mathbf{L})^{-1}\mathbf{U}\mathbf{x_k} + (\mathbf{D}+\mathbf{L})^{-1}\mathbf{b} \tag{2.16}$$

3. Successive Over-Relaxation:

$$\mathbf{x_{k+1}} = (\mathbf{D}+\omega\mathbf{L})^{-1}((1-\omega)\mathbf{D} - \omega\mathbf{U})\mathbf{x_k} + \omega(\mathbf{D}+\omega\mathbf{L})^{-1}\mathbf{b} \tag{2.17}$$

In above methods, $\mathbf{D}$ is the diagonal, $\mathbf{L}$ is the strictly lower triangular and $\mathbf{U}$ is the strictly upper triangular part of the system matrix. $\omega$ in (2.17) is a relaxation parameter such that $1 < \omega < 2$.

If we reconsider (2.14). Since $g(\mathbf{x})$ is stationary at the solution point $\mathbf{x}$, we have

$$\mathbf{x} = \mathbf{G}\mathbf{x} + \mathbf{c} \tag{2.18}$$

Subtracting (2.14) from (2.18) we get

$$\mathbf{e_{k+1}} = \mathbf{G}\mathbf{e_k} \tag{2.19}$$

Where $\mathbf{e_k}$ and $\mathbf{e_{k+1}}$ are the errors for $k^{th}$ and $k^{th}_{i+1}$ iterates, respectively. Equation (2.19) shows that the performance of the stationary methods are strictly dependent on the spectrum of $\mathbf{G}$. If $\mathbf{G}$ has large eigenvalues, the error along corresponding eigenvectors may decay slowly, even diverge. Therefore, for many ill-conditioned matrices, the stationary methods may have very slow convergence or no convergence at all. Preconditioners may provide or improve the convergence for such matrices.

As apparent from (2.19), the stationary iterative methods tend to reduce the high-frequency (i.e., oscillatory) components (corresponding to small eigenvalues) of the error rapidly, but reduce the low-frequency (i.e., smooth) components of the error much more slowly, which produces poor asymptotic rate of convergence. For this reason, such methods are also called *smoothers*.

Another widely used family of iterative methods is *Krylov Subspace Methods* (KSMs) [20]. The idea behind the KSMs is similar to that of the steepest

descent algorithm. The difference is that the KSMs proceed using a set of basis vectors for the solution space (rather than a blind, gradient-directed way) and use these vectors as search directions for the solution. These basis directions are generally determined at each step using the residual and the previous search direction vector(s).

The Krylov subspace is defined as:

$$K_k = span(r_0, \mathbf{A}r_0, \mathbf{A}^2 r_0 ... \mathbf{A}^{k-1} r_0) \tag{2.20}$$

All KSMs generate basis directions to span this space and eliminate the error along a basis vector at every iteration. The difference among them is in how they construct the basis vector set. Since the basis vectors are required to be linearly independent, KSMs converge at most in $N$ steps (for exact arithmetic), where $N$ is the dimension of the solution space. The reader is invited to refer to [53] for the proof that (2.20) is a space that contains the solution of interest (which is beyond the scope of this study).

The most popular KSM is the Conjugate Gradients (CG) method [54]. The theory behind the CG method is based on the calculus of variations: For any symmetric and positive-definite matrix $\mathbf{A}$, the quadratic function $Q(\mathbf{x}) = \frac{1}{2}\mathbf{x^T A x} + \mathbf{x^T b}$ is minimized for $\mathbf{x}$ that satisfies $\mathbf{Ax} = \mathbf{b}$. Therefore, solving the system $\mathbf{Ax} = \mathbf{b}$ is converted to an optimization problem for which the residual $(\mathbf{b} - \mathbf{Ax_i})$ is to be minimized. Although this method requires positive-definite and symmetric matrices (which is not the case for BEM matrices), it is presented here to provide the basic idea behind KSMs:

$$\begin{aligned}
&\mathbf{d_0} = \mathbf{r_0} = \mathbf{b} - \mathbf{Ax_0} \\
&\alpha_i = \tfrac{\mathbf{r_i^T r_i}}{\mathbf{d_i^T A d_i}} \\
&\mathbf{x_{i+1}} = \mathbf{x_i} + \alpha_i \mathbf{A d_i} \\
&\mathbf{r_{i+1}} = \mathbf{r_i} - \alpha_i \mathbf{A d_i} \\
&\beta_{i+1} = \tfrac{\mathbf{r_{i+1}^T r_{i+1}}}{\mathbf{r_i^T r_i}} \\
&\mathbf{d_{i+1}} = \mathbf{r_{i+1}} + \beta_{i+1} \mathbf{d_i}
\end{aligned} \tag{2.21}$$

As seen in (2.21), CG method generates search directions that are A-orthogonal (i.e., $\mathbf{d_i^T A d_{i+1}} = 0$). Additionally, the error is minimal over the space spanned by the search directions used so far, since the line search parameter $\alpha$ can be determined analytically. Therefore, the CG method provides an impressive convergence performance (especially for well-conditioned matrices), although it is applicable only for positive-definite and symmetric matrices. For ill-conditioned matrices, preconditioners may provide considerable improvement in convergence.

Modifications to CG also exist to cover generalized cases. All KSMs are based on the idea behind CG method. They differ in how the basis set for the Krylov subspace is generated. For the solution of BEM equations, we need methods that are applicable to general (i.e., not positive-definite or symmetric) matrices. The algorithms for tested KSMs through thesis work are presented in Appendix C. Here we present the characteristic properties of applied methods in table 2.1 [21].

Table 2.1: Properties of some KSMs that are applicable to dense matrices

| Algorithm | Convergence depends on | Number of matrix-vector products | Number of operations | Needed memory space |
|---|---|---|---|---|
| CGNE | $\mathbf{A^T A}$ | $2i$ | $2i$ | $n^2 + 6n$ |
| CGS | $\frac{1}{2}\mathbf{A}$ | $2i$ | $2i$ | $n^2 + 11n$ |
| BiCG | $\mathbf{A}$ | $2i$ | $2i$ | $n^2 + 10n$ |
| BiCGSTAB | $\frac{1}{2}\mathbf{A}$ | $2i$ | $4i$ | $n^2 + 10n$ |
| GMRES($\infty$) | $\mathbf{A}$ | $i$ | $\frac{i}{2}(i+2)$ | $n^2 + (i+5)n$ |

The major difference in algorithm complexity for iterative methods is

that the main criteria is the convergence rate since the total number of operations is strictly dependent on the number of iterations performed for convergence. It is also apparent from the table that the GMRES method has significant advantages since it requires only $i$ matrix-vector multiplications and its convergence is dependent on A only, although its needed memory space is also dependent to the number of iterations. This drawback is eliminated by restarting the algorithm after some number of iterations.

The there are two different measures for the KSMs for stopping iteration due to convergence: absolute error tolerance (atol) and relative error tolerance (rtol). atol is the maximum acceptable value of residual vector norm. rtol is the maximum acceptable ratio of residual norm at any an iteration over the initial residual norm. The values for atol and rtol are $10^{-50}$ and $10^{-5}$, respectively (for all of our experiments, convergence is detected by the KSMs due to rtol. Although these values provide safe operation, it is observed that, the results are not affected when rtol is increased to $10^{-2}$, while the operations are fastened). The matrix fill time for the selected rows of system matrix inverse could be fastened by changing these error tolerances, as long as they are safe.

## 2.4 Complexity for Phases of BEM

The first step in the FP solutions is to obtain the coefficient matrix $\mathbf{A}$ of the BEM equation (2.7). Matrix filling complexity is straightforward for single processor case, since matrix is filled using a standard algorithm. For every field node (row of $\mathbf{A}$), every element in the mesh is visited for surface integration. On every element, the surface integral is approximated by a weighted sum of potentials at the Gauss-Legendre quadrature points. The potential at a Gauss-Legendre quadrature point is represented in terms of element's node potentials using the interpolation functions and contribution of each node is then added to its column in matrix $\mathbf{A}$. The number of floating point operations (*flops*) is proportional to $N \times E \times N_{gp} \times N_{npe} \times C_{int}$, where

$N$ is the total number of nodes, $E$ is the total number of elements, $N_{gp}$ is the number of Gauss-Legendre points, $N_{npe}$ is the number of nodes per element and $C_{int}$ is the complexity of recursive integration.

Once the coefficient matrix $\mathbf{A}$ is obtained, the transfer matrix $\mathbf{E}$ is calculated by solving a matrix equation of size $N \times N$ for $m$ different right hand sides. In this stage, the total number of matrix-vector multiplications is $m \times it$ where $it$ denotes the number of iterations depending on the mesh size and the KSM used.

In this study, we preferred to use a single library (PETSc) for all matrix operations and linear equation solutions. Thus, the inverse of the coefficient matrix $\mathbf{A_s}$ in the isolated problem equation is obtained again using the GMRES implementation of PETSc. Using a single library for computations prevents useless input/output operations and additional code for data compatibility problems. The computational complexity of calculating $\mathbf{A_s^{-1}}$ is $2N^2 \times N_{mv} \times m$, where $N$ is the number of nodes and $N_{mv}$ is the number of matrix-vector multiplications.

As given in equation (2.11), the accelerated BEM approach reduces to simple matrix-vector multiplications. Once $\mathbf{E}$ and $\mathbf{A_s^{-1}}$ are obtained, the main computational load of the FP solution shifts to IPA solutions.

Magnetic field calculations is very similar except an additional matrix-vector multiplication at each FP solution, as given in 2.12. This brings flops proportional to $m \times N_{br}$ where $N_{br}$ is the number of the nodes on the brain-skull interface [12]. Note that, the same coefficient matrix $\mathbf{A}$ is used for the computation of $\mathbf{M}$.

In this study, various KSMs that are applicable to general dense matrices are tested and the fastest method (GMRES) is used for the solution of the BEM equations. These methods are introduced in Appendix C for completeness and providing a basis for interpretation of the results.

# CHAPTER 3

# PARALLELIZATION

BEM matrices are dense and large. Since finer meshes provide more accurate results, BEM implementers need their meshes as fine as possible. The limiting factor is the computational resources. There are two main limitations: computational power, so time; and available memory.

For a mesh of 15000 nodes, the resulting system matrix will be $15000 \times 15000$. For both forward and inverse problems, it is certain that, at least matrix-vector multiplications (which are $O(n^2)$) are unavoidable. If the entire matrix inverse is to be calculated, the cost is $O(n^3)$. And as presented in 3.2, each iterative solution performs at least $i \times n \times (2n-1)$ operations. The overall cost is higher for the inverse problem. For each forward problem solution (as a part of inverse problem solver), we need a matrix-vector product with a matrix of size $m \times N$, where $N$ is the total number of nodes and $m$ is the number of detectors. Since inverse problem has an under-determined nature, global optimization methods (such as the evolutionary methods), which necessitate many cost function evaluations (i.e. matrix-vector multiplications), are required. Furthermore, for each forward problem, IPA solutions are needed to correct right-hand side of the equation system. This, in fact is the dominant computational load of the inverse problem iterations (cost function evaluation).

For a 15000 noded mesh, the memory need for a double precision BEM matrix is 1.8GB. If the CSF-Skull interface has 5000 nodes, then another 200MB is needed for the matrix that will be used to solve the isolated problem. Though, these two matrices are not needed at the same time.

These facts constitute the basis of the motivation for high performance

computing through parallel processing for the EMSI of human brain by the BEM. The main consideration in parallelization is to provide a cheap, efficient and scalable parallelization scheme. For that, a parallel cluster of 8 personal computers is constructed on a fast local area network and a scalable single-instruction-multiple data (SIMD) code is developed using a number of scientific computation libraries.

## 3.1 The Marvin Cluster

For parallel processing, the parallel computing cluster at the Brain Research Laboratory is used. The cluster named *Marvin* was built in 1996 and consisted of four dual processor workstations, and a single processor controller workstation. Eight single processor nodes has been added to the cluster through the thesis work.

Figure 3.1 illustrates the present configuration of the cluster. The older dual processor nodes are called *Nodelins* and the eight new nodes are called *Athlins*. The four Nodelin workstations have dual PIII 933Mhz processors, for a total of eight nodes. They are connected to each other over a 100Mb/s Ethernet switch. The Athlin nodes are each AMD Athlon XP 1.83GHz workstations, connected to each other over a gigabit Ethernet switch. All cluster nodes are running under the Linux operating system, and the controlling workstation is running under FreeBSD, and provides access to the cluster nodes.

## 3.2 Parallel Libraries and Iterative Solvers

The most popular parallel communication libraries are Message Passing Interface (MPI) [23],[55] and Parallel Virtual Machine (PVM) [38]. These libraries provide basic inter-processor communication and parallel I/O routines.

Basic linear algebra libraries are: Basic Linear Algebra Subprograms

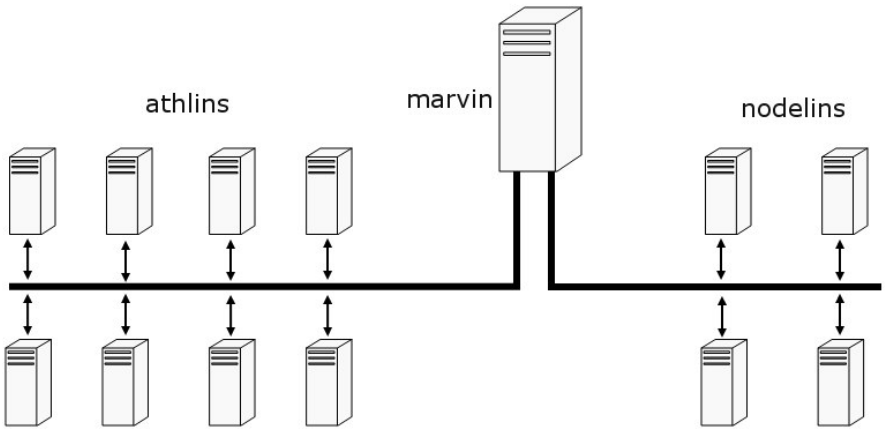


Figure 3.1: The Marvin cluster structure. The cluster is composed of two groups: athlins have single AMD 2500+ (1833MHz) processors on each PC, nodelins have dual PIII (933MHz) processors on each PC.

(BLAS) [56], [57], Basic Linear Algebra Communication Subprograms (BLACS) [58], and Parallel Basic Linear Algebra Subprograms (PBLAS) [59]. The latter two libraries are derivatives of BLAS and provide same family of operations, namely the basic (low level) linear algebra routines (matrix and vector operations). Although they can be used for application development, the programmer needs to implement many functions for real world applications.

BLAS performance can be tuned according to used hardware configuration (processor type and speed, data bus and RAM speed, etc.) For that Automatically Tuned Linear Algebra Subprograms (ATLAS) [60], [61] library is available. This library is run once and provides optimized BLAS routines. ATLAS provides significant improvement in performance, especially for dense matrix operations.

There are intermediate level libraries to complete missing operations in basic linear algebra libraries. Most commonly used one is Linear Algebra Package (LAPACK) [62], [63]. Lapack is an extension package to BLAS providing higher level linear algebra routines.

High-level scientific computation libraries can be classified into two main groups: Direct solvers (Scalable Linear Algebra Package (ScaLAPACK) [39] [64], Parallel Linear Algebra Package (PLAPACK) [40]) and iterative solvers (Portable, Extensible Toolkit for Scientific Computations (PETSc) [22], [65], AztecOO [66], High performance preconditioners (Hypre) [67]). The latter is a larger family. Both families use the low-level inter-processor communication and the basic linear algebra libraries. The direct solvers are focused on dense systems and the iterative solvers are focused on sparse systems caused by numerical methods such as Finite Element Method (FEM) and Finite Difference Methods (FDM). However, the iterative solvers provide the data types, operations and applicable solvers for dense matrices. There is no iterative solver library that is focused on dense matrices; therefore, one must use these solvers to use the iterative methods on dense matrices.

The details on these libraries are vast and the user is invited to refer to the references for further information.

The computation nodes of The Marvin Cluster have the following tools and libraries for parallel processing, and numerical operations:

- Message Passing Interface (MPI) [23].
- Parallel Virtual Machine (PVM) [38].
- Basic Linear Algebra Subprograms (BLAS)
- Linear Algebra Package (LAPACK) [62].
- Portable, Extensible Toolkit for Scientific Computation (PETSc)] [22].
- Octave (an interactive numerical computation environment similar to MATLAB.) [68]

In this work, the PETSc library is used for all tasks, including matrix fill ($\mathbf{A}$ and $\mathbf{A_s}$), construction of $\mathbf{E}$ (using iterative solvers), and matrix-vector operations. PETSc uses MPI, BLAS, and LAPACK. The library is written in C and provides;

1. Parallel matrix and vector data types,
2. All necessary parallel matrix and vector operations,
3. Almost all KSMs in literature (both for sparse and dense matrices)

PETSc library requires a number of lower level libraries installed on the system. The procedure for a complete PETSc installation is:

1. Configuring and installing MPI,
2. Configuring and running ATLAS (at the end, an optimized BLAS is available),
3. Configuring and installing LAPACK using optimized BLAS,
4. Configuring and installing PETSc.

The main limitations of PETSc in the BEM application are the available matrix distribution type and transposition. PETSc provides only row-wise distribution for matrices (column-wise distribution is more efficient in matrix-vector multiplications). Matrix transposition is not possible for very large matrices (library runs out of memory), which is the case for BEM matrices. This second draw-back can be eliminated by constructing the transpose directly, but matrix distribution limitation reduces the speed-up in matrix fill and individual forward problem solutions slightly (refer to chapter 6 for details).

## 3.3 Data Partitioning

The aim of using a parallel cluster is not only to reduce computation time but to combine the memory resources of the nodes. This allows the solution of problems that can not be solved using a single workstation. When distributing the problem into the cluster, the following resources are considered:

The mesh data (i.e., the node coordinates and elements used for computing the BEM coefficient matrix) are not memory consuming. For a mesh of even $10^5$ nodes and $10^5$ elements, the total size of mesh information is below 5 MB. Hence, the complete mesh information can be kept on each processing node. Additionally, the BEM system matrices are dense, in other words, every node's contribution to any other node must be computed. Therefore, keeping mesh information on every processor provides faster processing for matrix filling.

The BEM coefficient matrix, $\mathbf{A}$, is distributed among processors as blocks of rows. On Athlins and Nodelins, since these clusters are homogeneous, each processor keeps an equal number of successive rows. And each processor computes the terms of its rows only. Such matrix filling strategy minimizes inter-processor communication time since processors do not send any matrix information during matrix assembly, except some status check data. For the problems where number of nodes is much larger than the number of processors, such a matrix filling scheme will provide a speed up very near to the number of processors.

