PARALLELIZATION OF LIGHT SCATTERING SPECTROSCOPY AND ITS INTEGRATION WITH COMPUTATIONAL GRID ENVIRONMENTS

By

JITHENDAR PALADUGULA

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2004

This thesis is dedicated to the Almighty and my family.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Dr. Renato J. Figueiredo for providing me with such an interesting topic and for his valuable guidance and support in this research effort. I am also thankful to Dr. Jose A.B. Fortes and Dr. Alan D. George for serving on my supervisory committee and for their useful comments.

I would like to extend my sincere thanks to my colleagues in Advanced Computing and Information Systems (ACIS) lab for their help and inspiration. I am also grateful to my roommates and friends for being friendly and making my life at the University of Florida memorable.

Last but not least, I thank my family for their love, encouragement and constant support in all my endeavors.

TABLE OF CONTENTS

page

LIST OF TABLES

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

PARALLELIZATION OF LIGHT SCATTERING SPECTROSCOPY AND ITS INTEGRATION WITH COMPUTATIONAL GRID ENVIRONMENTS

By

Jithendar Paladugula

August 2004

Chair: Renato J. Figueiredo
Major Department: Electrical and Computer Engineering

In today's world, Grid computing is enabling the development of novel medical applications. Computational Grids have the potential to provide access to high-performance resources from medical facilities with network-enabled devices. This thesis considers the integration of medical applications with grids in the context of Light Scattering Spectroscopy (LSS), a nascent application in the medical imaging field.

The LSS analysis enables the extraction of quantitative information about the structure of living cells and tissues by comparing the spectrum of light backscattered from tissues with the databases generated using a Mie-theory-based analytical model. For high accuracy analysis, large databases with fine-grain resolution must be considered. This thesis describes the design and implementation of parallel algorithms for LSS and its integration with Grid environments. The high computational requirements of the problem are catered by parallelizing the LSS analysis and database generation algorithms using Message Passing Interface (MPI). The LSS analysis is amenable to parallelism, as it can

be done across multiple independent databases. The integration of LSS with grids enables the collection of data at a medical facility and processing of data to occur on remote high-performance resources. Data management and communication challenges with this network-computing model are addressed by data transfer techniques based on Virtual File Systems (VFS). The VFS technique is built on top of Network File System (NFS) via the use of proxies that intercept and forward Remote Procedure Call (RPC) requests and support client-side disk caching.

Results from performance analyses that consider the 16-processor parallel execution of LSS analysis on a local disk show 13x speedups for an image consisting of beads suspended in water. For this dataset, an accuracy of 99.4% in the diameter size is achieved. Experiments on the virtualized wide-area distributed file system setup and configured by Grid middleware show that performance levels (13% overhead or less) close to those of a local disk can be achieved in the presence of locality of references.

CHAPTER 1
INTRODUCTION

1.1 Motivation

Extensive research has been performed in recent decades to develop devices that can detect and treat cancers in order to save lives. Research is also being done to expand the capabilities of these devices to allow them to predict pre-cancerous changes in tissue so that patients and doctors may take preventive measures to avoid cancer.

Light Scattering Spectroscopy (LSS) devices, and the associated analysis techniques that allow non-invasive detection of pre-cancerous changes in human epithelium, have been recently proposed and investigated [1-5]. The LSS imaging differs from traditional biopsies by allowing in-vivo diagnosis of tissue samples; and by providing an automated, quantitative analyses of parameters related to cancerous changes (e.g., nuclei enlargement) via numerical techniques. As a result, LSS can yield significant advances in healthcare: it has the potential to detect the majority of currently undetectable cancers and significantly reduce cancer mortality (by up to 50%) [4].

However, to achieve high accuracy, the analysis of LSS images requires computational- and data-intensive solutions. The LSS analysis application is based on the processing of backscattered spectral image of a region of tissue using a Mie-theory-based inversion procedure. For each pixel in the image, the size and refractive indices of the scatterers that best fit the data among those stored in a database of Mie-theory generated spectra are found using a least-square error minimization approach. For high-accuracy

analysis, large databases with fine-grain high resolution of sizes and refractive indices must be considered.

In the past few decades, high-performance computing has driven the development of practical medical applications that are now widely available (such as magnetic resonance imaging and computerized tomography). These solutions have been enabled by sustained performance improvements in the embedded systems that typically support medical applications. In the recent years, information processing is undergoing rapid advances driven by the use of distributed computing systems connected by world-wide networks. Previous efforts in the coupling of instrumentation and distributed infrastructures have considered image-processing applications – such as parallel analysis of data collected from electron microscopes [6, 7]. Results from previous work motivate the use of network-based computing to solve applications with similar characteristics (computational-intensive and with high degrees of parallelism) including medical applications [8]. The LSS imaging is an application that can greatly benefit from a network-computing model, given its substantial computational requirements and amenability to parallelism.

Our goal was to develop computing techniques that enable accurate and fast analyses of LSS images; and to integrate the application with grid environments. Analogous to power grids, “computational grids” have the potential to provide seamless access to high-performance resources (e.g., parallel supercomputers) from ubiquitous, network-enabled devices, thus providing large computing power and data storage space.

The widespread deployment of LSS imaging depends on the availability of high-performance resources to enable the processing of computationally intensive

algorithms. From a computational standpoint, solutions are developed to allow LSS analysis with both fast response time and high accuracy. From a communication standpoint, data management techniques are developed that support a scenario where data is always produced and displayed by the medical device, while processing occurs in nodes that do not have local access to the data.

The computation of LSS analyses and the generation of Mie-theory databases are based on parallel algorithms implemented in C that use the Message Passing Interface (MPI) library for communication. The MPI [9] is an efficient message passing library that can be used on many types of architectures and operating systems environments, therefore making it highly portable. The data management technique is based on virtual file systems [10]. The virtual file system provides seamless access to data and facilitates application development by leveraging on-demand and application-transparent transfer of data using Network File System (NFS) abstractions and implementations available in existing Operating Systems (O/S).

1.2 Summary of Results

Performance of LSS image analysis and the database generation is improved by implementing them as parallel programs. Performance data reported in this thesis show that speedups of 13x and 16x are achieved with 16 processors for LSS analysis and database generation parallel implementations respectively.

The results from the experiments with virtual file system show that user-level proxy disk caches allow the proposed approach to achieve performance close to that of local disk (13% slower) and superior to non-virtualized NFS setups in both LAN (83% speedup) and WAN (680% speedup) environments for a 16-processor parallel execution of LSS analysis. Results also show that implementing a user-level write-back cache

policy allows the parallel generation of LSS databases to achieve performance close to that of local disk (3% slower) and non-virtualized LAN (2% faster); and superior to non-virtualized WAN (280% speedup).

1.3 Overview

The rest of the thesis is organized as follows. Chapter 2 describes the Light Scattering Spectroscopy application that was used in our study, its biomedical background and parallel implementations of algorithms based on MPI and file I/O. Chapter 3 discusses the data management requirements for the LSS application and solutions based on MPI and virtual file system. Chapter 4 presents results and analysis of experiments that consider the LSS performance with the grid virtual file system. Chapter 5 describes integration of the application with network-computing portals that can be accessed from clients across the Internet through interfaces ranging from those found in typical desktops (e.g., web browsers) to those found in portable and embedded devices. Chapter 6 discusses the related work. Conclusions and ideas for future work are presented in Chapter 7.

CHAPTER 2
LIGHT SCATTERING SPECTROSCOPY

2.1 Introduction

Light Scattering Spectroscopy (LSS) with polarized light is a nascent technology developed to probe the structure of the living cells and tissues by analyzing the spectrum of light backscattered from tissues [1-5]. It helps in non-invasive detection of pre-cancerous and early cancerous changes in human epithelium (e.g., oral cavity). The LSS application can yield significant advances in healthcare. It has the potential to detect the majority of currently undetectable cancers and significantly reduce cancer mortality (by up to 50%) [4]. However, to achieve high accuracy, the analysis of LSS images requires computational- and data-intensive solutions that perform spectral analyses based on Mie-theory inversion procedure.

Figure 2-1 shows the LSS process. The information obtained by LSS analysis consists of the size distribution and refractive index of an area of tissue, based on data obtained from an optical apparatus designed for LSS imaging. A sample of tissue is illuminated by a light source and the spectrum of backscattered light for both parallel and perpendicular polarizations is collected for multiple wavelengths using a charged-couple device (CCD). The spectral image obtained from the LSS instrument is compared with databases of Mie-theory spectra to obtain the size and refractive index parameters of the tissue based on least-square error minimization. Thus the LSS analysis application is based on processing the backscattered spectral image of a tissue using Mie-theory spectra databases.

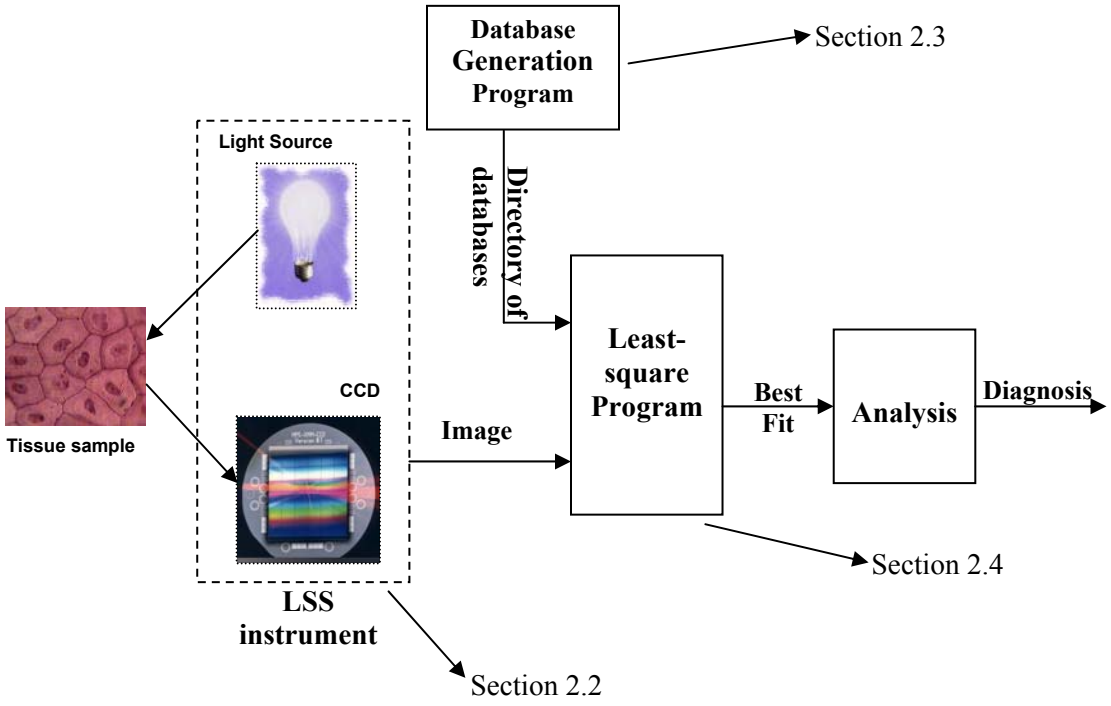

Figure 2-1. Overview of Light Scattering Spectroscopy process

2.2 Application Background

The LSS technique is based on an optical instrument (Figure 2-2) that includes a digital image CCD that records spectra of backscattered light for both parallel and perpendicular polarizations. A beam of light from a broadband source is spatially filtered and collimated and then refocused with a small solid angle onto the sample, using lenses (L1 and L2) and an aperture A1. A series of narrow-band optical filters F selects a narrow wavelength band. The beam is then polarized by a polarizer P1, passed through a second aperture A2 which reduces the diameter of the beam to about 3mm, reflected from the mirror M, and made incident on the sample stage on which the tissue sample is mounted. In order to avoid the specular reflectance, the incident beam is oriented at an angle of about 15º to the normal surface of the sample by adjusting the mirror. The sample stage is placed at one focal plane of lens L3 while a 512 x 512 pixel CCD camera is placed at

the other focal plane where a two dimensional map of the angular distribution of the backscattered light is formed. The polarization state of the scattered light is selected by the analyzer P2. When P1 and P2 are parallel, the scattered intensity is denoted by $I_{\parallel}$ and when the analyzers are cross-polarized, the scattered intensity is denoted by $I_{\perp}$.