We implement a modified filling scheme for matrix $\mathbf{A}$ as $\mathbf{A^T}$ is used in finding the selected rows of $\mathbf{A^{-1}}$. Since the matrix transposition is an expensive process, we preferred to construct $\mathbf{A^T}$ directly. Such matrix filling is not straightforward for row-based matrix distribution since in such a scheme, instead of field nodes, source nodes are distributed among the processors. Contribution of source nodes to the potential at a field node is obtained through surface integrals. Thus, for efficient computation, more complicated filling scheme is applied. This scheme is based on the effort to parse the

elements efficiently:

```
for every field node N_f
    set every element as 'not visited'
    for every locally owned source node N_s
        for every element E that N_s belongs
            if element is not visited
                for every gauss point gp of the element
                    for every node N_e of the element
                        if N_e is a locally owned source node
                            calculate and add contribution of N_e
                        end if
                    end N_e
                end gp
                Mark E as 'visited'
            end if
        end E
    end N_s
end N_f
```

Alternative schemes are also possible (transposed version of usual matrix filling with column based matrix distribution), but not with PETSc library.

Speed-up in filling the system matrix **A** strictly depends on the mesh representation, since the matrix is column-wise distributed and the transpose of the matrix is directly computed. That is, nodes are distributed among the processors as secondary sources, not the field points. To prevent inefficient element visits during matrix filling, the mesh information must be preserved so that the nodes of each element take close indices. The elements must also be ordered so that the neighboring elements have close indices. We obtain such mesh representation by a simple method: 1)the nodes of each boundary

are sorted according to their z-coordinates, 2) the new indices are updated in the element data, and 3) the elements are sorted according to the indices of their nodes. Finally, the meshes of different boundaries are merged. Note that, such re-ordering process would not be necessary if column-wise matrix distribution were available.

Distribution of $\mathbf{E}$ among the processors is similar. However, since these rows are calculated by iterative solvers, the solution is kept on a distributed vector after the calculation of each row. The inverse problem necessitates successive matrix-vector multiplications using $\mathbf{E}$ and for faster matrix-vector multiplications, this matrix must also be distributed among processors. In our work, each row of $\mathbf{E}$ is collected from every processor and stored by the appropriate processor. This choice for matrix storage does not bring significant additional communication burden (compared to column-wise distribution) in matrix vector multiplications, as long as the number of detectors, $m$, is larger than the number of processors, $N_{np}$. The ratio $N_{np}/m$ determines the communication burden (for a fixed cluster) in the speed-up. If this ratio is increased significantly, communication burden will increase linearly and be more effective. For such a case, column-wise matrix distribution will be more beneficial.

For solving the matrix equations, iterative solvers of PETSc [22] are used. PETSc library provides both parallel data types and Krylov Subspace Methods (KSMs) compatible with these data types. Although the details and theory of these methods are beyond the scope of this study, we feel the necessity of reporting their performance for the BEM in electro-magnetic source imaging, in the results chapter.

# CHAPTER 4

# THE INVERSE PROBLEM

The inverse problem of the EMSI is defined as finding the source(s) within the brain that cause(s) the measured (EEG/MEG) data. There are about $10^{10}$ of neurons in the brain, and there are many activities at the same time (related to senses, abstract functions, etc.) and even the simplest tasks are performed as a combined activity of these neurons. It is not possible to detect the activity of a single neuron using EEG/MEG measurements. Instead, certain activities are assumed to be performed by a small regions (foci) and the combined activity in a small region is approximated (modeled) as a (resultant) current dipole and the field is accepted to be created by this point source [5].

Furthermore, there is more than one active region at a certain instant. Hence, the measured field will be a combined resultant field. Thus, for accurate localization, multi-dipole models should be applied. Still, for certain experiments (such as EEG/MEG measurements with auditory or visual stimuli), the background activity (noise) can be suppressed by averaging over repetitive measurements. Single dipole model (or limited number of dipoles) may be acceptable for such experiments.

If the number of dipoles is limited, the resultant field can be calculated using superposition (solving the forward problem after adding up the individual right hand sides for each dipole with quasi-static assumption).

On the other hand, for the inverse problem, the available field data is limited by the number of electrodes (measurement points). Fortunately, we can still determine the source with a trial-and-error scheme using (2.9) and the measurement data. However, gradient-based optimization schemes are

not appropriate for this purpose, since the problem is underdetermined with many possible local optimums. For that reason, global optimization methods such as clustering, simulated annealing and evolutionary algorithms must be applied [43].

Global optimization algorithms provide the opportunity to reach global minimum with the cost of high computational complexity (many cost function evaluations) since they include randomness in the search for the optimum point (i.e., there are useless cost function evaluations). The need for global optimization methods makes parallelization more valuable for the BEM inverse problem solutions.

In this study, a basic genetic algorithm is developed and used for EEG inverse problem solutions. The developed algorithm can be used for single and multiple dipole models. Better methods are possible as in [69].

## 4.1 Problem Definition

For the inverse problem, available information is:

- geometry (mesh),
- conductivity distribution,
- measurement data ( left-hand-side of (2.9) )
- the dipole is located within the brain,
- the dipole satisfies (2.9).

For cost function evaluation, an error measure is needed. The data to be compared are the measurement vector, $\mathbf{\Phi_m}$ and the calculated potentials vector, $\mathbf{\Phi_c}$ due to the guess for the dipole. There are various error measures in the literature. Most common ones are; RDM, RDM*, MAG [8]. These error measures are based on $L_2$ norm. To summarize, they are:

$$RDM = \frac{\|\Phi_m - \Phi_c\|}{\|\Phi_m\|} \times 100\% \tag{4.1}$$

$$RDM^* = \left\| \frac{\Phi_m}{\|\Phi_m\|} - \frac{\Phi_c}{\|\Phi_c\|} \right\| \tag{4.2}$$

$$MAG = \frac{\|\Phi_c\|}{\|\Phi_m\|} \tag{4.3}$$

With these information and error measures, a formal definition for the inverse problem of EMSI with BEM on a 3-layer spherical head model (for simplicity) is:

Minimize RDM (or RDM* or MAG),

Subject to:

$$\|\vec{R}_p\| - R_b \leq 0$$

where, $\vec{R}_p$ is the dipole position vector, $R_b$ is the radius of the innermost (brain) layer. For realistic head models, $\vec{R}$ must be an element of the set that contains all position vectors inside the brain layer.

RDM* measures the topological accuracy and ignores dipole strength by normalizing the field, where MAG measures accuracy in field strength and ignores topological accuracy. RDM measures the accuracy in both topology and field strength. However, it is not as precise as RDM* when only topology is concerned (assume dipole strength is known) and MAG when only strength is concerned (assume topology is known). More clearly; a false located dipole may be found as the optimum point in the inverse problem if it compensates the error due to its location by another error in its magnitude, or vice versa. However, our results show that for most cases, RDM has the best performance among these three measures.

## 4.2 Methodology

The genetic algorithm is implemented and tuned. The choice for genetic algorithm is due to the results presented in [42] as it appears to be the most

robust method. The developed algorithm may be used for different number of dipoles (the code supports up to 10 dipoles). The developed algorithm is a genetic algorithm with real coded chromosomes. The user is allowed to change algorithm parameters through command line options.

### 4.2.1 Chromosomes and Generations

Each generation is composed of 20 chromosomes, each having 10 dipoles. Each dipole is represented by three (double precision) real position (x,y,z) and three (double precision) real orientation (px, py, pz) variables resulting to a chromosome length of 60. The choice for how many dipoles are taken into account is left as a user choice. For the results provided here, single dipole model is used.

### 4.2.2 Fitness Function

RDM is chosen as the fitness measure since it accounts for both field strength and topology. And since smaller RDM must correspond to better fitness, 1/RDM is used as the fitness function.

### 4.2.3 Cross-over

Two chromosomes are randomly selected with roulette wheel selection (figure 4.2) from the parent generation. Each chromosome is assigned to a probability equal to the ratio of its fitness to total fitness of all chromosomes. Then a random number is generated between 0 and 1 as a marker (represented as arrow in figure 4.2) and corresponding chromosome is chosen as a parent. Two parents cannot be the same. Then, a random positive integer index (say, rind) is generated. And the two new chromosomes are created using this index and a random real value a ($0 \leq \alpha \leq 1$) as in figure 4.1. For every pair, crossover is performed (with a probability of 95%) or the parents are directly copied to the new generation (with a probability of 5%).

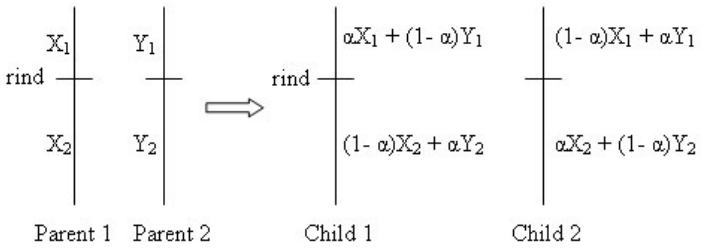


Figure 4.1: Cross-over

### 4.2.4 Boundary Cross-over

Since $\alpha$ is between 0 and 1, for successive generations, above mentioned cross-over scheme will result in an averaging effect, which may cause the algorithm to lose its probabilistic nature, and fail. To avoid such an effect, boundary cross-over is used as follows: at every 20 generations, the worst two chromosomes are replaced by two extreme chromosomes (which are located randomly near the innermost surface and have random amplitude near the maximum allowed value)

### 4.2.5 Mutation

After each cross-over, mutation is performed over the children pairs. Mutation is implemented as adding or subtracting a small real number (up to 3% of innermost radius for position entries and up to 5% of maximum dipole amplitude for orientation entries) from each chromosome entry (variable) with a mutation probability (default value is 20%). Default mutation probability is determined experimentally and kept relatively high with respect to conventional mutation probability values (which is mostly below 5%). This high probability is reasonable since the chromosomes are real coded and short, and

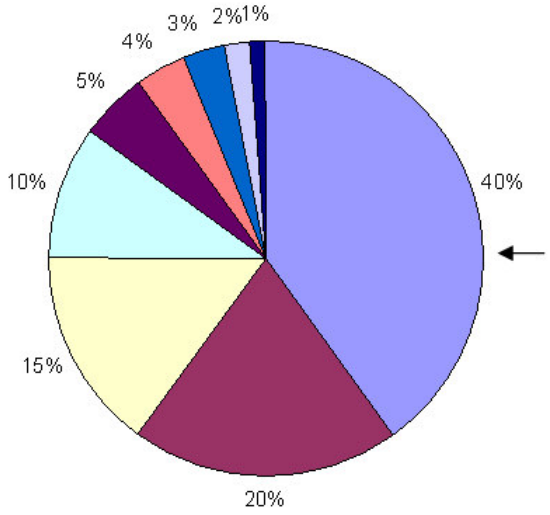


Figure 4.2: Roulette wheel selection example.

the mutation changes the entries slightly (unlike the mutations over binary coded chromosomes).

### 4.2.6 Eliticism

Eliticism provides memory to the genetic algorithm and avoids the algorithm to behave as a random search. For every generation, the best chromosomes (those having highest fitness values) are passed to the next generation without any modification. This will provide the algorithm to move towards the global optimum. Since the cross-over is determined due to fitness values of the chromosomes, the elite eliminate weak seeds. However, if the number of elite is kept high, the algorithm will behave conservative, the convergence will be very slow or the maximum number of generations is reached before

the global optimum is obtained, probably when a local optimum is at hand. The default choice for the number of elite is determined experimentally to be 2 (10% of the number of chromosomes).

### 4.2.7 Stopping Criteria

The algorithm is stopped when a maximum number of generations is reached, or a satisfactory fitness is obtained for a chromosome at some generation before the maximum number of generations is reached. Determining the stopping criteria is in fact a challenge, since the accuracy in forward problem solutions are limited and varying with dipole position. Hence;

i) The algorithm may stop before the global optimum is reached if the satisfactory fitness is kept less than the maximum accuracy for an acceptable dipole position; or the maximum number of generations is given to be smaller than the number that is necessary to reach the global optimum.

ii) The algorithm may continue to the useless trials for better dipoles (even if the global optimum is reached) if the maximum number of generations is too high and the satisfactory fitness is unreachable (due to the forward problem accuracy limitation) for original (or minimum error) dipole position.

Case (i) leads to errors in localization accuracy, and case (ii) leads to large computation time (since each fitness function evaluation costs at least two matrix-vector multiplications). Unfortunately, there is no way to avoid this problem since forward problem accuracy is varying with varying dipole positions and orientations. Forward problem accuracy is changing not only with dipole depth and orientation, but also with its position with respect to electrode distribution (there are no electrodes on face or bottom part of the head) and how fine the mesh is. Still, improvements may be obtained if an initial guess is available for dipole position (the brain region for certain activities is known a priori) and if the algorithm parameters are arranged accordingly.

### 4.2.8 Summary of Algorithm Parameters

The forward problem accuracy is dependent on many parameters, most of which are strictly determined by the unknown; dipole information. As the forward problem accuracy is taken to be the fitness measure, there is no apparent method for tuning the genetic algorithm parameters. The parameters below are determined by many trials. Still, for some positions, these parameters will result in localization errors worse than the limitations of the forward problem.

For mesh 2, if the satisfactory fitness is increased to 1000 (i.e., 0.1% RDM) and the maximum number of generations is increased to 5000, the algorithm is guaranteed to behave according to the forward problem limitations for a single dipole case (refer to chapter 5) with a high computational cost. Finer the mesh, higher the required satisfactory fitness (forward problem solutions are better for finer meshes).

Table 4.1: Default (used) genetic algorithm parameters for a 12294-noded mesh.

| | |
|---|---|
| Number of elite: | 2 |
| Number of parents used in cross-over | 20 |
| Cross-over probability | 0.95 |
| Satisfactory fitness (100/RDM) | 300 |
| Boundary cross-over period | 20 |
| Minimum extreme dipole radius (boundary cross over) | 0.9 × innermost radius |
| Actual dipole amplitude | $5 \times 10^{-6}$ A.m |
| Maximum dipole amplitude | 6 × actual dipole amplitude |
| Minimum extreme dipole amplitude (boundary cross over) | 0.75 × max dipole amplitude |
| Maximum number of generations | 2000 |
| Mutation probability | 0.20 |
| Location perturbation (for mutation) | 0.03 × innermost radius |
| Amplitude perturbation (for mutation) | 0.05 × max dipole amplitude |

# CHAPTER 5

# RESULTS

In this section, first the accuracy of the BEM solutions implemented by the parallel code and platform are presented. Then, the computer resources required to solve forward problems with realistic head models are discussed. Next, the performance of parallel KSMs used in the transfer matrix calculations are compared. Finally, the speed-up measures obtained by parallelization of different BEM phases are presented.

## 5.1 Used Head Models

For the forward and the inverse problems, four head models (meshes) are used:

1. Mesh 1: 1536 elements, 3078 nodes, spherical Rush & Driscoll head model [70]

2. Mesh 2: 6144 elements, 12294 nodes, spherical Rush & Driscoll head model

3. Mesh 3: Realistic head model with 7499 elements, 15011 nodes (figures 5.1, 5.2, 5.3),

4. Mesh 4: Realistic head model with 14999 elements, 29799 nodes (figures 5.4, 5.5, 5.6).

Spherical meshes are generated from an octahedron iteratively by keeping the mesh as uniform as possible. Realistic meshes are generated from segmented 3-D NMR images of Dr. Zeynep Akalin Acar's head using an adaptive

skeleton climbing code. For both spherical and realistic models, initially, a mesh of linear elements is obtained. Then, a new node is placed at the center of every element edge and that node is pulled to its actual position (sphere surface for spherical models and the nearest node of an extremely fine mesh for realistic models).

Each of these four meshes has layers containing equal (spherical meshes) or very similar (realistic meshes) number of nodes and elements.

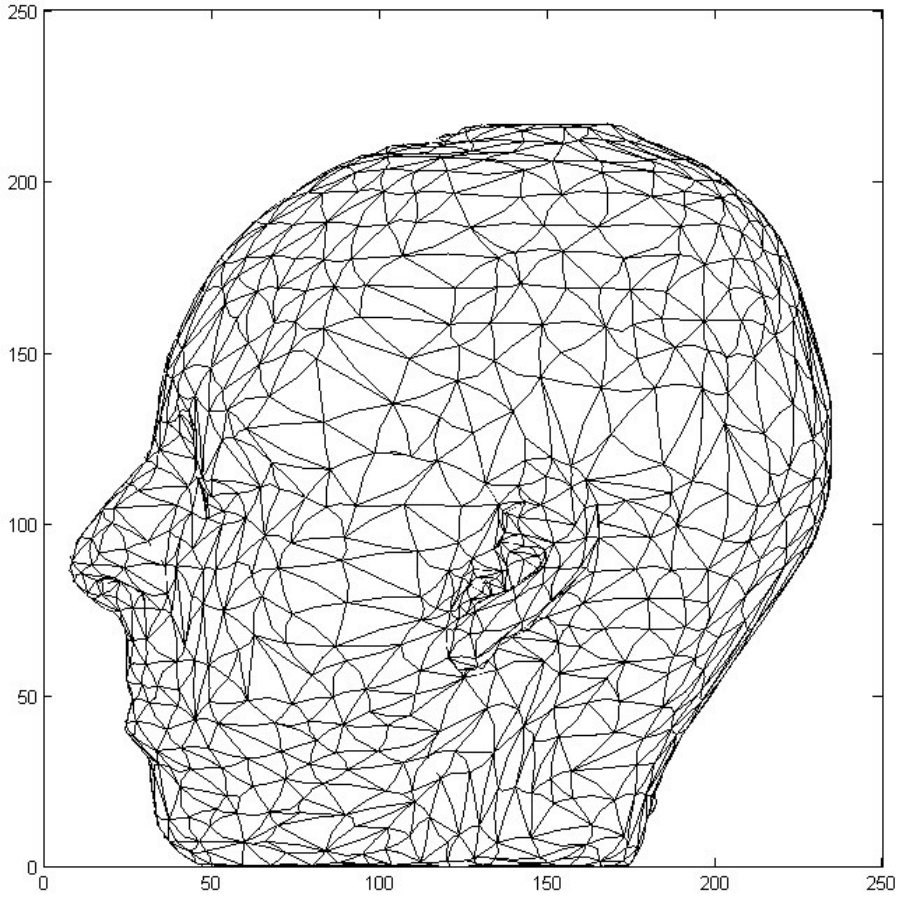


Figure 5.1: Scalp of mesh 3 (units are in mm).