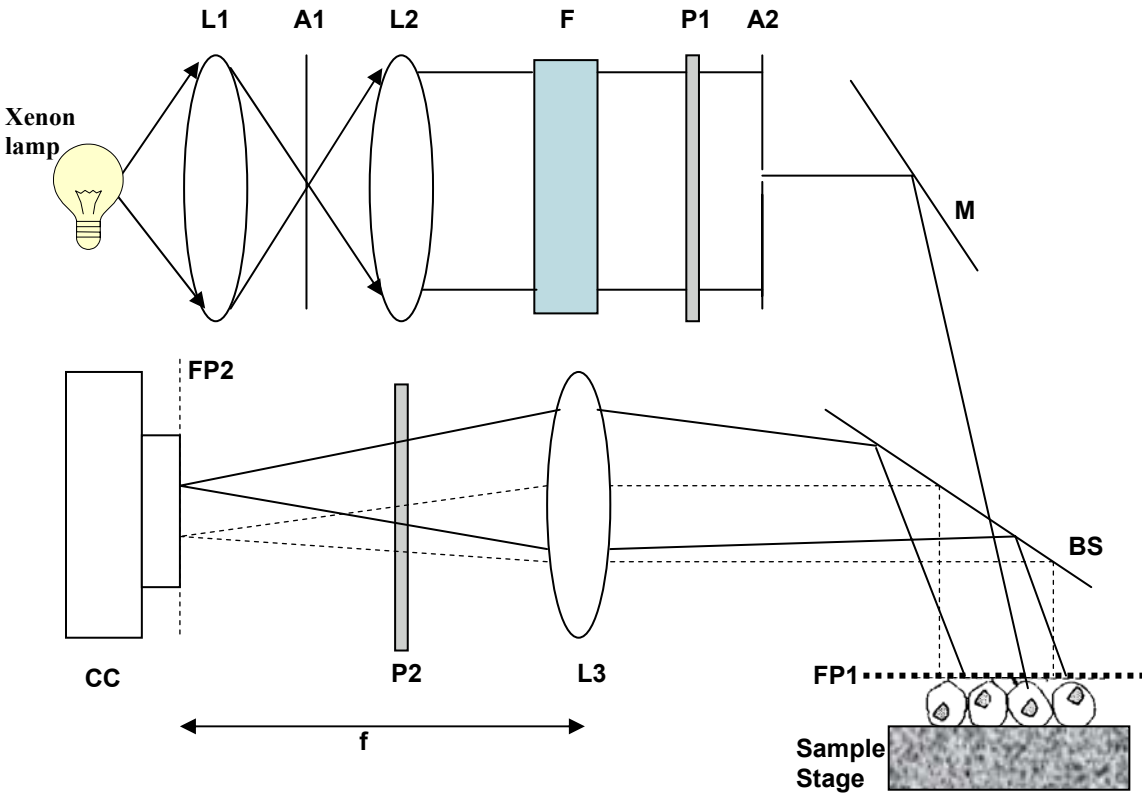

Figure 2-2. An LSS instrument. The sample is mounted on the sample stage.

The optical path from the sample stage to focal plane FP2 is unfolded and shown without the beam splitter and the components are placed in a straight line in Figure 2-3. It can be seen that a 2-D map of angular distribution of backscattered light is formed in the camera focal plane FP2. The center of the exact backscattering is mapped onto the center of the image at O. The two rays marked R1 and R2 represent light that is scattered at polar angle θ (measured from the backward direction) and azimuth φ to the direction of exact backscattering from different parts of the sample. φ = 0 for light scattered in a

direction with (x, y) component perpendicular to the direction of polarization of the incident light x. All such parallel rays that leave the sample with identical scattering angles (θ, φ) pass through the same point P in the focal plane FP2. Thus, light rays scattered from all parts of the sample at fixed angle (θ, φ) are mapped onto a fixed position in the camera image plane [3].

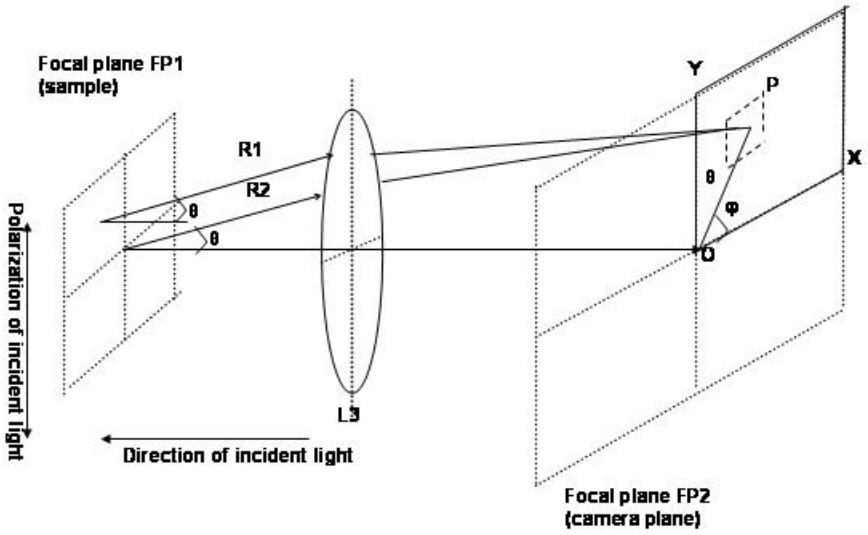

Figure 2-3. Unfolded view of the optical path from the sample stage to the camera plane. A spatial map of the angular distribution of light scattered by the sample in the plane FP1 is formed at the camera which is placed at the plane FP2.

The angular and spectral distributions of light scattered elastically by a particle are determined by the size of the particle relative to the wavelength of the incident light, and by the refractive index mismatch between the particle and the surrounding medium. The angular and spectral variations characterize the size and other features of the scattering particle. Broadly, there are two strategies by which particle size can be determined. The particle can be illuminated by light of fixed wavelength and the angular variations of the scattered intensity measured. Alternatively, light can be collected from a fixed angle (θ, φ) to measure the intensity changes of the scattered light as the wavelength is varied. The

technique LSS employs exploit both the angular and spectral features of the scattered light.

The backscattered light can be divided into two parts: one due to single scattering events and the other due to multiple scattered light. The multiple scattered light is used to probe the deeper tissues, while the single scattered light is due to the scatterers located close to the tissue surface. Hence the single scattered light is sensitive to very small changes in the cell nuclear parameters. When polarized light is incident on a sample of tissue, a small portion of the incident light is backscattered by the cells, retaining the polarization. The other portion of the light diffuses into the tissue and is depolarized by multiple scattering. The backscattered light is analyzed for two polarizations: parallel and perpendicular to the incident light. The contribution of the multiple scattering is significantly eliminated by subtracting off the depolarized portion of the total signal. This is achieved by subtracting the cross-polarized component ($I_{\perp}$) from the co-polarized component ($I_{\parallel}$). The resulting scattering intensity ($\Delta I = I_{\parallel} - I_{\perp}$) is due to single scattering alone and enables us to determine the nuclear sizes and their refractive index.

The LSS backscattered light images (i.e., intensity maps in the (θ, φ) plane) at various wavelengths and both polarizations are obtained. For each narrow wavelength band six intensity maps are recorded. First $I_{\parallel}$ and $I_{\perp}$, the parallel and perpendicularly polarized intensities scattered by the sample are measured. Then the background intensity when no sample is present is measured and subtracted to remove stray illumination. Hereafter, $I_{\parallel}$ and $I_{\perp}$ refer to the intensities measured after stray light subtraction. Further, to correct for non-uniformities in the intensity and wavelength response of each pixel, $I_{\parallel}$ and $I_{\perp}$ are normalized by flat-fielding using a reflectance

standard. The corresponding (θ, φ) maps obtained with the reflectance standard are denoted $I_{\parallel}^{b}$ and $I_{\perp}^{b}$. From these maps, LSS intensity maps, $\Delta I = (I_{\parallel}/ I_{\parallel}^{b}) - (I_{\perp}/ I_{\perp}^{b})$ are calculated. These maps are formed by single scattered light and discriminated against multiple scattering [4].

The image file obtained from an LSS instrument shows scattering intensity (ΔI) as a function of wavelength (λ) and scattering angle (θ). It has been shown that the angular and wavelength distributions of light scattered by a cell nucleus depend on nuclear size and refractive index. The scattering angle range (-5º to +5º) is split into 1392 frames. For each frame, the scattering intensity value is measured as the wavelength is varied from 413.5 nm to 677 nm in steps of 0.253 nm. This results in an image with 1447680 data points (35 MB size). This LSS images can be analyzed in two ways:

- Wavelength dependence (By averaging across the scattering angle)
- Angular dependence (By averaging across the wavelength)

It has been shown that both angular dependent and wavelength dependent LSS images fit with Mie-theory. In this thesis, only wavelength dependent images are analyzed. The wavelength dependent image is obtained by averaging the spectrum across different scattering angles. The wavelength dependent image consists of only 1040 data points (21 KB size).

The LSS backscattered spectra is analyzed using Mie-theory of light scattering by spherical particles of arbitrary size, which enables the prediction of the major spectral variations of light scattered by cell nuclei [2]. There are, however, no known analytical closed-form general solutions for inverse Mie functions. Hence, analysis of LSS images relies on the use of Mie-theory to generate a database of LSS spectra over a representative range of mean diameters, standard deviations and average relative

refractive indices. For each pixel in the image, the size and refractive indices of the scatterers that best fit the data among those stored in the database are found using a least-square error minimization approach. Therefore, the algorithm for quantitative LSS analysis requires two inputs: the spectrum of backscattered light (obtained from an LSS instrument), and the lookup database (obtained from Mie-theory results across a desired range of diameters and refractive indices).

The following sections describe the sequential and parallel implementations of LSS algorithms for the generation of Mie-theory databases and analysis of LSS images.

2.3 Database Generation

2.3.1 Sequential Implementation of Database Generation

The LSS analysis relies on Mie-theory-based inversion to determine the spectral variations of the light scattered by cell nuclei. Closed-form solutions to this problem are not available; hence, the inversion procedure is implemented numerically through least-square minimization against a database of spectra generated using the Mie function. The Mie function takes 8 parameters as inputs:

- The minimum and maximum backscattering angle
- The azimuthal angle
- The minimum, maximum and step wavelengths
- The diameter of sphere
- Standard Deviation of diameter
- Type of Distribution
- Relative refractive index
- Refractive index of the medium

The diameter, diameter deviation and relative refractive index are of significant importance in the generation of the lookup database; values chosen for these parameters need to be derived from the expected ranges of sizes and refractive indices of cells under investigation. The remaining parameters correspond to values specified by the apparatus

used to obtain the backscattered spectral image. The Mie function returns the scattering intensity (ΔI) as a function of wavelength and scattering angle.

The program that generates LSS databases takes minimum, maximum and step values for diameter, diameter deviation and refractive index (Figure 2-4). It then generates a sequence of input files; for each input, it runs a separate executable that calculates Mie function spectra and writes results to the output files. The Mie function output file is averaged across scattering angles, normalized and appended as a record to the database file.