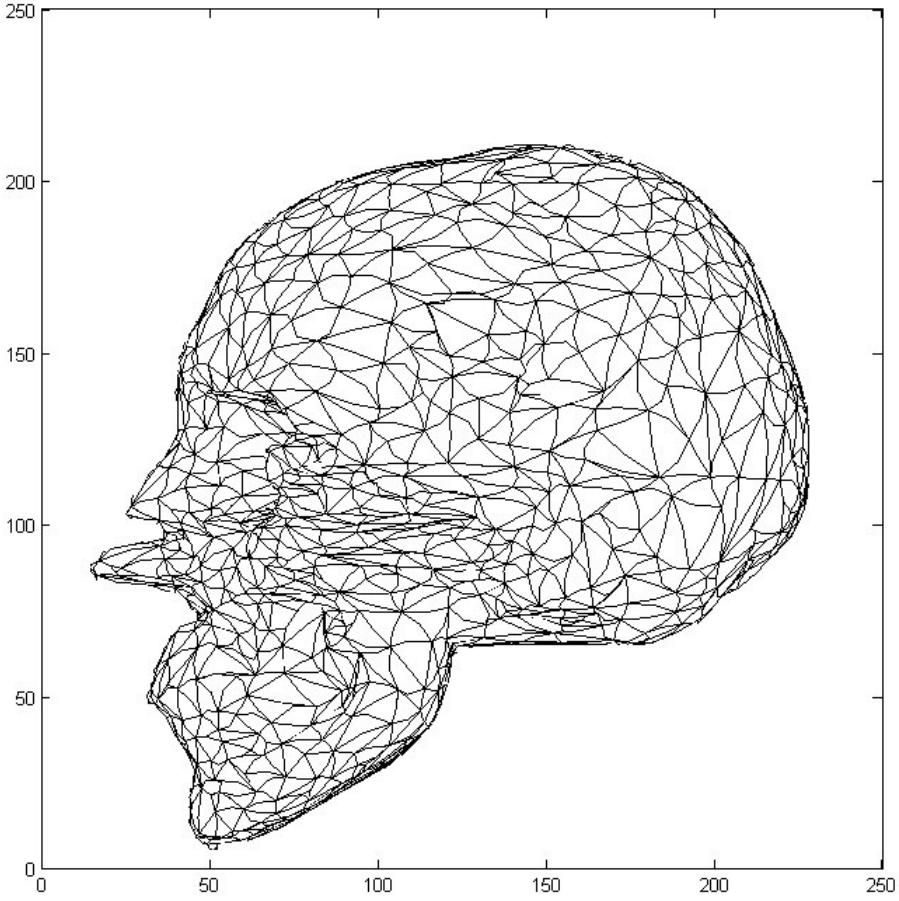


Figure 5.2: Skull of mesh 3 (units are in mm).

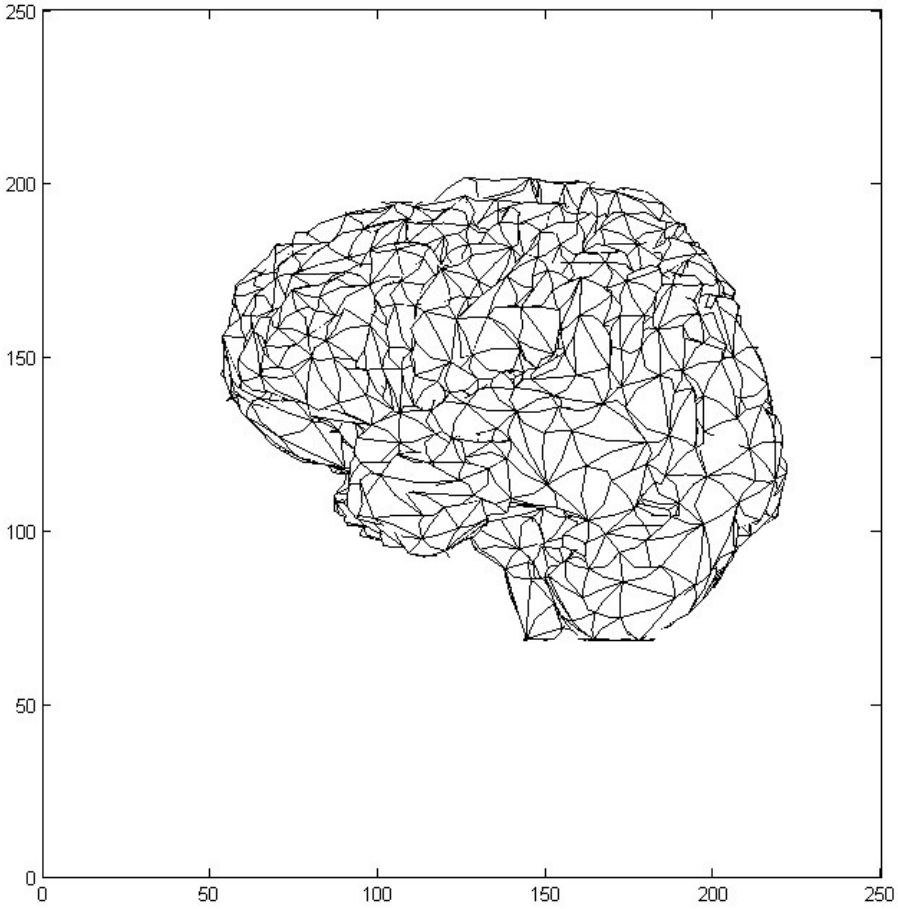


Figure 5.3: Cortex of mesh 3(units are in mm).

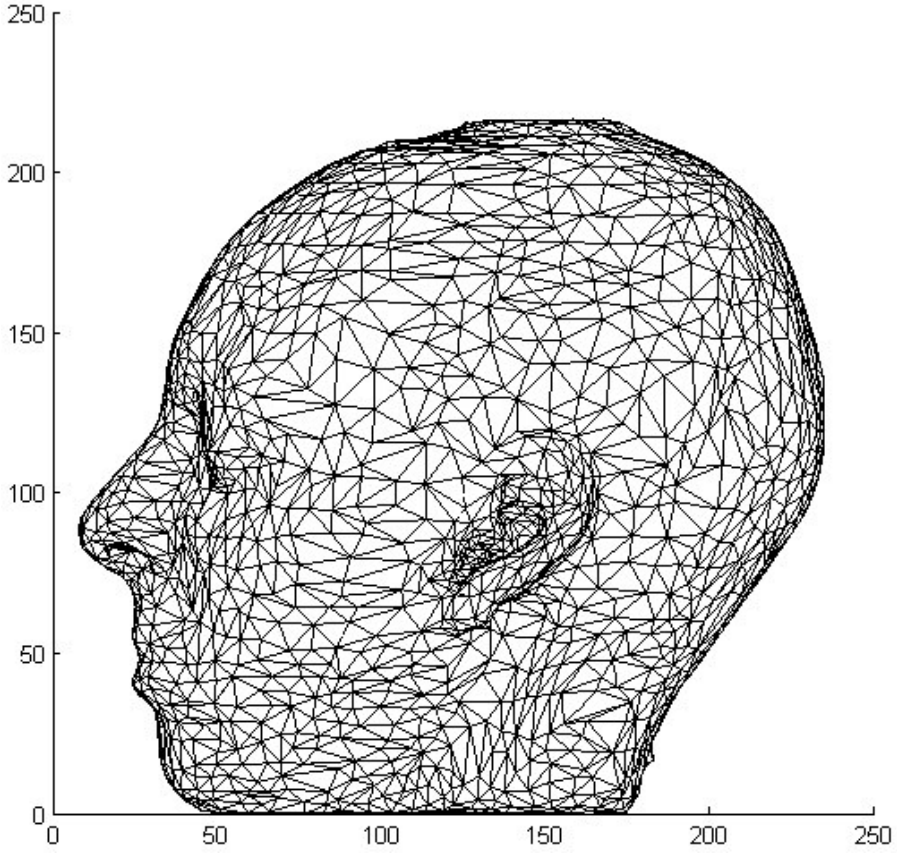


Figure 5.4: Scalp of mesh 4 (units are in mm).

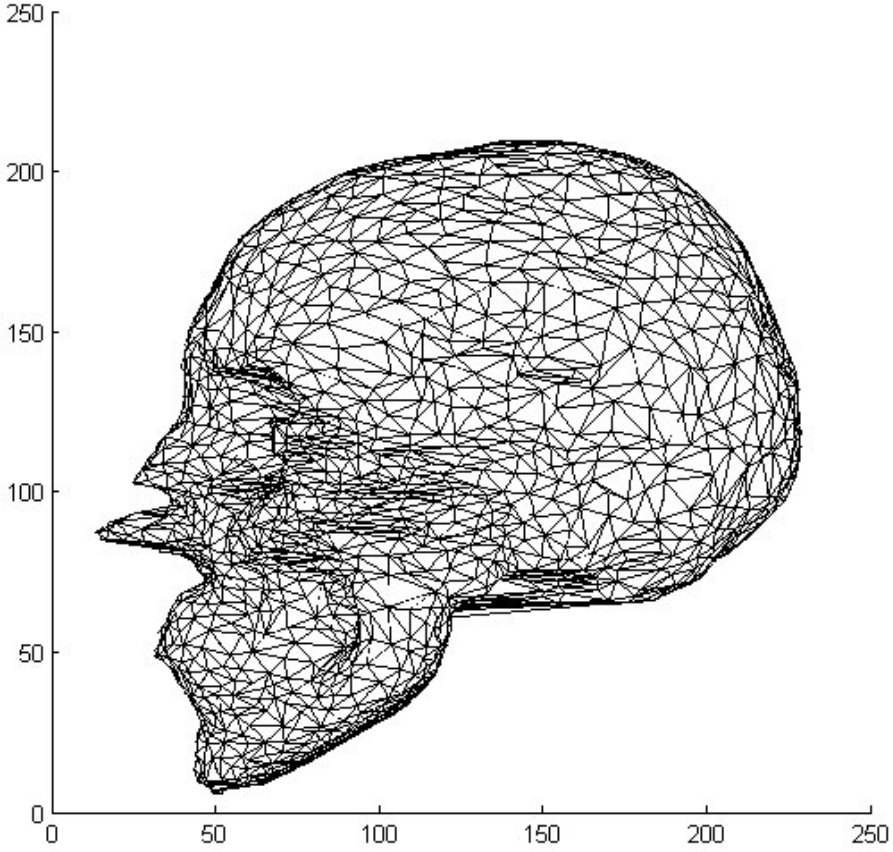


Figure 5.5: Skull of mesh 4 (units are in mm).

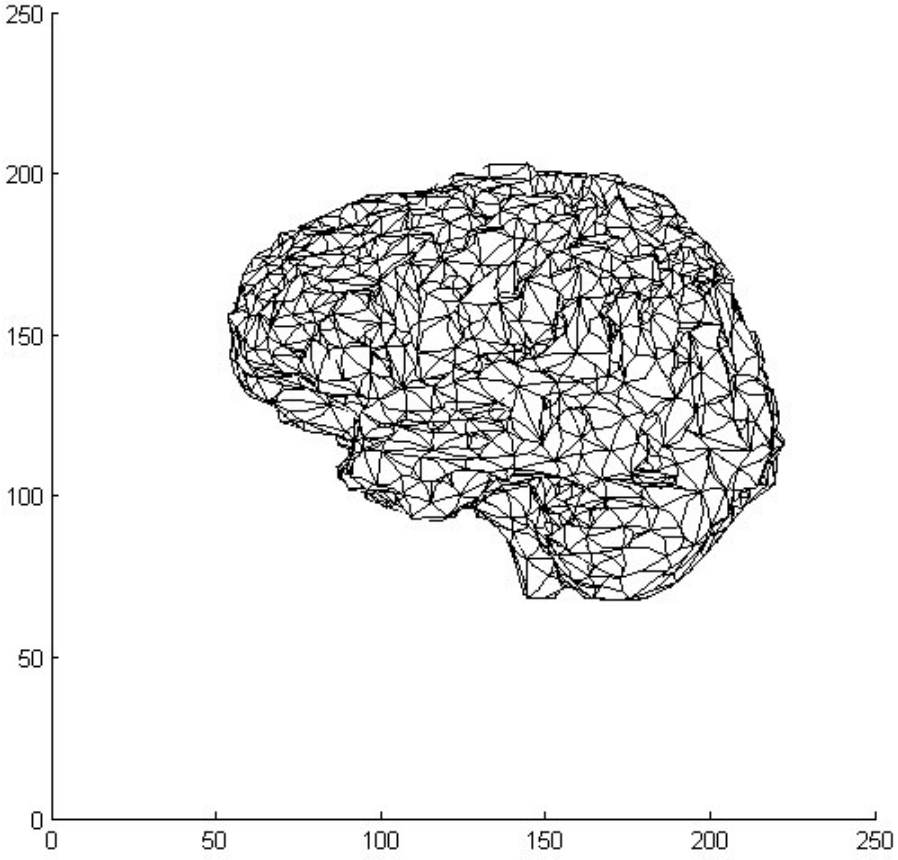


Figure 5.6: Brain of mesh 4 (units are in mm).

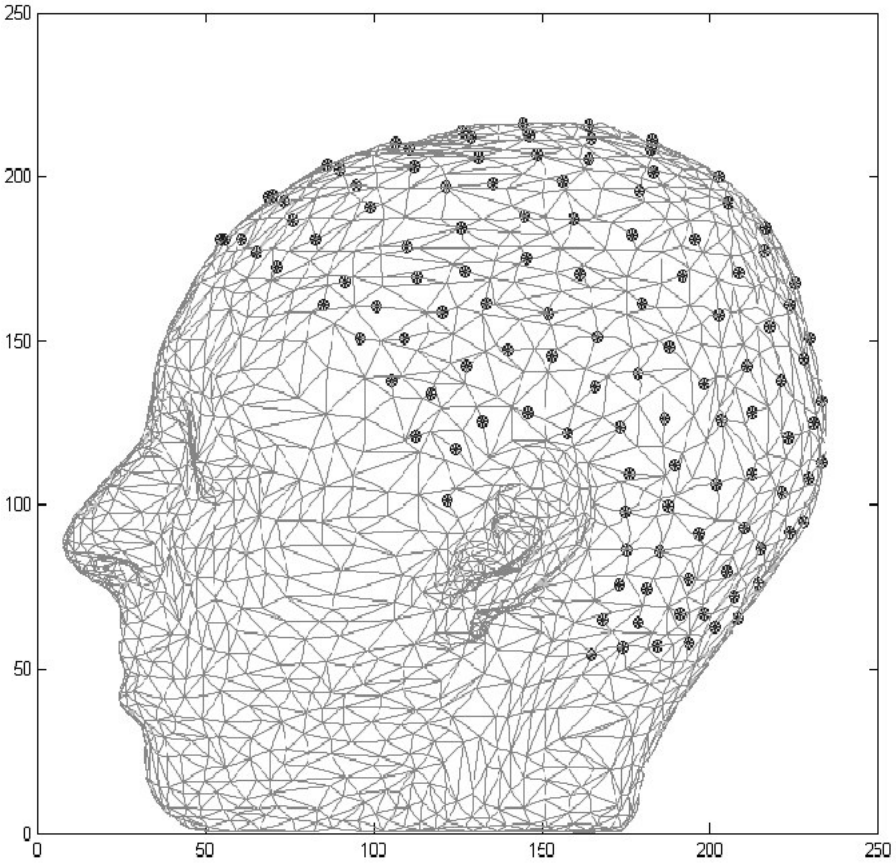


Figure 5.7: Electrode locations for an 256 electrodes system. These locations are used in both FP and IP solutions for mesh 3 and mesh 4.

## 5.2 Measures of Performance

For accuracy computations, error is calculated using RDM (4.1) and $RDM^*$ (4.2). MAG (4.3) is not preferred since it provides very limited information and RDM gives sufficient information for the main factors (dipole position and strength) affecting MAG.

For parallelization performance, two measures are used: speed-up and speed-up efficiency. Speed-up for $n$ processors is defined as:

$$speed - up = \frac{execution\ time\ for\ 1\ processor}{execution\ time\ for\ n\ processors} \tag{5.1}$$

Speed-up efficiency for n processors is defined as:

$$eff = \frac{speed - up}{n} \times 100\% \tag{5.2}$$

If single processor execution time is not available (which is the case for BEM on big meshes), (5.1) and (5.2) are obviously modified to include the ratio of the number of processors for compared cases.

The speed-up efficiency is a more meaningful measure than speed-up for parallelization performance. It provides information about the scalability of the parallelization. If efficiency is dropping fast with increased number of processors, the scalability is weak. Although it is called as efficiency, values over 100% (superlinear efficiency) are possible in some cases (mainly due to processor cache scaling). Thus, the speed-up efficiency is not a real efficiency measure. Still, it provides valuable information for the gain of parallelization.

## 5.3 Assessment of Accuracy

The adopted BEM implementation has already been tested in [25] and [12]. The parallel version of this approach is also tested with a spherical 3-shell Rush & Driscoll model [70]. In this model, the radii of the brain, skull and scalp surfaces are 8 cm, 8.5 cm and 9.2 cm and the corresponding conductivities are 0.2 S/m, 0.0025 S/m and 0.2 S/m, respectively. The accuracy for tangentially and radially oriented dipoles at varying depths are tested with the analytical solutions [26]. The results are in good agreement with the ones presented for a single processor [12].

To demonstrate the dependence of forward problem accuracy on the accuracy in the model, a simple experiment is performed with meshes 1 and

2. Even for a spherical mesh (which has a smooth geometry that can be modeled accurately by small number of quadratic elements) accuracy is dependent on the accuracy of the model, especially for tangential dipoles (figure 5.8). Although accuracy for deep radial dipoles may seem to be less affected by model accuracy, the improvement is vital, especially for shallow radial dipoles, as it is presented in the following sections.