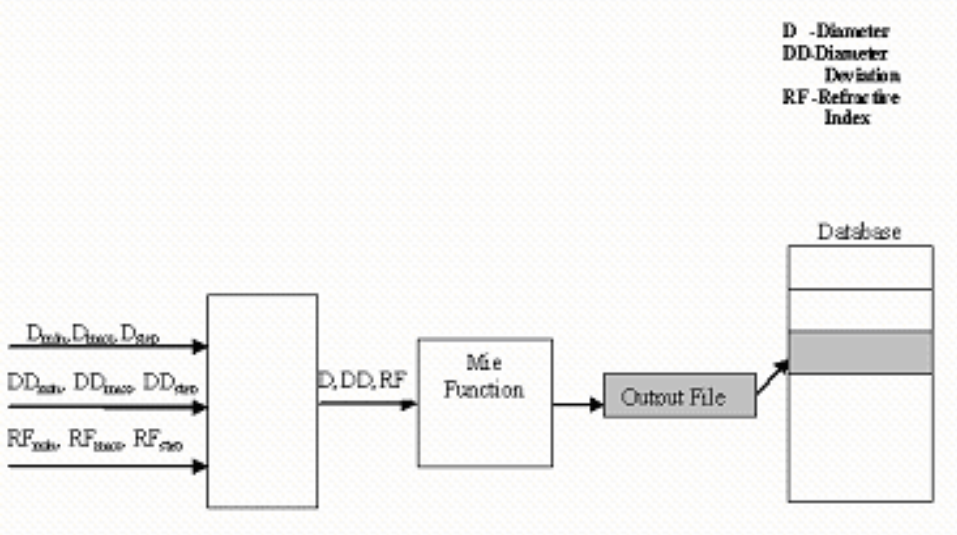

Figure 2-4. The LSS database generation. A range of diameters, diameter deviations and refractive indices are provided as inputs to a program that coordinates the execution of Mie function module and construct database records for each data point.

The format of the database file is shown in Figure 2-5. The header (first 7 lines) of the database describes the details of the database such as the number of records, the LSS instrument parameters used for this database, the range of wavelengths etc. Followed by the header are the original data points generated from Mie-theory for every combination of diameter, diameter deviation and refractive index values.

```
100                     #Represents the number of records in the database
400 700 1               #Represents the minimum, maximum and step wavelengths
-5 5                    #Represents the minimum and maximum scattering angle
0                       #Represents the azimuth angle
1                       #Represents the distribution
1.334                   #Represents the refractive index of the medium
2                       #Represents the width of data points

5.6 0.02 1.1            #Represents the diameter, diameter deviation and refractive
                              index of the first record

        400     0.57    #Represents the various data points for the record
        401     0.67
            ---
            ---
        550     1.00
            ---
            ---
        699     1.3
        700     0.8

5.6 0.02 1.2
    ----
    ----
    ----
    ----
6.0 0.02 1.1
6.0 0.02 1.2
        400     0.93
        401     1.56
            ---
            ---
        550     1.00
            ---
            ---
        699     1.38
        700      0.87
```

Figure 2-5. Sample format of the database file

The Mie function takes on average 20 seconds to compute one record in the database on a 2.4 GHz Pentium-4 machine with 1.5GB RAM. The sequential implementation of database generation might take time in the order of days to compute the databases that would be helpful in high accuracy analysis. Hence parallel implementation of database generation is considered.

2.3.2 Parallel Implementation of Database Generation

As each record is independent of each other, the databases can be generated in parallel (i.e., each process writes Mie-based records independently to a private file). The parallel implementation of the program (Figure 2-6) takes the number of processors as a parameter in addition to the above mentioned parameters (3 for diameter, 3 for diameter deviation and 3 for refractive index) required for sequential implementation.

Each processor generates a database file depending on the total number of processors and the rank of that processor. The program first calculates the number of records that can be generated with the different combinations of the diameter, diameter deviation and refractive index values. Then it checks if the total number of records can be distributed equally among the number of processors. If not, it decrements the number of processors in steps of 1 until the entire records can be partitioned equally. Then it checks if the entire records can be partitioned along a single direction. If not, it checks whether it can partition along any two directions. Once it partitions, depending on the rank of the processor, it allocates the parameters in such a way that no two processors generate the same record.