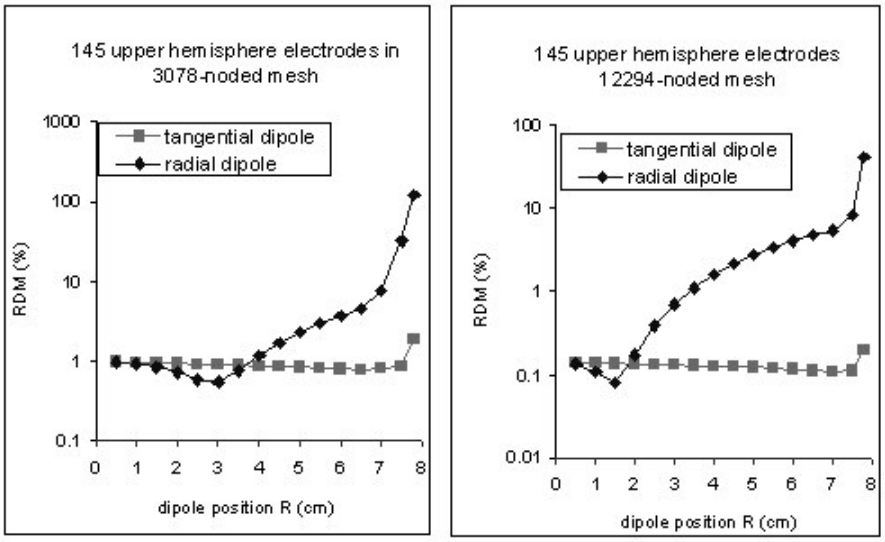


Figure 5.8: Dependence of error in forward problem solutions to accuracy in model (mesh). Electrodes are uniformly distributed and at the same places for both cases.

The accuracy for tangentially and radially oriented dipoles at varying depths (in upper hemisphere, on z-axis) and different electrode distributions over mesh 2 is presented in figure 5.9. Since accuracy is independent of the number of processors used (there is no precision loss in the network), the

results are taken with 8 processors only. RDM is used as the error measure. RDM* leads to very similar results with those obtained by RDM. Accuracy is poor for radially oriented shallow dipoles.

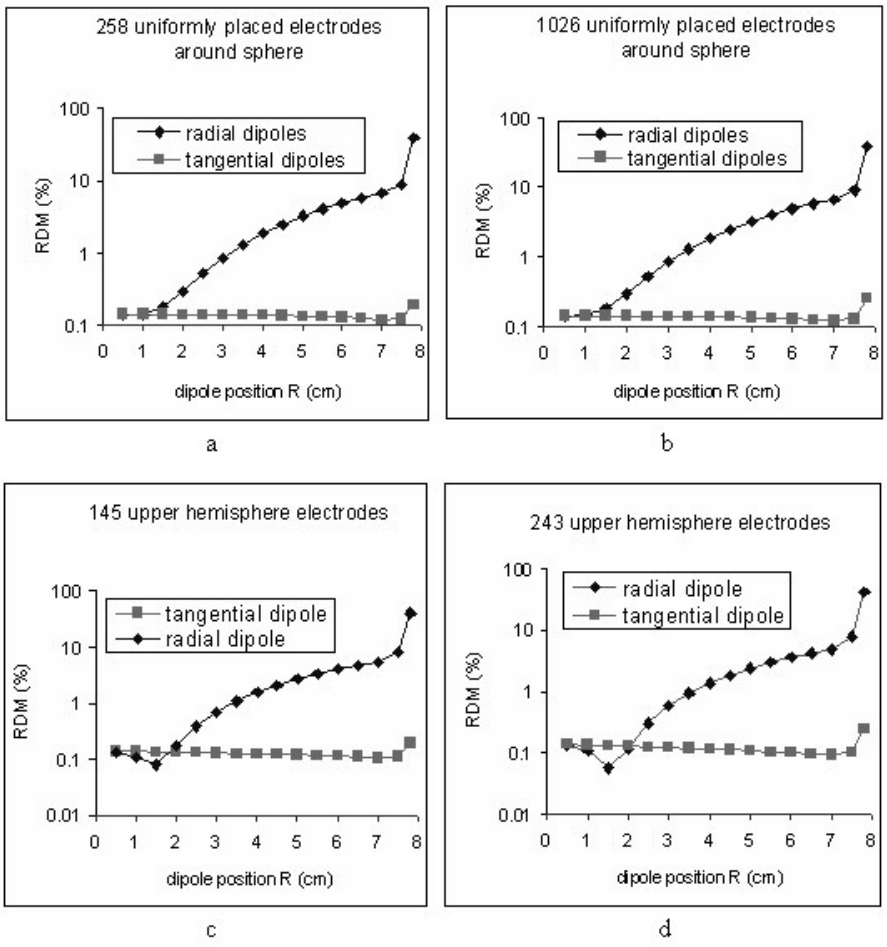


Figure 5.9: Forward problem accuracy with different electrode distributions.

In the tangential case (figure 5.10b), the dipole is perpendicular to $\vec{R}$ and the dot product vanishes for $N_1$ resulting in a zero field no matter how

close the node is. As dipole approaches $N_1$, although the dot product $\vec{p} \cdot \vec{R_2}$ increases, $\|\vec{R_2}\|$ does not tend to zero. Thus, the primary field ($\mathbf{g}$) remains smooth over the element. When the field variation is smooth near the dipole, second order elements with 13-point Gauss-Legendre quadrature can approximate the field with sufficient accuracy. The tangential dipole may be directed to some other node of the same element, but for most cases, R for this node is not intolerably small.

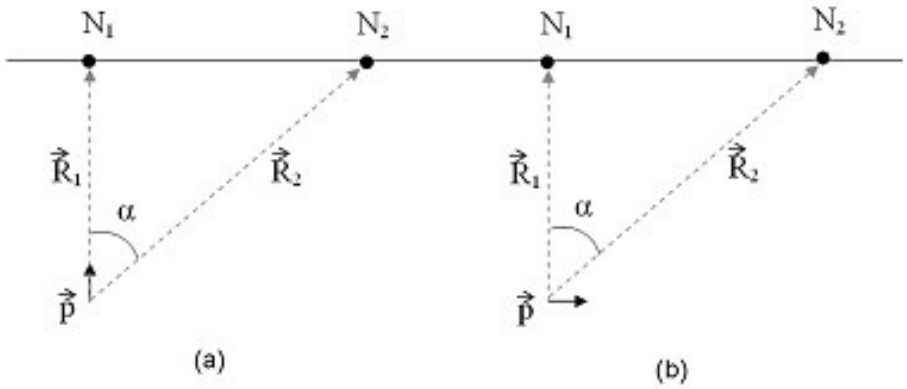


Figure 5.10: Dipole orientation: (a) Radial dipole just below a node, (b) Tangential dipole just below a node.

If the tangential dipole is very close to a node and directed towards another "far" node that does not share an element with the near node, the error will remain small since i) $R^3$ will be large, ii) the dot product and R for the nodes of the elements owning that "far" node will be similar, so the field variation will be smooth and second order elements with 13-point Gauss-Legendre quadrature will successfully approximate the field on the element owning that node.

On the other hand, for near tangential dipoles, since both nominator ($\vec{p} \cdot \vec{R}$) and denominator ($R^3$) are both small, accuracy is dependent on the precision. Furthermore, if the dipole is slightly moved in the tangential plane (i.e. the dipole direction changes towards the node), the angle between and $\vec{p}$ and $\vec{R_2}$, so their dot product will change rapidly. For such a case, error is expected to be larger.

Unlike the tangential orientation case, accuracy for radially directed dipoles is quite poor, especially, when the dipole is near (just below) the brain-skull interface. This is due to the fast varying behavior of the field near the dipole. Although the RDM is large for these cases, this error mainly comes from the electrodes that are near the dipole while the errors are moderate at other nodes. Since $\mathbf{A}$ is diagonally dominant, field value at a node is mainly determined by the dot product in (2.3) at that node. Thus, the field variation over an element is sharp if this dot product is changing fast among element nodes.

When the dipole is normal-directed and very near to the innermost surface, $\vec{p} \cdot \vec{R}$ varies very rapidly over the closest elements. This is because $\vec{R}$ and $\vec{p}$ are aligned for the closest node while they are almost perpendicular for "far" nodes of the element (figure 5.10a). As the dipole approaches $N_1$, the decrease in $\|\vec{R_1}\|$ causes a cubic increase in $\mathbf{g}$ at $N_1$. On the other hand, $\alpha$ approaches $\pi/2$, resulting in a zero dot product at $N_2$. Therefore, $R^3$ term in the denominator amplifies (2.3) significantly for near dipoles while this effect is not apparent for relatively far nodes of the element (the dot product is small and R is larger). Such a fast change may not be accurately approximated by Gauss-Legendre quadrature with 13 points on a quadratic element. The error is most effective for the elements in the innermost surface that are close to the dipole. The error in approximating the field distribution over these elements is reflected strongly to the nearest measurement nodes through the integrals in (2.3) and (2.4). The $R^3$ term in the integral suppresses this effect for further measurement nodes. Additionally, the field peaks at the nearest elements (and their nodes), so the RDM is dominated

by this approximation error.

## 5.4 Performances of the KSMs

In this section, the computation times for various KSMs are explored for different number of processors. For this purpose, first the BEM system matrix and the right-hand-side (required for calculation of 256 rows of **E**) are calculated for mesh 2. Then, the same matrix equation is solved by each KSM for different number of processors. Table 5.1 presents average solution times for various KSMs. The relative residual error tolerance (the ratio to terminate iterations: $rtol = \|\mathbf{r_i}\|/\|\mathbf{r_0}\|$, $r_i$ is the residual for $i^{th}$ iterate) is taken as $10^{-5}$.

Table 5.1: Solution times for various KSMs using different number of processors (for mesh 2)

| Procs | GMRES | Bi-CGSTAB | CGS | TFQMR | CR | CGNE |
|---|---|---|---|---|---|---|
| 1 | 93.16 | 112.29 | 124.36 | 114.62 | 151.16 | 1120.01 |
| 2 | 42.87 | 45.86 | 50.51 | 50.21 | 66.91 | 469.82 |
| 3 | 41.45 | 48.75 | 49.85 | 49.42 | 65.55 | 462.18 |
| 4 | 20.57 | 25.36 | 24.71 | 24.57 | 34.18 | 248.97 |
| 5 | 17.09 | 20.43 | 16.44 | 16.67 | 22.49 | 169.24 |
| 6 | 16.48 | 20.46 | 19.70 | 16.98 | 23.98 | 157.94 |
| 7 | 12.50 | 14.39 | 14.47 | 13.31 | 19.50 | 147.69 |
| 8 | 10.99 | 13.10 | 13.18 | 13.16 | 18.11 | 137.02 |

It is observed that, the GMRES method has the best performance among

the tested KSMs. This is an expected result since GMRES is generally the most robust method for dense matrices. It generates orthogonal basis vectors for the Krylov subspace with no need for calculating the actual residual and requires single matrix-vector multiplication per iteration. Preconditioning does not improve the convergence time (the BEM matrices are well-conditioned). These results are consistent with the ones reported in [36]. In the rest of this study, all results are obtained using the GMRES method with $rtol = 10^{-2}$ (rtol choice changes the number of iterations before convergence, not the speed-up).

In the computation process of $\mathbf{E}$, a non-zero initial guess for each row ($\mathbf{e_i}$) greatly improves the convergence. This is due to the diagonally dominant characteristic of $\mathbf{A}$ (and $\mathbf{A^{-1}}$). Selecting the initial guess as the right-hand side vector is observed to be useful.

## 5.5 Speed-up

The performance in the parallelization of various stages (filling the system matrix, construction of the transfer matrix $\mathbf{E}$, solution of the isolated problem, computation of the modified right hand side vector $\mathbf{g'}$ and obtaining the potential distribution by $\mathbf{Eg'}$) are presented in figures 5.11,5.12 and 5.13 for the spherical and two realistic head models.

Table 5.2: Efficiency of various phases of the BEM implementation

| Mesh | $\mathbf{A_s}$ fill | $\mathbf{A_s^{-1}}$ computation | $\mathbf{A}$ fill | $\mathbf{E}$ computation | Single FP solution |
|---|---|---|---|---|---|
| Mesh 2 | 65.8% | 131.5% | 66.4% | 127.9% | 117.4% |
| Mesh 3 | 65.6% | 96.6% | 95.3% | 97.5% | 94.2% |
| Mesh 4 | 67.7% | - | 91.2% | 85.8% | 98.7% |

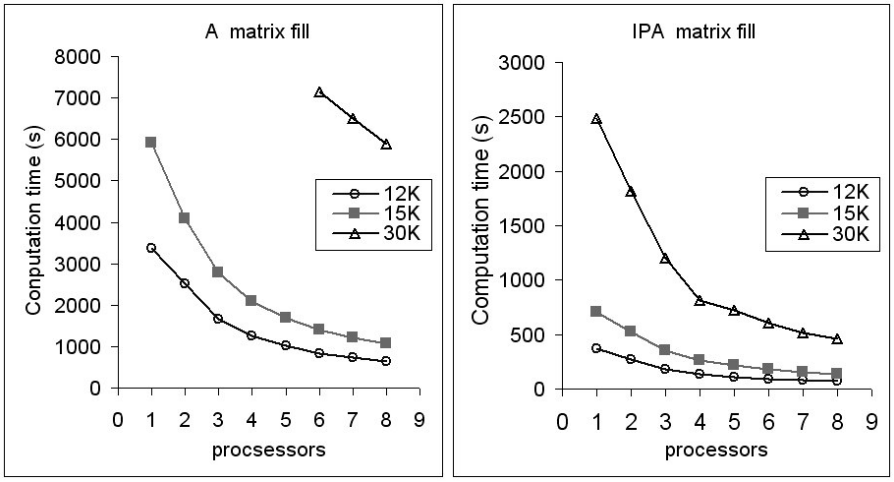


Figure 5.11: Matrix filling speed-up for $\mathbf{A_s}$ and $\mathbf{A}$.

When the FP solutions are obtained using large number of nodes with a single processor, computation time and memory requirements increase (even a 1.5 GB of RAM is not sufficient). For a mesh of 30 000 nodes, for example, the system matrix contains 900 million double precision entries which corresponds to a memory need of 7.2GB. Computer memory requirement of that amount can only be supported by five or more processors of the Athlins cluster, since each computer has only 1.5GB of RAM. Thus, to report on speed-up for incremental number of increase in the number of processors (2 to 8 for Athlins) first a three-layer concentric sphere model with a mesh of 12294 nodes is used.

The performance of parallelization is also tested for two realistic head models. The two meshes (mesh 1 and mesh 2) has 14999 and 7499 second order elements (with equal number of elements for every boundary) with 15011 (mesh3), 29799 (mesh 4) nodes, respectively. In both models, numbers of elements and nodes are very similar for each layer.

For the spherical (12294 noded) and a realistic (15011 noded) meshes,

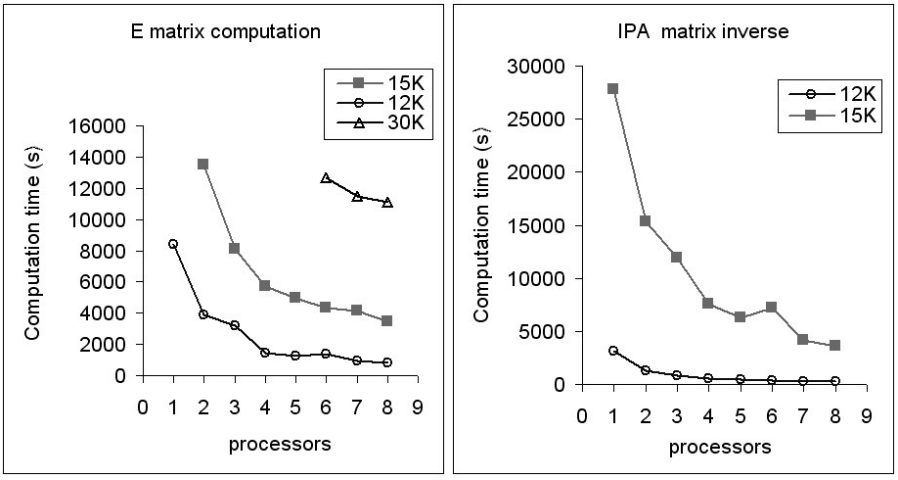


Figure 5.12: Speed-up for computation of $\mathbf{E}$ and $\mathbf{A_s^{-1}}$.

the speed-up assessment for filling $\mathbf{A}$ and $\mathbf{E}$ matrices is made by taking the two processors experiment as the basis instead of the one with single processor experiment, since the swap space usage in single processor case causes slow operation. Thus, the calculated speed-up will not be realistic if single processor case is taken into account.

The speed-up and efficiency varies for different phases of the FP solution process. As expected, efficiency drops for increased number of processors. The efficiency values for the spherical and realistic models are presented in table 5.2.

The $\mathbf{A_s}$ matrix filling efficiencies get their values at the transition from single processor to two processors case (the library switches to parallel mode) and remain very close to these values for increased number of processors.

The time required to obtain $\mathbf{A}_s^{-1}$ is quite long with KSMs, especially for realistic meshes. Inversion with direct methods or computing the IPA solution with an iterative solver would be more reasonable.

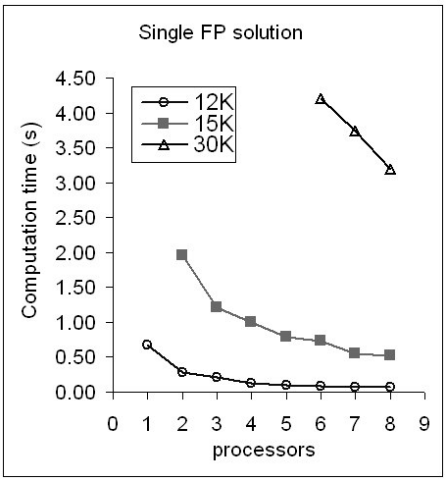


Figure 5.13: Speed-up for single FP calculation.

## 5.6 The Inverse Problem

Localization accuracy for inverse problem is tested with mesh 2. Meshes 3 and 4 are used to test processing times, since no analytical solution is at hand for realistic meshes.

### 5.6.1 Tests on The Genetic Algorithm

The developed genetic algorithm is tested for 3-shell Rush&Driscoll model using the analyitcal solutions for the electric potentials and the magnetic field. The conductivity values are taken as 0.2 S/m, 0.0025 S/m 0.2 S/m for scalp, skull and brain layers, respectively. A grid of 213 points is used for the tests. At each test point, 8 different dipole orientations are used. The dipoles are taken in the x-z plane for the electric potentials and in the tangential plane for the magnetic field. The algorithm is run for 128 and 256 upper hemisphere electrodes and SNR values of 15dB and 20dB as well as the noise-free case.The total number of runs is 20448.

The noise-free simulations provide perfect localization errors (below 0.1mm) for all locations. For noisy cases, the localization accuracy is improved with increased number of electrodes/sensors and higher SNR for both electric potential and magnetic field experiments. Localization using the magnetic field measurements is observed to be very sensitive to the noise. The genetic algorithm necessitates many generations (≈30000) for meaningful results with the MEG simulations while 2000 generations suffice for the EEG. With the inverse problem algorithm that is used in this study, the electric potential experiments provide more robust solutions for localization.

Table 5.3: Mean localization error MLE (mm) and mean orientation error MOE (degrees) for EEG experiments

| Number of electrodes | MLE(15dB) | MOE (15dB) | MLE(20dB) | MOE (20dB) |
|---|---|---|---|---|
| 128 | 4.026 | 2.090 | 2.239 | 1.170 |
| 256 | 2.937 | 0.995 | 1.668 | 0.559 |

Table 5.4: Mean localization error MLE (mm) and mean orientation error MOE (degrees) for MEG experiments

| Number of sensors | MLE(15dB) | MOE (15dB) | MLE(20dB) | MOE (20dB) |
|---|---|---|---|---|
| 128 | 2.089 | 3.566 | 1.456 | 2.631 |
| 256 | 0.972 | 2.199 | 0.614 | 1.276 |

### 5.6.2 Localization Accuracy

Figure 5.14 shows the minimum forward problem error points for radially and tangentially oriented shallows dipoles on the z-axis. The experiment for generating these plots is performed as follows: For 4 different locations on the z-axis, analytical solutions for radial/tangential dipoles are taken as the reference and the error between these analytical solutions and forward problem solutions for correctly oriented near dipoles are calculated. The motivation is to obtain primitive information for the relationship between the inverse problem localization errors and the dipole orientation.

There is a significant difference in error pattern behavior for radial and tangential dipoles. As apparent from the graphs, for shallow radial dipoles, the inverse problem may converge to some false locations since better accuracy can be obtained there.

Note that, this is a 1-D experiment (dipoles are forced to be on z-axis and correctly oriented), hence minimum error may be obtained for some other point in 3-D space and different dipole orientation. Still, the global minimum can be expected to be only slightly different (near and with similar orientation). The error pattern gets smoother for deeper dipoles, so it is not likely that inverse problem will converge to the exact location.

As a matter of fact, these expectations are verified by corresponding inverse problem solutions. The obtained inverse problem localization errors after 2000 generations for radial dipoles are: 2.92mm, 1.66mm, 1.45mm, 1.57mm for R=7.8cm, 7.5cm, 7.2cm and 6.9cm, respectively. For tangential dipoles, obtained values are perfect: 0.1mm, 0.01mm, 0.06mm, and 0.19mm, respectively.

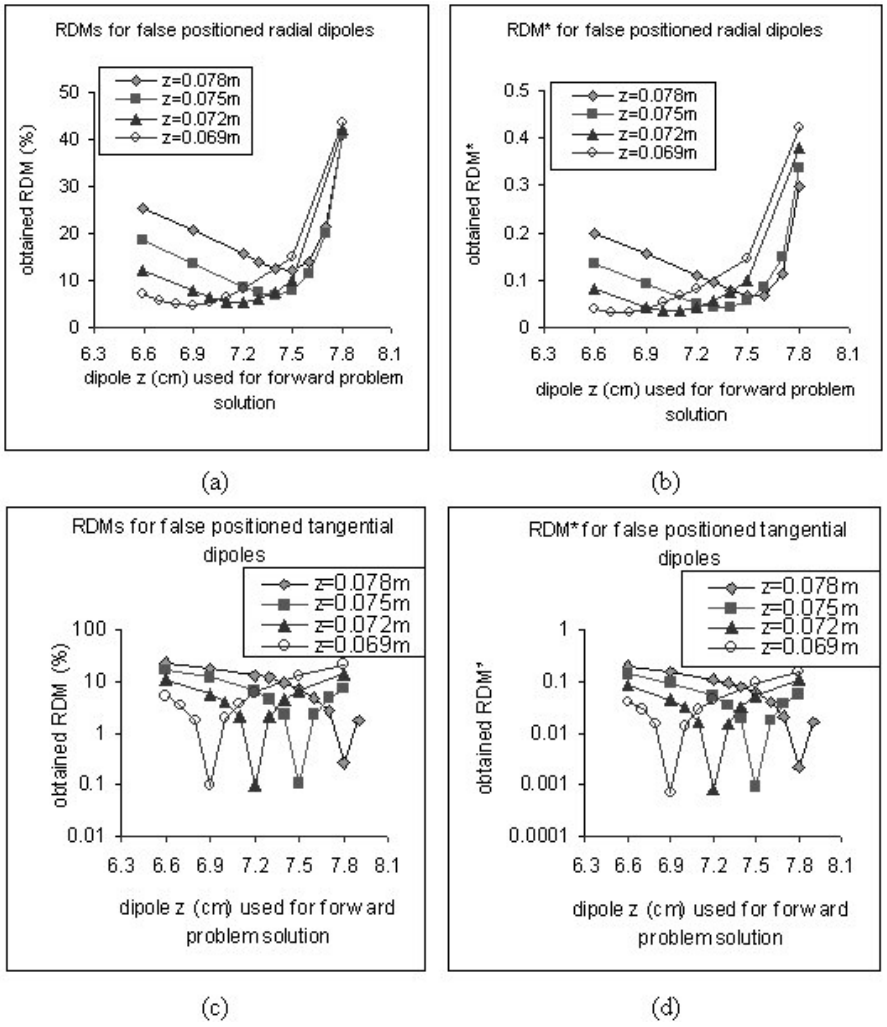


Figure 5.14: Error plots for false dipole positions and true dipole amplitudes.

Note that, this simple experiment is performed to connect forward problem accuracy considerations to inverse problem. Since electrode distribution is not uniform over the surface, localization errors will not be the same for dipoles at different positions (even if these positions have the same distance from the innermost surface), as seen in figures 5.7, 5.15 and 5.16.

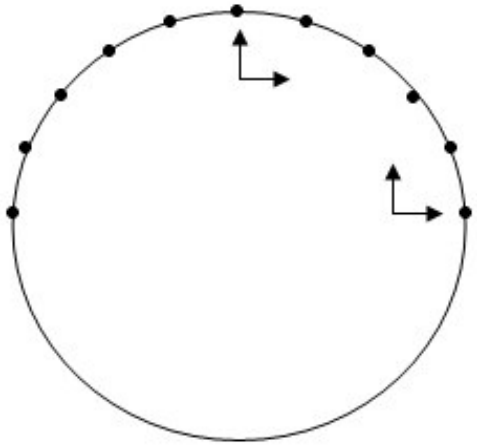

Figure 5.15: Two tangential and two radial dipoles located near the innermost sphere with uniform upper-hemisphere electrode distribution.

The main drawback of the significant difference between the patterns for radial and tangential dipoles (figure 5.14) is that the stopping criteria for inverse problem become complicated. To guarantee correct localization, the algorithm must be run with highest possible satisfactory fitness and maximum possible number of generations.

It must also be noted that for real-world applications, detectors are not uniformly distributed around head surface (figure 6.16). Rather, they are placed around the upper part of the head. Hence, dipole localization errors will be different for sources at different regions of the head.

The localization accuracy of genetic algorithm on the numerical model is tested using mesh 2 for the noise-free case. The electrode positions are arranged such that, no electrode is placed on the facial region (figure 5.16).

When the numerical method is used for the solutions, fitness function calculations require expensive matrix-vector multiplications. Thus, the results

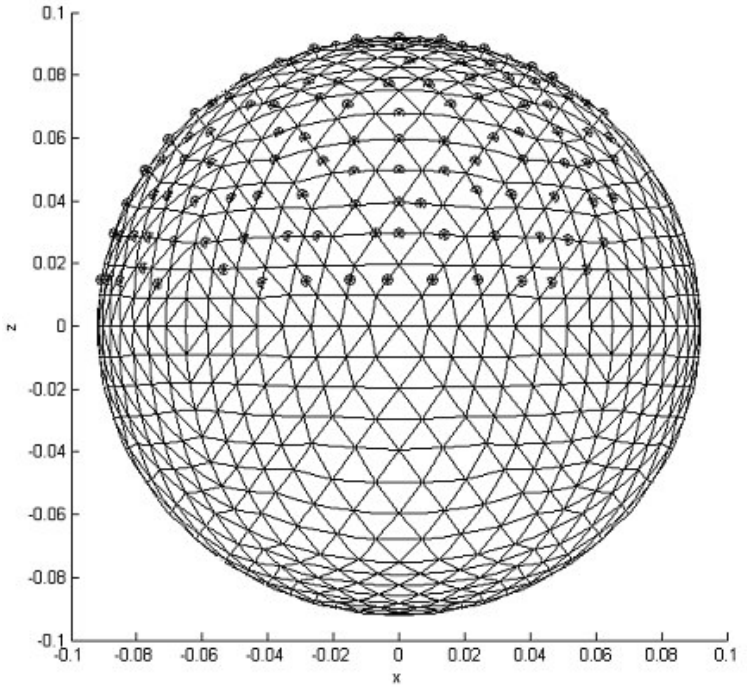


Figure 5.16: Electrode positions for mesh 2 and 243 electrodes. Units are in meters. No electrodes are placed in front since there are no electrodes on the face.

for the numerical method are taken on a more sparse grid (35 points) in the x-z plane with three major orientations (for x,y and z-directed dipoles). The results are presented in the tables 5.5, 5.6, 5.7:

The localization accuracy is worst for the x-directed dipoles that are on the x axis and near the innermost surface. In this region no electrodes are present and the dipole is radial. Since there no electrode is present at the points where the potential is strongest, the genetic algorithm attempts to fit the far electrodes' potentials, which are weak. The approximation errors on the nearest elements are felt less in the fitness. Consequently, the fitness is

weakly related to the correct localization and the error resulting approximation error is larger than other cases. When x-directed dipole is placed near the z-axis, it has a strong tangential component which forces the algorithm to converge to the correct location even for the shallow case.

The y-directed dipoles are always tangential since the dipole is placed on the x-z plane. The potential approximation error is small even for the nearest elements when the dipole is near the innermost boundary. Thus, the lack of electrodes near the x-axis does not affect the localization error significantly.

Table 5.5: Localization accuracy for x-directed dipoles. All units are in mm

| z | x | | | | | | |
|---|---|---|---|---|---|---|---|
| | 11 | 22 | 33 | 44 | 55 | 66 | 77 |
| 77 | 0.395 | - | - | - | - | - | - |
| 66 | 0.241 | 0.063 | 0.248 | 0.515 | - | - | - |
| 55 | 0.054 | 0.179 | 0.558 | 1.253 | 1.064 | - | - |
| 44 | 0.119 | 0.356 | 0.920 | 1.759 | 2.572 | 1.649 | - |
| 33 | 0.129 | 0.400 | 0.920 | 1.948 | 4.220 | 4.884 | - |
| 22 | 0.184 | 0.387 | 0.669 | 1.496 | 3.315 | 4.553 | - |
| 11 | 0.083 | 0.347 | 0.539 | 3.035 | 2.349 | 3.85 | 6.226 |

The z-directed dipoles behave differently. They have strong radial components when they are near the z-axis. Thus, at these positions, the localization error for z-directed dipole is higher than both x-directed and y-directed dipoles. However, unlike radial x-directed dipoles case, the electrodes are dense near the z-axis. Therefore, the localization error never reaches to that of an x-directed radial dipole. When the z-directed dipole is near the x-axis, it is has a strong tangential component. However, its radial component

Table 5.6: Localization accuracy for y-directed dipoles. All units are in mm

| z | x | | | | | | |
|---|---|---|---|---|---|---|---|
| | 11 | 22 | 33 | 44 | 55 | 66 | 77 |
| 77 | 0.138 | - | - | - | - | - | - |
| 66 | 0.061 | 0.060 | 0.304 | 0.103 | - | - | - |
| 55 | 0.092 | 0.078 | 0.021 | 0.183 | 0.136 | - | - |
| 44 | 0.103 | 0.069 | 0.056 | 0.064 | 0.105 | 0.424 | - |
| 33 | 0.060 | 0.024 | 0.139 | 0.080 | 0.049 | 0.181 | - |
| 22 | 0.053 | 0.059 | 0.019 | 0.065 | 0.166 | 0.117 | - |
| 11 | 0.041 | 0.070 | 0.043 | 0.258 | 0.208 | 1.160 | 0.749 |

causes a localization error, which is more than y-directed (fully tangential) dipole case.

Localization accuracy for realistic meshes is not reported here due to the unavailable analytical solutions. Still, inverse problem can be solved on these meshes using numerical solutions of the forward problem (for the actual dipole position and orientation) as the measurement data. Obviously, for noise–free experiments, the algorithm is expected to converge to the actual position. These experiments are performed to report solution times and speed-up for the inverse problem. There are many possible experiments that can be performed for the inverse problem. Again, it is clear that, no new information other than figure 5.13 can be obtained for speed-up through these experiments.

Table 5.7: Localization accuracy for z-directed dipoles. All units are in mm

| z | x | | | | | | |
|---|---|---|---|---|---|---|---|
| | 11 | 22 | 33 | 44 | 55 | 66 | 77 |
| 77 | 2.496 | - | - | - | - | - | - |
| 66 | 1.517 | 1.201 | 1.070 | 1.272 | - | - | - |
| 55 | 1.347 | 1.147 | 0.790 | 0.140 | 0.838 | - | - |
| 44 | 0.870 | 0.669 | 0.246 | 0.528 | 1.738 | 4.476 | - |
| 33 | 0.350 | 0.147 | 0.293 | 1.047 | 2.335 | 4.602 | - |
| 22 | 0.270 | 0.201 | 0.471 | 1.067 | 2.087 | 3.971 | - |
| 11 | 0.246 | 0.150 | 0.319 | 0.778 | 1.419 | 2.652 | 3.571 |

### 5.6.3 Solution Times

For testing the speed-up in the inverse problem, it is not necessary to perform the same experiments on different number of processors, since the number of generations necessary to converge to the global minimum is not dependent to the number of processors. Hence, the comparison for the average time required to complete the computations for a single generation on different number of processors is sufficient.

In fact, the single solution speed-up (figure 5.13) gives the necessary information for the speed-up without performing any inverse problem experiments. Nevertheless, we have verified its results with single generation computation times in the inverse problem solution and connected it to the total time required to localize a single dipole. The processing time for mesh 2 and 2000 generations takes about 48 minutes on 8 processors.

For mesh 3 and mesh 4, the algorithm is tested for radial and tangential dipoles for 10 different positions (figure 5.18). Same locations are used for mesh 4. For the forward problem solutions in the fitness function calculation, IPA equations are solved iteratively. Experiments show that, for mesh 3,

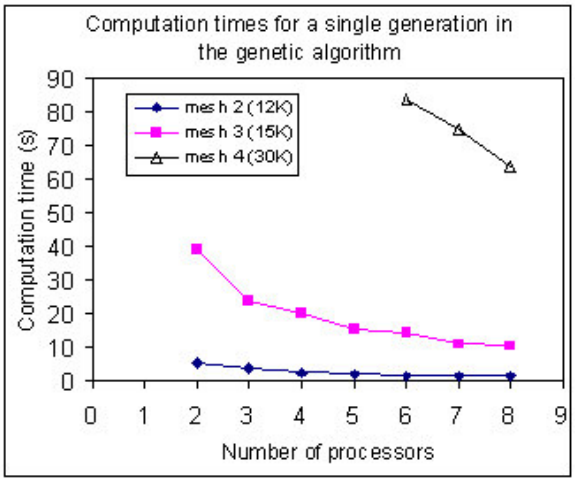


Figure 5.17: Average processing time for one generation.

2000 generations (corresponding to 346.7 minutes of processing time on 8 processors) suffices for tested dipole locations. This number is observed to be the same (corresponding to 2130 minutes of processing time) for mesh 4. Again, for less number of processors, solution times are in agreement with figure 5.13. Thus, the speed-up efficiency values for the FP in figure 5.13 are also valid for the IP.

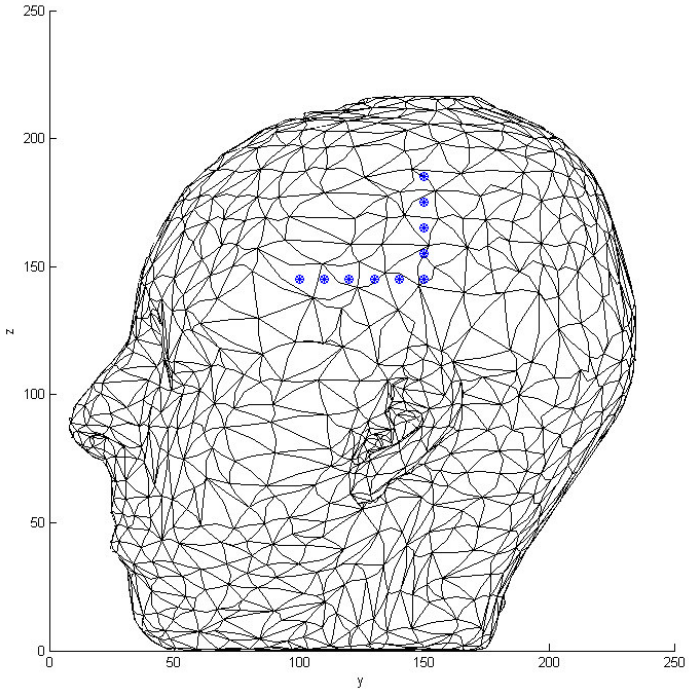


Figure 5.18: Dipole locations used for IP on realistic models. Units are in mm.

# CHAPTER 6

# CONCLUSIONS

This study introduces a parallel processing approach to speed up the the forward problem solutions and to enable the usage of dense meshes for EMSI of the human brain. The accelerated BEM proposed for faster FP solutions [12] was implemented using a parallel PC platform. A Beowulf cluster was developed with 8 processors. The PETSC library allowed the use of KSMs in the solution of resultant dense matrix equations. The solution times were compared for various KSMs. The shortest solution time was obtained using the GMRES algorithm. It was observed that parallelization provides faster operation with a considerable speed-up in the matrix fill, transfer matrix calculation and obtaining solutions for a specific source configuration. Thus, it also enables faster inverse problem solutions.

## 6.1 The Forward Problem

The first income of parallelization is the memory scaling. For all of the three meshes used in this study, swap space usage is unavoidable on a single processor of Marvin even under Linux operating system that consumes very small portion of RAM. Parallelization immediately solves this problem. However, although parallelization provides the opportunity to work on dense meshes, the number of nodes in the model could be increased only by the square root of the cluster size (number of PCs in the cluster). For the 30K mesh, it is observed that, for Marvin cluster (1.5GB RAM/processor) the number of processors must be more than 5 for proper operation.

The second advantage of parallelization is the speed up it provides. The

use of a parallel PC platform is vital for the calculation of the system matrix **A**, and the transfer matrix **E**. Our experiments show that, parallelization provides a speed-up efficiency of 66.4% for 12K mesh, 95.3% for 15K mesh and 85.8% for 30K mesh, for filling **A**. The efficiency values for computation of **E** are 127.9%, 97.5% and 85.8%, respectively.