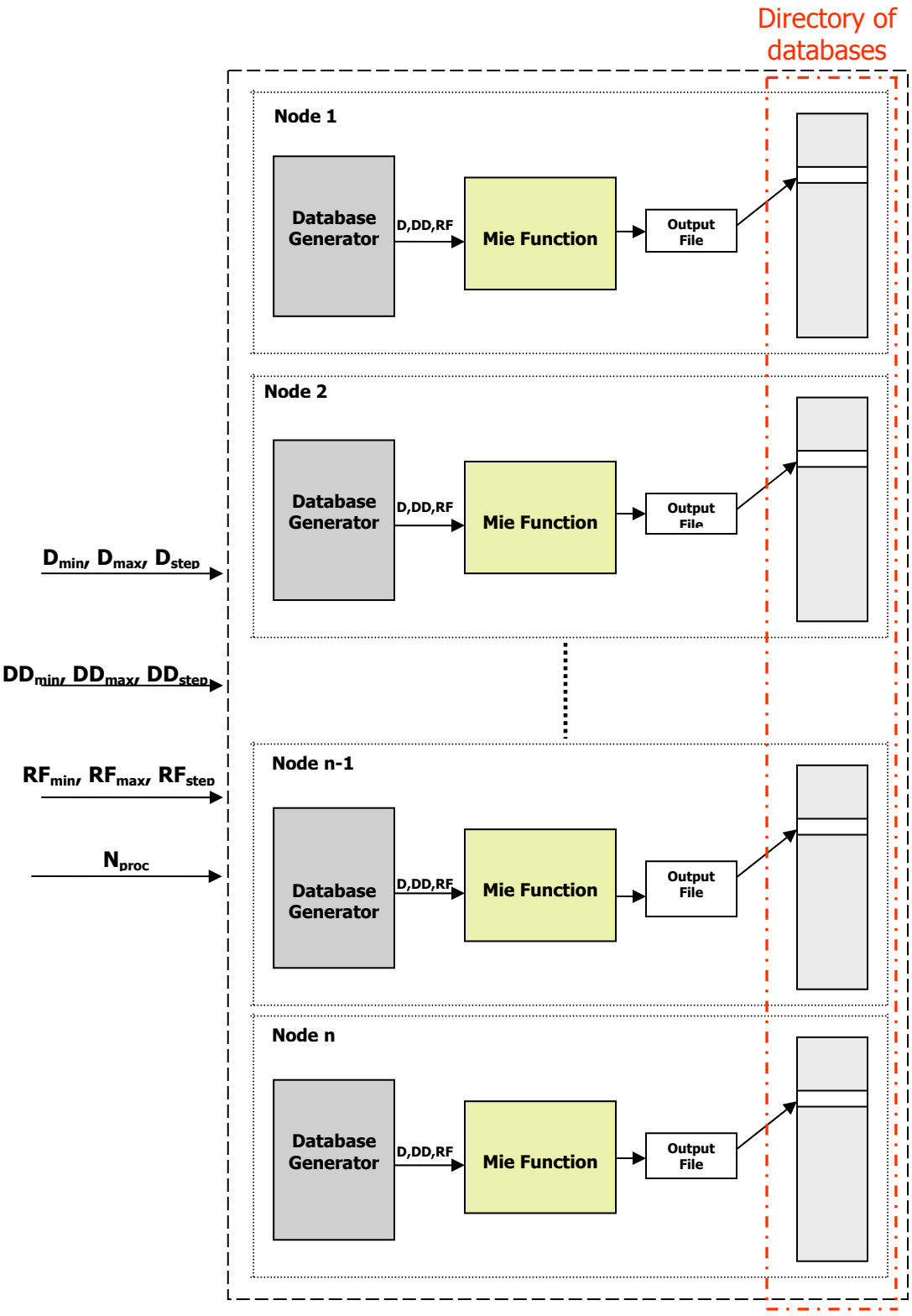

Figure 2-6. Parallel generation of LSS databases. The program distributes the input parameters to each processor depending upon the rank of the processor. Each process writes to an independent file.

The algorithm for the distribution of input parameters across different nodes is as follows:

Let D, DD, RF be arrays that contains the minimum, maximum and step values of diameters, diameter deviations and refractive indices respectively. Let Dcount, DDcount, RFcount be the total number of diameter, diameter deviation and refractive index points. For example, the total number of diameter points Dcount is (D[1] –D[0])/D[2] .The total number of database records (R) is Dcount * DDcount * RFcount. Let Nproc be the number of processors the database generation parameters have to be distributed. The program checks if the total number of records be distributed equally across the nodes by dividing R by Nproc. If it cannot, it decrements the Nproc by 1 and checks again, until it finds an Nproc where the parameters can be distributed equally.

Then, the program checks if it can distribute the parameters along any single direction by checking if any of Dcount, DDcount, RFcount is divisible by Nproc. If so, the array of the parameter (represented by A) is modified as follows:

```
A[3] = (Acount / Nproc) * A[2];
A[0] = A[0] + (rank-1) * A[3];
A[1] = A[0] + A[3] - A[2];
```

If the parameters can be divided across a single direction, the program checks if it can distribute along any two directions by checking if the product of any two of Dcount, DDcount, RFcount is divisible by Nproc. If so, the arrays of the corresponding parameters (represented by A and B) are modified as follows:

```
int i=1, j;
while(i <= Nproc)
{
    j = Nproc/i;
    if (Acount % i == 0 && Bcount % j == 0)
    {
```

```
            tempA = i;
            tempB = j;
            break;
        }
        i++;
    }

    A[3] = Acount/tempA * A[2];
    B[3] = Bcount/tempB * B[2];

    max = ((tempA > tempB) ? tempA : tempB);

    index = (rank - 1) % max;
    rank = (rank - 1) / max;

    A[0] = A[0] + index * A[3];
    A[1] = A[0] + A[3] - A[2];

    B[0] = B[0] + rank * B[3];
    B[1] = B[0] + B[3] - B[2];
```

Once the distribution is done, each processor will have independent set of input parameters, and will invoke the sequential database generation program and generate the databases in parallel.

2.4 Least-Square Analysis

2.4.1 Sequential Implementation of Least-Square Analysis

The LSS image spectrum is analyzed by performing lookups on the database of LSS spectra generated using Mie-theory. A program that performs the lookup and provides the best data fit using least-square error minimization approach has been developed. This program takes the database file name and the image file name as input arguments. The image files are obtained from instruments as described in Section 2.2 and Mie database files are obtained from Mie-based database generators as shown in Sections 2.3.1 and 2.3.2.

The flowchart for the program is shown in Figure 2-7. For each pixel in the image, the program calculates the error for each record in the database and returns the least square error and the corresponding diameter, diameter deviation and refractive index of the pixel.