Once **E** is calculated, the time gained for a single forward problem solution is not important as the forward problem solution can be obtained in short time using even a single processor. However, to obtain **E**, the available memory of the single processor should be sufficient for storing **A** and this is not possible if the number of nodes in the mesh is large.

The inversion of the IPA matrix with iterative solvers is expensive, especially for dense meshes. Thus, we preferred to perform IPA solutions iteratively instead of taking the inverse of $\mathbf{A_s}$. For the forward problem solutions, this choice increases the single solution time (from 32ms to 84ms for the mesh 2 and from 41ms to 520 ms for the mesh 3). The inversion of $\mathbf{A_s}$ may take hours depending on the matrix. Thus, the choice for solving the IPA equation iteratively is cheaper than taking the inverse of $\mathbf{A_s}$ by iterative methods if the number of right-hand-sides is less than a few thousands. If available, use of direct inversion and LU factorization are definitely better choices for IPA solutions.

The speed-up for the FP solution for a single dipole (with $\mathbf{Eg'}$) is directly reflected to the inverse problem solution, in which many FP solutions are performed. Thus, the speed-up in a single FP solution is also critical. For single FP solution, obtained speed-up efficiencies are: 117.4%, 94.2% and 98.7% on mesh 2, mesh 3 and mesh 4 respectively.

## 6.2 The Inverse Problem

In this study, a simple genetic algorithm is implemented in parallel. The parellel part of the algorithm is in fitness function evaluations, which includes FP solutions and is the main computational load. If the IPA matrix is

small enough to be stored on a single processor's RAM, the parallelization of the genetic algorithm can be implemented using the traditional genetic algorithm parallelization (dividing the population into processors), in which each chromosome is handled by a single processor.

The algorithm is tested on a grid of 213 points in the first quadrant of x–z plane for the spherical model, with analytical solutions for two different spatial SNR as well as the noise-free case. The noise-free case provides very accurate localization results for all locations (below 0.1mm error). The magnetic field calculatione necessitates significantly more generations for convergence compared to the electric potential calculations.

For noisy cases, it is observed that an increase in SNR or number of sensors/electrodes enhances both orientation and localization accuracy. With improved SNR and increased number of electrodes/sensors, the localization error decreases from 4.026mm (15dB, 128 electrodes) to 1.668mm (20dB, 256 electrodes) for EEG and from 2.089mm (15dB, 128 sensors) to 0.614mm (20dB, 256 sensors) for MEG. Similarly, the orientation errors decrease from 2.090 (15dB, 128 electrodes) degrees to 0.559 (20dB, 256 electrodes) degrees for EEG and 3.566 degrees (15dB, 128 sensors) to 1.276 degrees (20dB, 256 sensors) for MEG.

Although this study reports the speed-up for a simple genetic algoritm, in the literature, the EMSI IP solutions with genetic algorithms are generally performed as hybrid methods, in which a deterministic (gradient-based) method is coupled to the genetic algorithm. This approach compensates the slow convergence of the genetic algorithm near the global optimum. Thus, the computation time can be further reduced by the utilization of hybrid methods.

The numerical experiments show that the localization error is significant (up to 6.226mm) for the cases where the dipole is radial and shallow, and there are no near electrodes (e.g. facial region). Localization error is also larger for radially oriented shallow dipoles than tangentially oriented shallow dipoles. As explained in chapter 5, this is due to the topological differences

in the field for the two cases as well as the nature of the numerical approximations.

## 6.3 Factors Affecting The Computation Time

The first and obvious factor that affects the computation time is the number of nodes and elements used in the head model. A better approximation (i.e., increased number of nodes and elements) neccessitates larger matrices. As the matrix size increases, both the number of floating point operations per matrix–vector multiplication and the condition number of the system matrix increases. Consequently, the convergence of the iterative solver becomes slower.

Another critical point for the usage of KSMs is rtol. Computation times can be reduced by changing the value for rtol. If a good initial estimate of the solution is available, the residual error takes a small value at the first iteration (e.g. $10^2$ for a 30000×30000 system). We observed that rtol values up to $10^{-2}$ provide sufficient accuracy in FP solutions for many cases. The iterates saturate after $10^{-2}$ is satisfied for rtol and hence, further iterations become inefficient. Furthermore, especially for the realistic models, depending on the characteristics of the resultant matrix, choosing small rtol (e.g. $10^{-5}$) may prevent the KSM convergence. The answer for the question "How accurate must the transfer matrix be?" is also a determining factor for the choice of rtol, since the accuracy is already limited by the geometric approximation. Although rtol changes the computation time, it does not effect the speed-up since it determines the number of KSM iterations before convergence and the number of iterations is not related to the the number of processors. The effect of rtol needs further investigation, especially for its effect in the inverse problem accuracy.

In this study, second order elements are used in the models. Second order elements require second order interpolation functions, which becomes the main computational load in matrix filling. However, if the number of

elements in the mesh is large, the benefits of using second order elements can also be questioned. If linear elements are used, the number of the elements in the head mesh can be increased (by a factor of four) using the same number of nodes with the quadratic model. Although this will increase the number of element visits in matrix filling, each visit to an element will be cheaper. Note that, the speed–up is not expected to be affected by the element type. The effect of element order (for accuracy and processing time) should be investigated in future studies.

The proposed parallelization scheme is independent of the number of processors, i.e, faster operation is possible if more processors are used. In fact, BEM implementation is not sufficiently fast for accurate models even with 8 processors. Theoretically, the speed-up is expected to saturate after a specific number of processors. However, in Beowulf practice, the number of processors is always much less than the number of nodes and elements. Consequently, over a fast network, increased number of processors will provide further improvement in speed-up, with small loss in speed-up efficiency.

## 6.4 Future Studies

There are excessively many topics for future research on the subject. These include:

- Extension of the computational power of The Marvin Cluster by adding new processing units to fasten operations.
- Usage of hybrid methods that use both stationary/direct methods and KSMs for faster solutions of the BEM equations.
- Investigation of the effects of element order and type when the number of elements is high.
- Case studies for the inverse problem (algorithms, source models, noise levels, electrode/sensor distribution, etc).

# REFERENCES


[1] S.Baillet, J.C.Mosher, M.R.Leahy, Electromagnetic brain mapping, *IEEE Signal Processing Magazine*, 18, no. 6, pp.14–30, 2001.

[2] N.G.Gençer, C.E.Acar, I.O.Tanzer, Forward problem solution of magnetic source imaging *Magnetic Source Imaging of the Human Brain* ed. Z L Lu and L Kaufman (Hillsdale, NJ: Lawrence Erlbaum Associates), 2003.

[3] B.He, High-resolution source imaging of brain electrical activity, *IEEE Eng. in Med. and Biol.* 17, no. 5 pp.123-129, 1998.

[4] B.N.Cuffin, EEG dipole source localization, *IEEE Eng. in Med. and Biol.*, September/October 1998, pp.118–122, 1998.

[5] J. C. DeMunck, B. W. van Dijk, H. Spekreijse, Mathematical dipoles are adequate to describe realistic generators of human brain activity, *IEEE Trans. Biomed. Eng.*, 35, p.960., 1988

[6] A.Crouzeix, Yvert B, Bertrand O, and Pernier J., An evaluation of dipole reconstruction accuracy with spherical and realistic head models in MEG, *Clinical Neurophysiology*, 110, pp.2176–2188, 1999.

[7] B.NbCuffin, EEG Localization accuracy improvements using realistically shaped head models, *IEEE Trans. Biomed. Eng.*, 44, pp.299–303, 1996.

[8] J.W.H.Meijs, O.Weier, and M.J.Peters, On the numerical accuracy of the Boundary Element Method *IEEE Trans. Biomed. Eng.*, 36, pp.1038-1049, 1989.

[9] M.S.Hämäläinen, J.Sarvas, Realistic conductivity geometry model of the human head for interpretation of neuromagnetic data, *IEEE Trans. on Biomed. Eng.* 36, no. 2, pp.165–171, 1989.

[10] M.Fuchs, R.Drenckhahn, H.A.Wischmann, M.Wagner, An Improved Boundary Element Method for Realistic Volume-Conductor Modeling, *IEEE Trans. Biomed. Eng.*, 45, no. 8, pp.980–997, 1998.

[11] J.H.M.Frijns, S.L.de Snoo, R.Schoonhoven, Improving the accuracy of the Boundary Element Method by the use of second-order interpolation functions *IEEE Trans. on Biomed. Eng.*, 47, no. 10, pp.1336-1346, 2000.

[12] Z.Akalin-Acar, N.G.Gençer, An advanced boundary element method implementation for the forward problem of electromagnetic source imaging, *Phys. Med. Biol.*, 49, no:21, pp. 5011–5028, 2004.

[13] N.G.Gençer, Z.Akalın-Acar, Use of the isolated problem approach for multi-compartment BEM models of electro-magnetic source imaging, *Phys. Med. Biol.*, 50, pp.3007-3022, 2005.

[14] C.E.Acar, Parallelization of the Forward and Inverse Problems of Electromagnetic Source Imaging of the Human Brain, Ph.D. Thesis, Middle East Technical University, Ankara, Turkey, 2003

[15] N.G.Gençer, C.E.Acar, Sensitivity of EEG and MEG measurements to tissue conductivity, *Phys. Med. Biol.*, 49, pp.701-717, 2004.

[16] C.H.Wolters, L.Grasedyck, W.Hackbusch, Efficient computation of lead field bases and influence matrix for the FEM-based EEG and MEG inverse problem, Inverse Problems, 20, pp.1099-1116, 2004.

[17] R.Natarajan, D.Krishnaswamy, A Case Study in Parallel Scientific Computing: The Boundary Element Method on a Distributed-Memory Multicomputer, *Proceedings, ACM/IEEE Conference on Supercomputing*, 1995.

[18] S.W.Song, R.E.Baddour, Parallel processing for boundary element computations on distributed systems, *Engineering Analysis with Boundary Elements*, 19, pp. 73–84, 1997.

[19] B.A.Baltz, M.S.Ingber, A parallel implementation of the boundary element method for heat conduction analysis in heterogeneous media, *Engineering Analysis with Boundary Elements*, 19, pp. 3–11, 1997.

[20] R.Barrett, M.Berry, T.F.Chan, J.Demmel, J.Donato, J.Dongarra, V.Eijkhout, R.Pozo, C.Romine, H.Van der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Ed., *SIAM*, Philadelphia, PA, 1994. http://www.netlib.org/linalg/html_templates/report.html, Last accessed date: September 2005.

[21] M.Kreienmeyer,E.Stein, Efficient parallel solvers for boundary element equations using data decomposition, *Engineering Analysis with Boundary Elements*, 19, pp. 33–39, 1997.

[22] S.Balay, V.Eijkhout, W.D.Gropp, L.C.McInnes, B.F.Smith, Efficient Management of Parallelism in Object Oriented Numerical Software Libraries,*Modern Software Tools in Scientific Computing*, Birkhäuser Press, pp. 163–202, 1997.

[23] W.Gropp, E.Lusk, N.Doss, A.Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, *Parallel Computing*, 22, no. 6, pp.789–828, 1996.

[24] Beowulf.org: The Beowulf Cluster Site, http://www.beowulf.org/, Last accessed date: September 2005.

[25] N.G.Gencer, I.O.Tanzer, Forward problem solution of electromagnetic source imaging using a new BEM formulation with high-order elements, *Phys. Med. Biol.*, 44, pp.2275–2287, 1999.

[26] C.J.Stok, The inverse problem in EEG and MEG with application to visual evoked responses, ISBN 90-70647-06-0, pp.101-117, 1986.

[27] S.Finke, R.M.Gulrajani, J.Gotman, Conventional and reciprocal approaches to the inverse dipole localization problem of electroencephalography, *IEEE Trans. on Biomed. Eng.*, 40, no. 6, pp. 657–666, 2003.

[28] B.N.Cuffin, Eccentric spheres models of the head, *IEEE Trans. Biomed. Eng.*, 38, no. 9, pp.871–878, 1991.

[29] J.J.Ermer, J.C.Mosher, S.Baillet, R.M.Leahy, Rapidly recomputable EEG forward models for realistic head shapes,*Phys. Med. Biol.*, 46, pp.1265–1281, 2001.

[30] G.Nolte, T.Fieseler, G.Curio, Perturbation theory as a new analytical approach to the MEG forward problem for realistic volume conductor modeling of the head,*Proceedings, Biomag2000*, 2000.

[31] Z.Akalin, C.E.Acar, N.G.Gençer, Development of realistic head models for electromagnetic source imaging of the human brain, *Proceedings of 23rd EMBS*, pp. 899–902, 2001.

[32] S.He, X.Shen, Y.Yang, R.He, W.Yan, Research on MRI brain segmentation algorithm with the application in model-based EEG/MEG, *IEEE Trans. On Magnetics*, 37, no. 5, pp.3741–3744, 2001.

[33] R.Valdés-Cristerna, V.Medina-Bañuelos, O.Yáñez-Suárez, Coupling of radial-basis network and active contour model for multispectral brain MRI segmentation, *IEEE Trans. on Biomed. Eng.*, 51, no. 3, pp.459–470, 2004.

[34] A.S.Heidi, L.Heller, R.Aaron, E.Best, D.M.Ranken, Evaluation of Boundary Element Methods for the EEG Forward Problem: Effect of Linear Interpolation, *IEEE Trans. Biomed. Eng.*, 42, no. 1, pp.52–58, 1995.

[35] M.Clerc, A.Dervieux, R.Keriven, O.Faugeras, J.Kybic, T.Papadopoulo, Comparison of BEM and FEM methods for the E/MEG problem, *Proceedings of Biomag 2002*, Jena, Germany, 2002.

[36] J.Rahola, and S.Tissari, Iterative solution of dense linear systems arising from the electrostatic integral equation in MEG, *Phys. Med. Biol.* 47, pp. 961-975, 2002.

[37] P.Schimpf, J.Haueisen, C.Ramon, H.Nowak, Realistic computer modeling of electric and magnetic fields of human head and torso, *Parallel Computing*, 24, pp.1433–1460, 1998.

[38] PVM: Parallel Virtual Machine, http://www.csm.ornl.gov/pvm/pvm_home.html, Last accessed date: September 2005.

[39] L.S.Blackford, J.Choi, A.Cleary, E.D'Azevedo, J.Demmel, I.Dhillon, J.Dongarra, S.Hammarling, G.Henry, A.Petitet, K.Stanley, D.Walker, R.C.Whaley, ScaLAPACK Users Guide, Society for Industrial and Applied Mathematics, Philadelphia, PA, ISBN:0-89871-397-8

[40] PLAPACK, www.cs.utexas.edu/users/plapack/, Last accessed date: September 2005.

[41] E.Menninghaus, B.Lütkenhöner, S.L.Gonzalez, Localization of a dipolar source in a skull phantom: Realistic versus spherical model, *IEEE Trans. Biomed. Eng*, 41, no. 10, pp. 986–989, 1994.

[42] K.Uutela, M.Hamalainen, R.Salmelin, Global Optimization in the Localization of Neuromagnetic Sources, *IEEE Trans. on Biomed. Eng.*, 45, no.6, pp. 716–723, 1998.

[43] M.Obitko, An introduction to genetic algorithms with Java applets, http://cs.felk.cvut.cz/∼xobitko/ga/, 1998.

[44] D.B.Geselowitz, On bioelectric potentials in an inhomogeneous volume conductor *Biophys. J.*, 7, pp.1-11, 1967

[45] R.C.Barr, T.C.Pilkington, J.P.Boineau, M.S.Spach, Determining surface potentials from current dipoles, with application to electrocardiography *IEEE Trans. Biomed. Eng.*, 13, pp.88-92, 1966.

[46] A.C.Barnard, I.M.Duck, M.S.Lynn, The application of electromagnetic theory to electrocardiology: I. Derivation of the integral equations, *Biophys. J.*, 7 pp.443-462, 1967.

[47] A.C.Barnard, I.M.Duck, M.S.Lynn, W.P.Timlake, The application of electromagnetic theory to electrocardiology: II. Numerical solution of the integral equations *Biophys. J.*, 7, pp.463-491, 1967.

[48] J.C.Mosher, R.M. Leahy, and P.S. Lewis, EEG and MEG: Forward Solutions for Inverse Methods, *IEEE Trans. Biomed. Eng.*, 46, no. 3, pp 245–259, 1999.

[49] G.R.Cowper, Gaussian quadrature formulas for triangles, *Int. J. Num. Methods Eng.*, 7, pp. 405-408, 1973.

[50] M.S.Lynn, W.P.Timlake, The use of multiple deflations in the numerical solution of singular systems of equations, with applications to potential theory, *SIAM Journal of Numerical Analysis*, 5, pp. 303–322, 1968.

[51] LU decomposition and its applications, *Numerical Recipies In C: The Art of Scientific Computing*, Cambridge University Press, ISBN 0–521–43108–5, pp.43–50, 1988–1992. http://www.library.cornell.edu/nr/bookcpdf/c2–3.pdf, Last accessed date: September 2005.

[52] M.T.Heath, *SCIENTIFIC COMPUTING: An Introductory Survey*, Second Edition, McGraw-Hill, New York, 2002,

http://www.cse.uiuc.edu/heath/scicomp/notes/, Last accessed date: September 2005.

[53] J.R.Shewchuk, An introduction to the conjugate gradient method without the agonizing pain, Edition 1.25, 1994, http://www.cs.cmu.edu/∼quake-papers/painless-conjugate-gradient.pdf, Last accessed date: September 2005.

[54] Conjugate gradient methods in multidimensions, *Numerical Recipies In C: The Art of Scientific Computing*, Cambridge University Press, pp.420–425, 1988–1992, http://www.library.cornell.edu/nr/bookcpdf/c10–6.pdf, Last accessed date: September 2005.

[55] Message Passing Interface, http://www-unix.mcs.anl.gov/mpi/mpich/, Last accessed date: September 2005.

[56] C.L.Lawson, R.J.Hanson, D.Kincaid, F.T.Krogh, 1979, Basic Linear Algebra Subprograms for FORTRAN usage, *ACM Trans. Math. Soft.*, 5, pp. 308–323.

[57] BLAS, http://www.netlib.org/blas, Last accessed date: September 2005.

[58] BLACS, http://www.netlib.org/blacs/, Last accessed date: September 2005.

[59] PBLAS HOME Page, http://www.netlib.org/scalapack/pblas_qref.html, Last accessed date: September 2005.

[60] R.C.Whaley, A.Petitet, J.J.Dongarra, Automated Empirical Optimization of Software and the ATLAS Project, *Parallel Computing*, 27, no. 1–2, pp. 3–35, 2001.