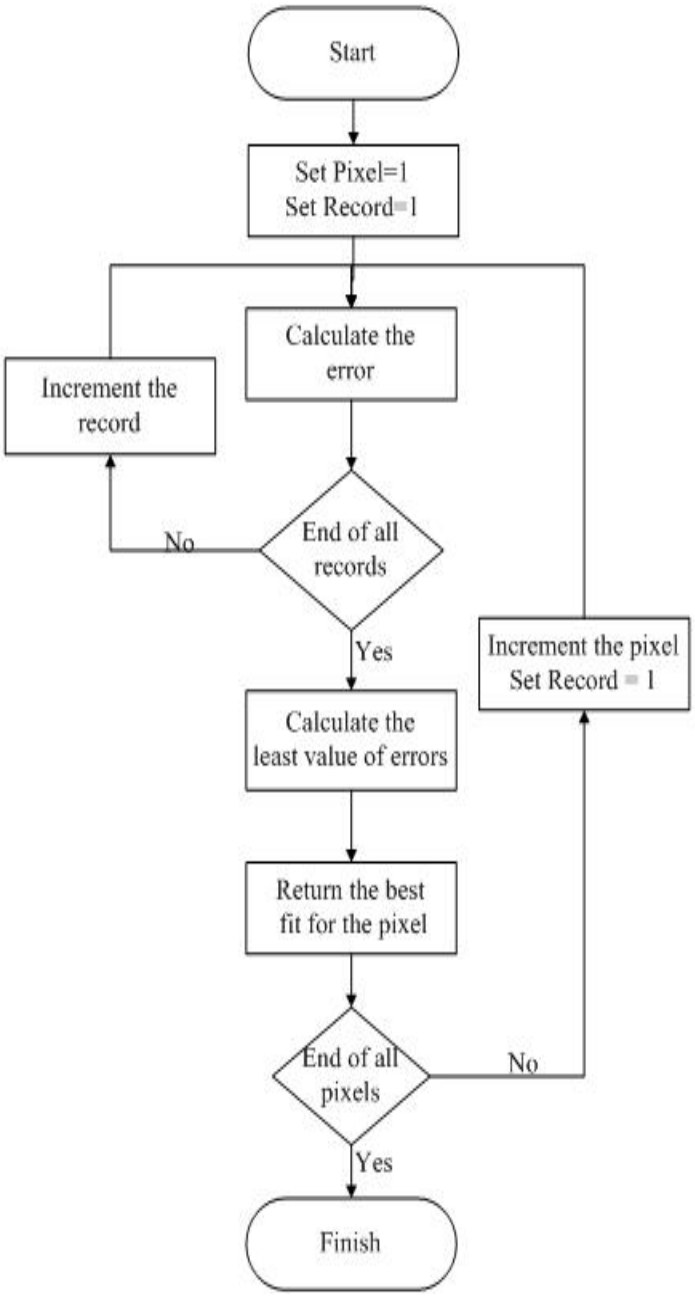

Figure 2-7. Flowchart for the sequential algorithm of LSS analysis. Each pixel is compared with each record of the database and the best fit parameters are returned.

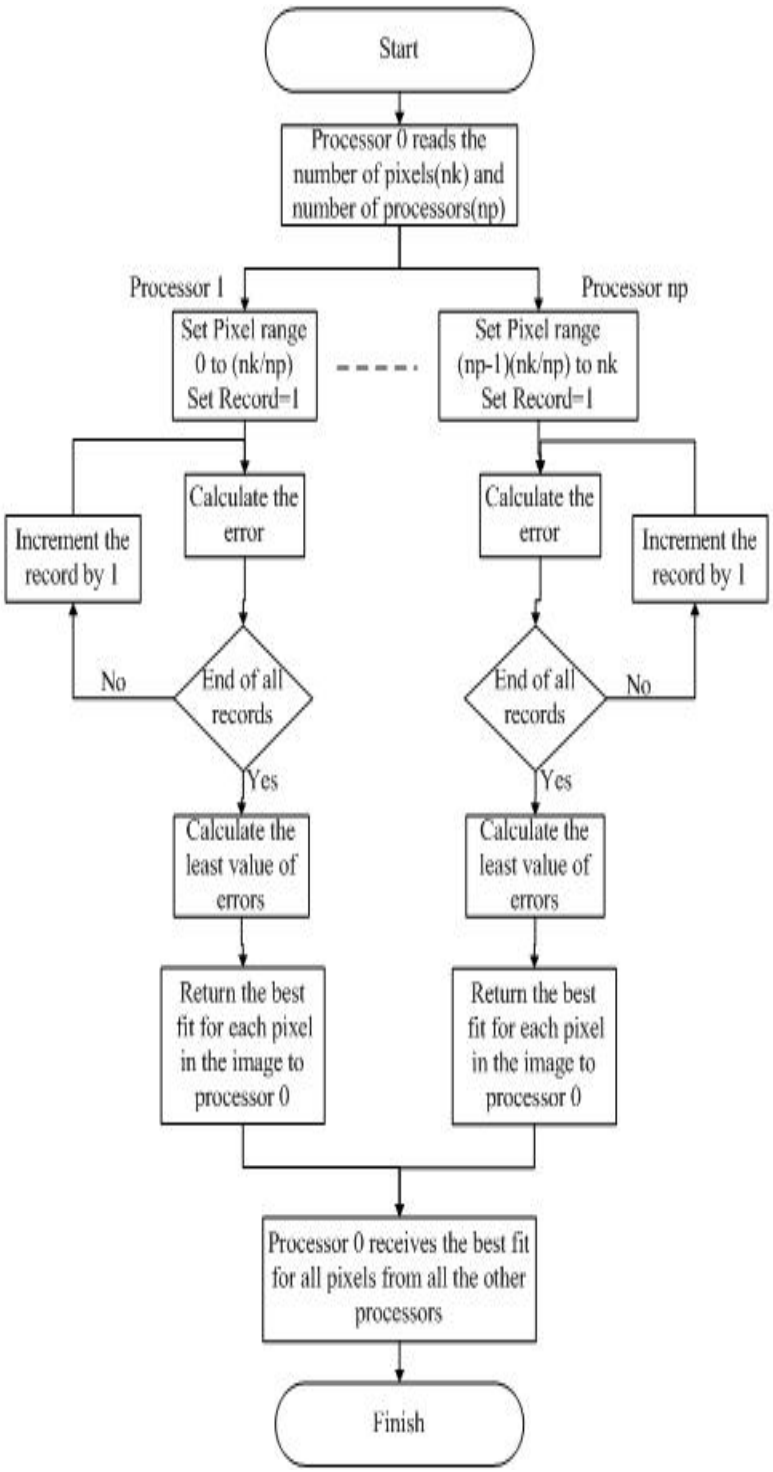

Figure 2-8. Flowchart for parallel analysis across different pixels. The master node distributes the pixels across all the processors. The slave processors compute the best fit for that pixel and return it to the master node.