[61] Automatically Tuned Linear Algebra Software (ATLAS), http://math-atlas.sourceforge.net/, Last accessed date: September 2005.

[62] E.Anderson, Z.Bai, C.Bischof, S.Blackford, J.Demmel, J.Dongarra, J.Du Croz, A.Greenbaum, S.Hammarling, A.McKenney, D.Sorensen, LAPACK Users' Guide, 3.ed. *Society for Industrial and Applied Mathematics*, PA, ISBN 0-89871-447-8, 1999.

[63] LAPACK – Linear Algebra PACKage, http://www.netlib.org/lapack/, Last accessed date: September 2005.

[64] ScaLAPACK, http://www.netlib.org/scalapack/, Last accessed date: September 2005.

[65] PETSc: Documentation, http://www-unix.mcs.anl.gov/petsc/petsc-2/, Last accessed date: September 2005.

[66] AztecOO - Home, http://software.sandia.gov/trilinos/packages/aztecoo/index.html, Last accessed date: September 2005.

[67] Scalable Linear Solvers Project, http://www.llnl.gov/CASC/linear_solvers/, Last accessed date: September 2005.

[68] Octave Home Page, http://www.octave.org/, Last accessed date: September 2005.

[69] Zeynep Akalin Acar, 2005, Electromagnetic source imaging using realistic head models, PhD Thesis, Middle East Technical University, Ankara, Turkey, June 2005.

[70] S.Rush and D.A.Driscoll, EEG electrode sensitivityan application of reciprocity *IEEE Trans. Biomed. Eng.*, 16, pp.15-22, 1969.

[71] V.Sarin, Iterative Methods, *Parallel and distributed numerical algorithms lecture notes*, Texas A&M University, Spring 2005, http://courses.cs.tamu.edu/sarin/CPSC659/, Last accessed date: September 2005.

[72] M.H.Bücker, A 1-norm quasi-minimal residual variant of the Bi-CGSTAB algorithm for nonsymmetric linear systems, *IEEE Int. Conf. on Parallel Processing Workshops*, 7 Sept. 2001, pp.143–148, 2001.

[73] R.Barrett, M.Berry, T.F.Chan, J.Demmel, J.Donato, J.Dongarra, V.Eijkhout, R.Pozo, C.Romine, H.Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Ed.*, SIAM, Philadelphia, PA, p.26, 1994, http://www.netlib.org/linalg/html_templates/report.html, Last accessed date: September 2005.

[74] M.Bücker, A.Basermann, A comparison of QMR, CGS and TFQMR on a distributed memory machine, Technical Report, John von Neumann Institute for Computing, Jülich, Germany, May 1994, http://www.fz-juelich.de/zam/docs/autoren94/buecker2, Last accessed date: September 2005.

[75] B.Hadri, M.Garbey, Case studies in computational fluid dynamic problems, *Parallel Computing Workshop*, April 7-9, University of Houston, Houston, Texas, USA, 2005, http://www.cs.uh.edu/shortcourses/shortcoursesfiles/Hadri/workshop_presentation.ppt, Last accessed date: September 2005.

[76] M.H.Gutknecht, Iterative Methods, ETH Zurich, pp.2–22, 2005, http://www.sam.math.ethz.ch/∼mhg/unt/IterMeth/itmethNM05a.ps, Last accessed date: September 2005.

# APPENDIX A

# DISCRETIZATIONS AND ISOPARAMETRIC MAPPING

In this study, the isoparametric mapping, presented in [25] is applied.

The surfaces that are dealt with for BEM are irregular and hence, discrete models are used for approximating the field on the surfaces. There are various element types and orders for approximation of surfaces by meshes. The order of the element is defined by the order of interpolation functions and the number of nodes per element varies with element degree. The widely used element types are triangle and modifications of triangle to obtain higher order approximations for surface and field over the element. Most commonly used mesh structure is based on triangular (linear) surface elements. However, it is shown that higher order elements can provide significant improvements in approximating curved surfaces and the field on these surfaces.

Since the approximated geometries are not regular, elements are not identical in a mesh. Hence, to be able to define each point on an element, a generic mapping is defined. In this mapping, a local isosceles right triangle with unit perpendicular edges is used as the "generating element" in a fictitious *local domain* with local coordinates $\zeta$ and $\eta$ (figure A.1). The global coordinates of any point on an element are defined by its match in the local domain and global coordinates of element nodes. The mapping from the local domain to the global domain is done through interpolation (shape) functions. A shape function has the form: $N(\zeta, \eta)$. If there are $n$ nodes per element, $n$ shape functions can be defined (one per element node). The global coordinates $(x_p,y_p,z_p)$ of a point $p$, which has a match $(\zeta , \eta)$ in the local domain is given

by:

$$x_p = \sum_{i=1}^{n} N_i(\zeta,\eta)x_i \quad , \quad y_p = \sum_{i=1}^{n} N_i(\zeta,\eta)y_i \quad , \quad z_p = \sum_{i=1}^{n} N_i(\zeta,\eta)z_i$$

where, $x_i$, $y_i$ and $z_i$ are the global coordinates of the i$^{th}$ node of the element, and $N_i$ is the interpolation function for that node.

The elements of first three degree and their interpolation functions are presented below:

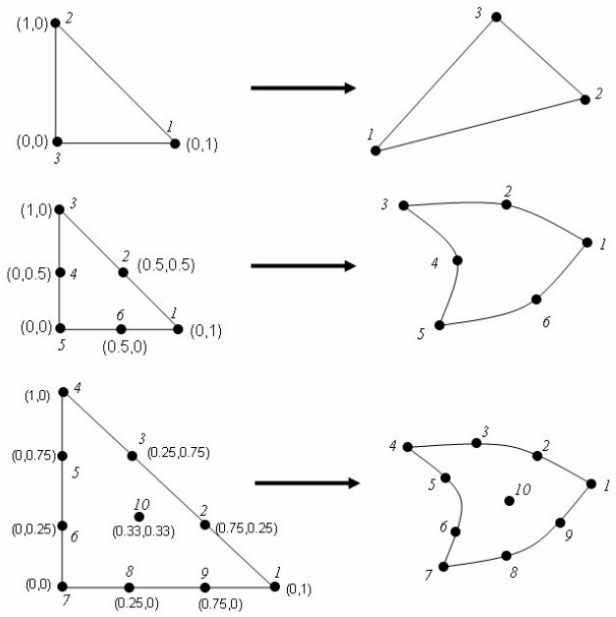


Figure A.1: Elements (on the right) and their local domain models for first three degrees: (a) linear, (b) quadratic, (c) cubic elements.

linear element:

$$\begin{aligned} N_1(\zeta,\eta) &= \zeta \\ N_2(\zeta,\eta) &= \eta \\ N_3(\zeta,\eta) &= 1-\zeta-\eta \end{aligned} \tag{A.1}$$

quadratic element:

$$
\begin{aligned}
N_1(\zeta,\eta) &= \zeta(2\zeta-1)\\
N_2(\zeta,\eta) &= 4\eta\zeta\\
N_3(\zeta,\eta) &= \eta(2\eta-1)\\
N_4(\zeta,\eta) &= 4\eta(-\eta-\zeta+1)\\
N_5(\zeta,\eta) &= \zeta(2\zeta+4\eta-3)+\eta(2\eta-3)+1\\
N_6(\zeta,\eta) &= 4\zeta(-\zeta-\eta+1)
\end{aligned}
\tag{A.2}
$$

cubic element:

$$
\begin{aligned}
N_1(\zeta,\eta) &= 4.5\zeta^2(\zeta-1)-\zeta\\
N_2(\zeta,\eta) &= \eta(13.5\zeta^2-4.5\zeta)\\
N_3(\zeta,\eta) &= \zeta(13.5\eta^2-4.5\eta)\\
N_4(\zeta,\eta) &= 4.5\eta^2(\eta-1)-\eta\\
N_5(\zeta,\eta) &= \eta(-13.5\eta^2+18\eta-13.5\zeta\eta+4.5\zeta-4.5)\\
N_6(\zeta,\eta) &= \eta(-13.5\eta^2-22.5\eta+13.5\zeta^2+27\zeta\eta-22.5\zeta+9)\\
N_7(\zeta,\eta) &= -4.5\zeta^3-4.5\eta^3+9\zeta^2+9\eta^2-13.5\zeta^2\eta-13.5\eta^2\zeta\\
&\qquad +18\zeta\eta-5.5\zeta-5.5\eta+1\\
N_8(\zeta,\eta) &= 13.5\zeta^3-22.5\zeta^2+27\zeta^2\eta-13.5\zeta\eta^2-22.5\zeta\eta+9\zeta\\
N_9(\zeta,\eta) &= -13.5\zeta^3+18\zeta^2-13.5\zeta^2\eta+4.5\zeta\eta-4.5\zeta\\
N_{10}(\zeta,\eta) &= -27\zeta^2\eta-27\eta^2\zeta+27\zeta\eta
\end{aligned}
\tag{A.3}
$$

If the field distribution over the element (potential or magnetic field for BEM case) are also approximated using the same interpolation functions and field values at nodes, the mapping is said to be *isoparametric.* i.e., for example a point $p$ (on the element) has a local domain match ($\zeta$,$\eta$), the potential field value $\phi_p$ at that point is approximated by:

$$\phi_p = \sum_{i=1}^{n} N_i(\zeta,\eta)\phi_i$$

where, $\phi_i$ is the potential value at i$^{th}$ node of the element.

Isoparametric mapping provides a generic field approximation approach for elements of all degree.

# APPENDIX B

# GAUSS-LEGENDRE QUADRATURE AND NUMERICAL INTEGRATION

Surface integrals that cannot be calculated analytically can be approximated by weighted sums of sample points taken on the surface. Accuracy in the calculations is dependent on the number of sample points used, and their distribution. The methodology to determine the sample points is; to determine the number sample points considering the computational load it brings, and then to find the distribution that accurately approximates the highest possible order polynomial for the integration.

Gauss-Legendre quadrature is basically a choice for sample points. In this study, Gauss-Legendre quadrature for 13 points, which is presented in [49] is used. The reader is invited to refer to this text for details.

Gauss-Legendre quadrature is used for matrix filling and hence, no real integration is performed. Rather, for each element, the field at a sample point is approximated by the interpolation functions and node potentials, and the share (in integration) of the element (secondary source) in the field value (at a field point) is distributed to the columns of $\mathbf{A}$ (in the field point's row) corresponding to nodes of that element through interpolation functions.

As mentioned in Appendix A, an isosceles triangle in the local domain is used as a seed for irregularly shaped surface elements and the quadrature points are defined on this local element. Used sample points are shown below and their numerical values (local coordinates) and weights are as follows:

Using the discretizations and mapping presented in Chapter 1 and Appendix A, we convert the surface integral on a surface $S_k$ to its local form:

Table B.1: Local coordinates and weights for Gauss-Legendre quadrature with 13 points

| Sample point (j) | $\zeta$ | $\eta$ | Weight ($w_j$) |
|---|---|---|---|
| 1 | 0.0651301029 | 0.0651301029 | 0.0533472356 |
| 2 | 0.8697297941 | 0.0651301029 | 0.0533472356 |
| 3 | 0.0651301029 | 0.8697297941 | 0.0533472356 |
| 4 | 0.3128654960 | 0.0486903154 | 0.0771137608 |
| 5 | 0.6384441885 | 0.3128654960 | 0.0771137608 |
| 6 | 0.0486903154 | 0.6384441885 | 0.0771137608 |
| 7 | 0.6384441885 | 0.0486903154 | 0.0771137608 |
| 8 | 0.3128654960 | 0.6384441885 | 0.0771137608 |
| 9 | 0.0486903154 | 0.3128654960 | 0.0771137608 |
| 10 | 0.2603459660 | 0.2603459660 | 0.1756152574 |
| 11 | 0.4793080678 | 0.2603459660 | 0.1756152574 |
| 12 | 0.2603459660 | 0.4793080678 | 0.1756152574 |
| 13 | 0.3333333333 | 0.3333333333 | -0.1495700444 |

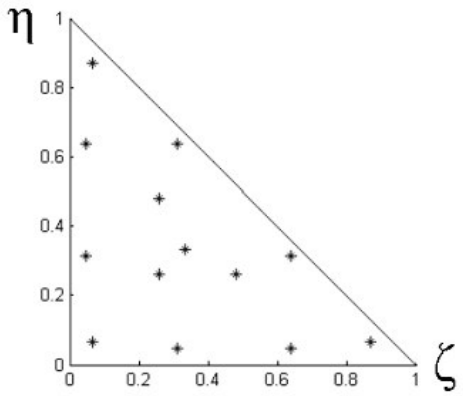


Figure B.1: Gauss-Legendre quadrature with 13 points on local element.

$$\int_{S_k} \phi(\vec{r}\,') \frac{\vec{R}}{R^3} \vec{S}_k(\vec{r}\,') = \sum_{i=1}^{N} \int_0^1 \int_0^{1-\eta} \phi(\vec{r}\,') \frac{\vec{R}}{R^3} \cdot \vec{n} G d\zeta d\eta \tag{B.1}$$

where, $N$ is the total number of surface patches (elements) that constitute $S_k$ and G is the Jacobian due to the transformation from global domain to local domain, which is given by :

$$G = \left\| \frac{\partial \vec{r}'}{\partial \zeta} \times \frac{\partial \vec{r}'}{\partial \eta} \right\|$$

For discretizing the integration in the local domain, we use a summation using Gauss-Legendre quadrature:

$$\int_0^1 \int_0^{1-\eta} f(\zeta, \eta) d\zeta d\eta \approx \frac{1}{2} \sum_{j=1}^{gp} f(\zeta_j, \eta_j) \omega_j \tag{B.2}$$

Here, *gp* is the number of quadrature points, ($\zeta_j$,$\eta_j$) is the local coordinates of jth quadrature point, and $\omega_j$ is its weight. The values presented in table B.1 is used in this study.

After this point, the integration over surface $S_k$ is converted to a summation:

$$\int_{S_k} \phi(\vec{r}\,') \frac{\vec{R}}{R^3} d\vec{S}(\vec{r}\,') = \sum_{i=1}^{N} \frac{1}{2} \sum_{j=1}^{gp} \phi(\zeta_j, \eta_j) \omega_j \frac{\vec{R}(\zeta_j, \eta_j)}{R(\zeta_j, \eta_j)^3} \cdot \vec{n}(\zeta_j, \eta_j) G(\zeta_j, \eta_j) \tag{B.3}$$

We can approximate (through shape functions) the position vector (so the normal) and potential field at ($\zeta_j$,$\eta_j$) by node coordinates and potentials, respectively. i.e., the field and position of a sample point is a summation of "contributions of nodes". Hence, the integration on $S_k$ is converted to a weighted sum of node potentials on $S_k$. Now we can represent the integration in (2.1) as a matrix equation, as in the equation (2.7).

# APPENDIX C

# ALGORITHMS FOR USED KRYLOV SUBSPACE METHODS

There are various forms of KSMs. We present one example for each used method. In the following algorithms, vectors are written in bold italic, algorithm control statement ("for", "end", etc.) are written in italic, matrices are written in bold and other variables are written in regular fonts. The system to be solved is $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$.

## C.1 GMRES

$\boldsymbol{q}_1 = \boldsymbol{b}/||\boldsymbol{b}||$
*for* m = 1, 2, 3, ...
  $\mathbf{v} = \mathbf{A}\boldsymbol{q}_m$
  *for* i = 1,...,m
    $\boldsymbol{h}_{im} = \boldsymbol{q}_i^T \boldsymbol{v}$
    $\boldsymbol{v} = \boldsymbol{v} - \boldsymbol{h}_{im}\boldsymbol{q}_i$
  *end*
  $\boldsymbol{h}_{m+1,m} = ||\boldsymbol{v}||$
  $\boldsymbol{q}_{m+1} = \boldsymbol{v}/\boldsymbol{h}_{m+1,m}$
  find $\boldsymbol{y}$ to minimize $||\mathbf{H}_m\boldsymbol{y}$ - $||\boldsymbol{b}||\boldsymbol{e}_1||$
  $\boldsymbol{x} = \mathbf{Q}_m\boldsymbol{y}$
*end*

GMRES algorithm [71]

Here, $\mathbf{H}_m$ is an upper Hessenberg matrix of size *(m+1)* $\times$ $m$ which is constructed using previous search directions. The minimization problem is solved using QR factorization over $H_m$.

## C.2 Bi-CGSTAB

Choose initial guess $\boldsymbol{x}_0$, $\boldsymbol{r}_0 = \boldsymbol{b}$ - $\mathbf{A}\boldsymbol{x}_0$, $\boldsymbol{v}_0 = \boldsymbol{p}_0 = 0$
Choose $\boldsymbol{r}_0$' such that $(\boldsymbol{r}_0\text{'})^T\boldsymbol{r}_0 \neq 0$
$\rho_0 = \alpha_0 = \eta_0 = 1$
*for* n = 1, 2, 3...
$\rho_n = (\boldsymbol{r}_0\text{'})^T\boldsymbol{r}_{n-1}$
$\beta_n = (\rho_n/\ \rho_{n-1})(\ \alpha_{n-1}/\ \eta_{n-1})$
$\boldsymbol{p}_n = \boldsymbol{r}_{n-1} + \beta_n(\boldsymbol{p}_{n-1} - \eta_{n-1}\boldsymbol{v}_{n-1})$
$\boldsymbol{v}_n = \mathbf{A}\boldsymbol{p}_n$
$\alpha_n = \rho_n/\ (\boldsymbol{r}_0\text{'})^T\boldsymbol{v}_n$
$\boldsymbol{s}_n = \boldsymbol{r}_n$-1-$\alpha_n\boldsymbol{v}_n$
$\boldsymbol{t}_n = \mathbf{A}\boldsymbol{s}_n$
$\eta_n = \boldsymbol{t}_n^T\boldsymbol{s}_n/\boldsymbol{t}_n^T\boldsymbol{t}_n$
$\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + \alpha_n\boldsymbol{p}_n + \boldsymbol{s}_n$
*Stop if* $\boldsymbol{x}_n$ converged
$\boldsymbol{r}_n = \boldsymbol{s}_n - \eta_n\boldsymbol{t}_n$
*end*

Bi-CGSTAB algorithm [72]

## C.3 CGS

Choose initial guess $\boldsymbol{x}_0$, $\boldsymbol{r}_0 = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}_0$
Choose r'$_0$ such that, $\boldsymbol{r}_0^T \boldsymbol{r}_0\,\boldsymbol{'} \neq 0$, $\rho_0 = \boldsymbol{r}_0^T \boldsymbol{r}_0\,\boldsymbol{'}$
$\beta_{-1} = \rho_0$; $\boldsymbol{p}_{-1} = \boldsymbol{q}_0 = \boldsymbol{0}$
*for* i = 0, 1, 2...
$\boldsymbol{u}_i = \boldsymbol{r}_i + \beta_{i-1}\mathbf{q}_i$
$\boldsymbol{p}_i = \boldsymbol{u}_i + \beta_{i-1}(\boldsymbol{q}_i + \beta_{i-1}\boldsymbol{p}_i)$
solve $\boldsymbol{p}\boldsymbol{'}$ from $\mathbf{K}\boldsymbol{p}\boldsymbol{'} = \boldsymbol{p}_i$
$\boldsymbol{v}\boldsymbol{'} = \mathbf{A}\boldsymbol{p}\boldsymbol{'}$
$\alpha_i = \rho_i/(\boldsymbol{v}\boldsymbol{'}^T \boldsymbol{r}\boldsymbol{'}_0)$
$\boldsymbol{q}_{i+1} = \boldsymbol{u}_i - \alpha_i \boldsymbol{v}\boldsymbol{'}$
solve $\boldsymbol{u}\boldsymbol{'}$ from $\mathbf{K}\boldsymbol{u}\boldsymbol{'} = \boldsymbol{u}_i + \boldsymbol{q}_{i+1}$
$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \alpha_i\ \boldsymbol{u}\boldsymbol{'}$
*if* $\boldsymbol{x}_{i+1}$ is accurate enough, *stop*
$\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - \alpha_i \mathbf{A}\boldsymbol{u}\boldsymbol{'}$
$\rho_{i+1} = \boldsymbol{r}_{i+1}^T \boldsymbol{r}\boldsymbol{'}_0$
*if* $\rho_{i+1} = 0$ method fails to converge!
$\beta_i = \rho_{i+1}/\rho_i$
*end*

CGS algorithm with preconditioner $\mathbf{K}$ [73]

## C.4 TFQMR

Choose initial guess $\boldsymbol{x}_0$, $\boldsymbol{w}_1$=$\boldsymbol{y}_1$ = $\boldsymbol{r}_0$ = $\boldsymbol{b}$ - $\mathbf{A}\boldsymbol{x}_0$, $\boldsymbol{v}_0$ =$\mathbf{A}\boldsymbol{y}$, $\boldsymbol{d}_0$=$\boldsymbol{0}$

$\tau_0 = ||\boldsymbol{r}_0||$, v'$_0 = 0$, $\eta_0 = 0$.

Choose $\boldsymbol{r'}_0$ such that $\boldsymbol{r'}_0 \neq \boldsymbol{0}$

$\rho_0 = r_0'^T \boldsymbol{r}_0$

*for* n = 1, 2, 3...

$\sigma_{n-1} = r_0'^T \boldsymbol{v}_{n-1}$

$\alpha_{n-1} = \rho_{n-1} / \sigma_{n-1}$

$\boldsymbol{y}_{2n} = \boldsymbol{y}_{2n-1} - \alpha_{n-1}\boldsymbol{v}_{n-1}$

*for* m=2n-1,...2n

$\boldsymbol{w}_{m+1} = \boldsymbol{w}_m - \alpha_{n-1}\mathbf{A}\boldsymbol{y}_m$

v'$_m = ||\boldsymbol{w}_{m+1}||/\tau_{m-1}$, c$_m = 1/\sqrt{1 + v_m'^2}$

$\tau_m = \tau_{m-1}$v'$_m$c$_m$, $\eta_m$ = c$_m 2\alpha_{n-1}$

$\boldsymbol{d}_m = \boldsymbol{y}_m + (\text{v'}_{m-1}\eta_{m-1}/\alpha_{n-1})\boldsymbol{d}_{m-1}$

$\boldsymbol{x}_m = \boldsymbol{x}_{m-1} + \eta_m \boldsymbol{d}_m$

*if* $\boldsymbol{x}_m$ converged, *stop.*

*end*

$\rho_n = r_0'^T \boldsymbol{w}_{2n+1}$

$\beta_n = \rho_n / \rho_{n-1}$

$\boldsymbol{y}_{2n+1} = \boldsymbol{w}_{2n+1} + \beta_n \boldsymbol{y}_{2n}$

$\boldsymbol{v}_n = \mathbf{A}\boldsymbol{y}_{2n+1} + \beta_n(\mathbf{A}\boldsymbol{y}_{2n} + \beta_n \boldsymbol{v}_{n-1})$

*end*

TFQMR algorithm [74]

## C.5 CR

Choose initial guess $\boldsymbol{x}_0$, $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$, $\boldsymbol{p}_0 = \boldsymbol{r}_0$
*for* j = 0, 1, 2...
$\alpha_j = \boldsymbol{r}_j^T \mathbf{A} \boldsymbol{r}_j / (\mathbf{A}\boldsymbol{p}_j)^T \mathbf{A}\boldsymbol{p}_j$
$\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \alpha_j \boldsymbol{p}_j$
$\boldsymbol{r}_{j+1} = \boldsymbol{r}_j - \alpha_j \mathbf{A}\boldsymbol{p}_j$
*if* $\boldsymbol{x}_{j+1}$ is accurate enough, *stop*
$\beta_j = (\boldsymbol{r}_{j+1}^T \mathbf{A} \boldsymbol{r}_j) / \boldsymbol{r}_j^T \mathbf{A} \boldsymbol{r}_j$
$\boldsymbol{p}_{j+1} = \boldsymbol{r}_{j+1} + \beta_j \boldsymbol{p}_j$
$\mathbf{A}\boldsymbol{p}_j = \boldsymbol{A}\boldsymbol{r}_j + \beta_j \mathbf{A}\boldsymbol{p}_j$
*end*

CR algorithm [75]

## C.6 CGNE

Choose initial guess $\boldsymbol{x}_0$, $\boldsymbol{v}_0 = \boldsymbol{r}_0 = \boldsymbol{b}$-$\mathbf{A}\boldsymbol{x}_0$, $\delta_0 = ||\boldsymbol{r}_0||^2$, $\psi_{-1} = 0$
*for* n = 1, 2, 3...
$\delta_n$' $= ||\boldsymbol{v}_n||^2$
$\omega_n = \delta_n / \delta_n$'
$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{v}_n \omega_n$
$r_{n+1} = \boldsymbol{r}_n - \mathbf{A}\boldsymbol{v}_n \omega_n$
*if* $||\boldsymbol{r}_{n+1}||$ is sufficiently small, *stop*
$\delta_{n+1} = ||r_{n+1}||^2$
$\psi_n = -\delta_{n+1} / \delta_n$
$\boldsymbol{v}_{n+1} = \mathbf{A}^T \boldsymbol{r}_{n+1} - \boldsymbol{v}_n \psi_n$
*end*

CGNE algorithm [76]

# APPENDIX D

# CONSTRUCTION OF THE ATHLINS CLUSTER

The construction of Athlins cluster has two phases: 1)Hardware setup, 2)Software setup. The details of these phases are presented below:

## D.1 Hardware Setup

The hardware setup starts with the selection of the hardware components that are suitable for parallelization purposes. The most important parts are the Central Processor Units (CPUs), RAMs and the Motherboards (Main boards). These three parts also constitute the main expense of cluster construction.

Today's CPUs have two alternatives for CPU producers: Advanced Micro Devices Corp. (AMD) and Intel Corp. These producers provide a variety of CPUs dedicated for various applications. During CPU selection the following properties are important:

- **CPU clock speed:** CPUs operate faster with increased clock speed as long as the clock speed is fully utilized by the CPU.

- **CPU architecture and instruction sets:** CPU architecture (i.e, the instruction set) determine the number of clock cycles spent per operation (addition, multiplication, looping, etc.). AMD is known to provide reduced instruction set computing (RISC) CPUs which has

significant advantage compared to complex instruction set computing (CISC) CPUs of Intel.

- **CPU cache:** CPU cache is the memory that is embedded in the CPU. Generally, there are two levels of caches: level–1 (L1) and level–2 (L2) caches. L1 caches operate at the same frequency (clock speed with the processor) and are generally smaller in size (typical PC CPUs have a few hundred KB L1 cache). L2 caches are larger in size (up to a few MB), but slower than L1 caches although they are faster than RAMs. Since both L1 and L2 caches are faster than RAMs, larger cache sizes improve CPU performance.

- **CPU Front Side Bus (FSB) speed:** FSB is the data bus between the processor and the RAM. FSB is always slower than the CPU clock speed and thus, the performance is limited by FSB speed. Currently, Intel CPUs support up to 800MHz FSB speed while ADM CPUs have relatively slower FSB speed. However, as long as today's RAM speed limitations are concerned, there is no significant difference between AMD and Intel CPUs since both support the upper limit for RAM speeds (except for some expensive RAMs).

The processors selected for Athlins cluster are AMD XP 2500+ (1833MHz) processors. Each processor has 128KB L1 cache and 512KB L2 cache.

Selection of RAM is not very complicated. Due to large memory needs, the RAM size per PC should be kept as large as possible. Additionally, the RAM speed must be kept as much as possible. Athlins cluster has 1.5 GB of RAM per PC, each of which operate at 333MHz.

Selection of the motherboard is again relatively simple. The key point is the FSB speed supported by the motherboard. If this speed is high, further

modifications of the cluster will be possible. The motherboards of the Athlins cluster support up to 400MHz FSB speed.

The Athlins cluster is constructed over a 1000MB/s Ethernet with a compatible Ethernet switch and Ethernet adapters on the PCs.

## D.2 Software Setup

The construction of the cluster is done by installing and configuring the software used in the platform. The steps for constructing a fully functional scientific computation cluster that provides the PETSc library is as follows:

1. Installing the operating system,
2. Setting up the network,
3. Configuring and installing the MPI library (version: mpich-1.2.5.2),
4. Configuring and installing BLAS library by running ATLAS (version: ATLAS-3.6.0),
5. Configuring and installing LAPACK library with installed BLAS (version: LAPACK-3.0 + updates at May 21, 2001),
6. Configuring and installing PETSc library (version: petsc-2.2.0) with installed MPI, BLAS and LAPACK.

The details of these steps are presented in the following sections. Testing phases are complicated and omitted here. However, performing the tests is strongly recommended. The interested reader should refer to manuals of these libraries.

### D.2.1 Installing the Operating System

The operating system (OS) is selected to be Linux. Linux is a flexible and open-source OS, which is also user friendly and easy to configure. On

the Athlins cluster, the Mandrake 10.0 Linux distribution is installed. The graphical interface is installed, but configured such that it is not started at the OS start-up. This is to prevent unnecessary RAM usage. Necessary driver for the Ethernet devices is also installed manually.

### D.2.2 Network Setup

The MPI library is used with remote shell (rsh), in which the processing nodes are accepted to be trusted agents. For that, the cluster uses a closed network which cannot be accessed directly from The Internet. Instead, the gateway (Marvin) is used. Marvin has two Ethernet devices; one is for The Internet connection and the other is for accessing the closed network.

The configuration of the Athlins nodes for network setup is as follows:

Every user's home directory is kept on Marvin under `/home/linux/username`. This avoids synchronization problems. For that, the file `/etc/fstab` is modified on each processor to be as:

```
/dev/hde1 / ext3 defaults 1 1 # determined by OS
none /dev/pts devpts mode=0620 0 0 # determined by OS
none /proc proc defaults 0 0 # determined by OS
/dev/hde7 /usr ext3 defaults 1 2 # determined by OS
/dev/hde5 swap swap defaults 0 0 # determined by OS
master:/home/linux /home nfs defaults 0 0 # home dir.
master:/home/export/opt /opt nfs defaults 0 0
```

During OS installation, rsh is selected to be installed and run at start-up. A number of files are modified for network setup:

```
/etc/sysconfig/network
     HOSTNAME=<computer name> # computer1, computer2, etc.
     NETWORKING=yes
```

```
GATEWAY=10.0.0.1
```

/etc/sysconfig/network-scripts/ifcfg-eth0

```
DEVICE=eth0
BOOTPROTO=none
IPADDR=10.0.0.41 # 42 for computer2, etc.
NETMASK=255.0.0.0
NETWORK=10.0.0.0
BROADCAST=10.255.255.255
ONBOOT=yes
```

/etc/hosts and /etc/hosts.equiv

```
10.0.0.1 master
10.0.0.41 computer1
10.0.0.42 computer2
#...  ...
```

/etc/pam.conf or /etc/pam.d/rsh

```
# the line that says
# rsh auth required pam_rhosts_auth.so.1
# is replaced with:
rsh auth sufficient pam_rhosts_auth.so.1
```

/etc/securetty

```
rsh # this is added in a separate line if not existent.
```

/home/username/.rhosts (/home/linux/username/.rhosts on Marvin)

```
master
computer1
computer2
```

```
#...
```

### D.2.3 MPI Installation

Assuming the MPI archive is extracted under `/usr/local/mpich-<version>/` directory, the installation is done as follows:

In `/usr/local/mpich-<version>/`, first the configuration script is run by

```
./configure --with--device=ch_p4
--prefix=/usr/local/mpich-<version>/ch_p4
--with-common-prefix=/usr/local/mpich-<version>
```

Installation:

```
make
make install
```

For directly calling mpirun (the binary for running MPI commands) as a Linux command, this file must be copied to appropriate locations:

```
cp /usr/local/mpich-<version>/bin/mpirun /usr/bin
cp /usr/local/mpich-<version>/bin/mpirun /usr/local/bin
```

After the installation, the machine names of the cluster must be inserted into the machines file:

```
/usr/local/mpich-<version>/ch_p4/share/machines.LINUX
  computer1:2
  computer2:2
```

```
computer3:2
# ...
```

The number after the colon states the number of processors of the computer.

### D.2.4 BLAS Installation with ATLAS

Assuming the ATLAS archive is extracted under `/usr/local/ATLAS/`, the installation is done as follows:

In `/usr/local/ATLAS/` directory,

```
make
```

After obeying the suggestions of ATLAS,

```
make install arch=<your_architecture_name>
```

At the end of installation, the optimized libraries of ATLAS will be under `/usr/local/ATLAS/lib/<your_arch_name>/`

### D.2.5 LAPACK Installation

Assuming the LAPACK archive is extracted under `/usr/local/LAPACK/`, the installation is done as follows:

```
cp /INSTALL/make.inc.LINUX ./make.inc
```

This file should be modified (the line in which the BLASLIB is stated) as

```
BLASLIB=/usr/local/ATLAS/lib/<your_arch_name>/libf77blas.a
/usr/local/ATLAS/lib/<your_arch_name>/libatlas.a
```

After that, the library is installed (without any tests) using

```
make install lib
```

### D.2.6 Inserting ATLAS Routines in LAPACK

ATLAS provides all BLAS routines in addition to some LAPACK routines. For inserting these optimized binaries to the LAPACK library one should perform:

```
cd /usr/local/LAPACK
mkdir tmp
cd tmp
ar x ../<lapack_ARCH>.a
cp <your ATLAS LAPACK library path> ../liblapack.a
ar x ../liblapack.a
ar r ../liblapack.a *.o
cd ..
rm -rf tmp
mv /usr/local/ATLAS/lib/<your_arch_name>/liblapack.a
/usr/local/ATLAS/lib/<your_arch_name>/liblapack_original.a
cp liblapack.a /usr/local/ATLAS/lib/<your_arch_name>
```

## D.3 PETSc Installation

Assuming the PETSc archive is extracted under `/usr/local/petsc-<version>/`, the installation is done as follows:

```
export PETSC_ARCH=linux
export PETSC_DIR=/usr/local/petsc-<version>
cd /usr/local/petsc-<version>/bmake/linux/
```

The file /usr/local/petsc-<version>/bmake/linux/packages should be modified:

```
#For static linking use (recommended):
#BLASLAPACK_LIB = -Wl,-Bstatic
-L$/usr/local/ATLAS/lib/<your_arch_name>/lib -llapack -lf77blas
-latlas
# For dynamic linking use:
BLASLAPACK_LIB = -L$/usr/local/ATLAS/lib/<your_arch_name>
-llapack -lf77blas -latlas
MPI_HOME = /usr/local/mpich-<version>
MPI_LIB = ${CLINKER_SLFLAG/usr/local/mpich-<version>/lib
-L/usr/local/mpich-<version>/lib -lmpich -lpmpich
/usr/local/mpich-<version>/lib/libfmpich.a
```

The usage of static libraries is recommended for PETSc. Static libraries provide faster operation. For installing PETSc with static libraries, one must modify /usr/local/petsc-<version>/bmake/linux/petscconf.h by replacing

```
#define PETSC_USE_DYNAMIC_LIBRARIES 1
```

with

```
#undef PETSC_USE_DYNAMIC_LIBRARIES
```

Although this modification seems to be conflicting with the configuration made in the packages file, it is not the case. The library is correctly installed.

In /usr/local/petsc-<version>/bmake/linux/petscconf.h, the dele-

tion of all occurrences of "-lfrtbegin" may be necessary in some cases.

In /usr/local/petsc-<version> directory, the installation is performed using:

```
make BOPT=g all # debugging libraries

make BOPT=g_c++ all # debugging libraries with C++ bindings

make BOPT=O_c++ all # optimized libraries with C++ bindings

make BOPT=O all # optimized libraries
```

Tests are performed by:

```
make BOPT=O_c++ testexamples

make BOPT=O_c++ testfortran
```

The user must also set the PETSC_DIR as required. This can be done by editing one of the .bashrc, .bash_profile, .shrc, etc. files depending on the used shell. In bash shell, this can be done by adding two lines to the .bash_profile as:

```
PETSC_DIR=/usr/local/petsc-<version>
export PETSC_DIR
```

### D.3.1 Sample Makefile

Here, a simple and useful makefile is presented. The makefile is used with one of the four options which are:

```
make BOPT=g MyProgram
make BOPT=O MyProgram
make BOPT=g_c++ MyProgram
make BOPT=O_c++ MyProgram
```

A simple makefile is as follows:

```
# Makefile*****************************
# make file based on the minimum given in the manual

ALL:MyProgram
CFLAGS = -Wall -Wshadow
FFLAGS =
CPPFLAGS =
FPPFLAGS =

MYPROGRAM_SRC = MyProgram.c
MYPROGRAM_OBJ = ${MYPROGRAM_SRC:c=o}

include ${PETSC_DIR}/bmake/common/base

MyProgram:  ${MYPROGRAM_OBJ} chkopts
${CLINKER} -o $@ ${MYPROGRAM_OBJ} ${PETSC_LIB} -lz
-L/usr/local/ATLAS/lib/<your_arcitecture_name>/ -lcblas -latlas

# END Makefile ***********************
```