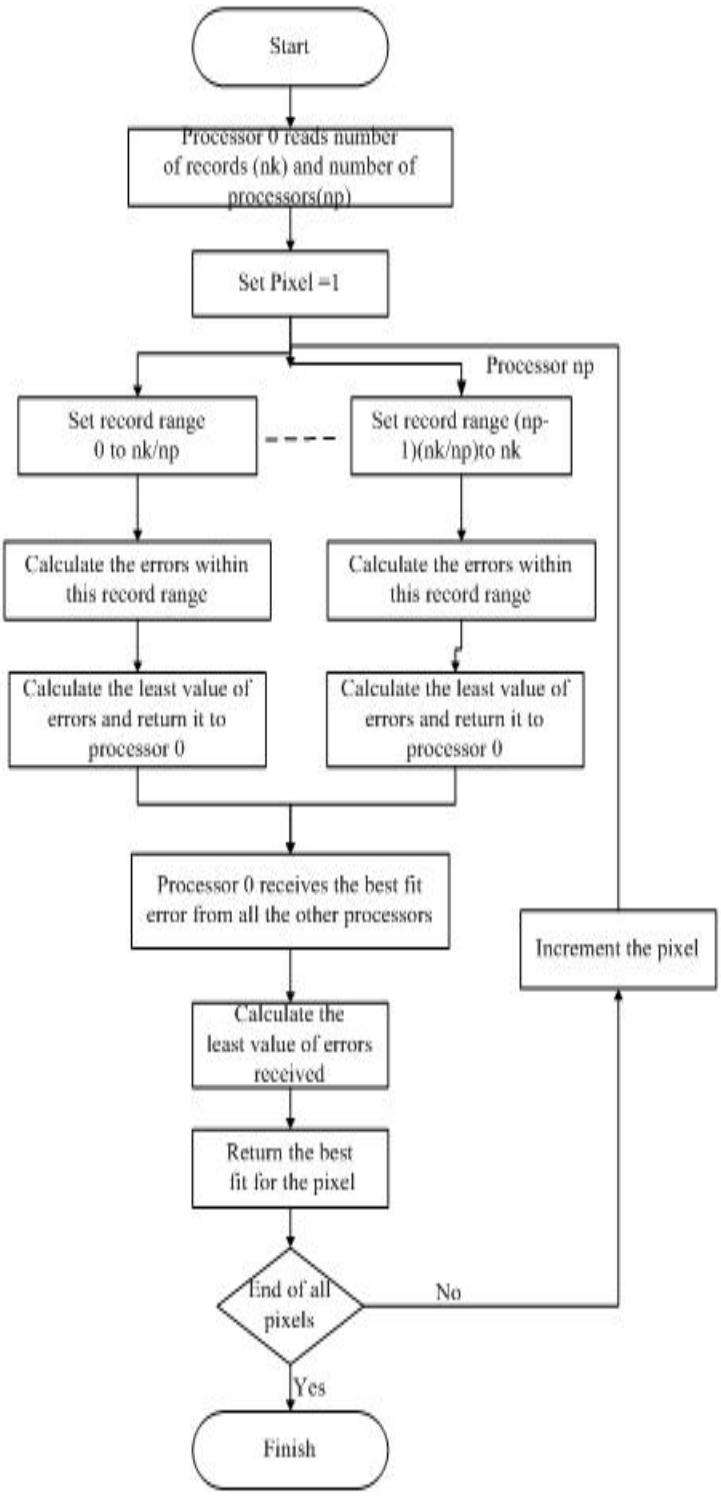

Figure 2-9. Flowchart for parallel analysis across different records in a database. The master node distributes the records across all the processors and the slave processors return their best fit. The master node in turn determines the global fit for each pixel in the image.

The analysis of large LSS images against large databases is a computationally- and data-intensive process. In order to allow spectral analysis with high performance, the best-fit analysis is implemented as a parallel program. It can be seen that the sequential program has two loops that can be parallelized as the body of each iteration is independent. Performance of LSS analysis can be improved by parallelizing across pixels (the best fit of a pixel can be calculated independently from other pixels) and across database entries (local minima across database sub-sets can be calculated in parallel; these can be reduced to a single minimum sequentially, or in a tree-like fashion) The parallelization can also be done across combinations of pixels and databases but this would result in a complicate design.

2.4.2 Parallel Implementation of Least-Square Analysis

Across pixels. For the pixel-based parallel implementation, the LSS analysis program takes a database file name and image file name as input arguments. The parallel algorithm uses the property that pixels can be independently processed. The process flow of this algorithm is shown in Figure 2-8. The task for each processor depends on the number of pixels and the number of processors. Each processor will calculate the outputs for a set of pixels given by (total number of pixels/number of processors). The pixels of the image are evenly distributed among all the processors. Each processor will return the best fit for the pixel by measuring the least-square error across the entire database.

As the image size is much smaller than the database size, it would be beneficient to parallelize across the database records rather than across pixels in the image. The database size is in the order of Terabytes. By splitting the database into small databases, the databases can be stored in cache memory and the LSS analysis can be performed at much faster rate. As the parallel implementation of database generation writes multiple

database files, the parallel implementation across different databases is considered instead of parallel implementation across different records in a single database.

Across databases. For the database-parallelized implementation, the program takes a directory name rather than a database file name as an argument. The directory contains multiple database files. The resulting program uses MPI for coordination and for determining the global minimum from independently computed local minima, while file I/O is used by each MPI process to independently access its databases. As the fit for each database can be processed independent of each other, the program is parallelized across the database files in the directory. A master-slave strategy as shown in Figure 2-10 is used to parallelize the program.

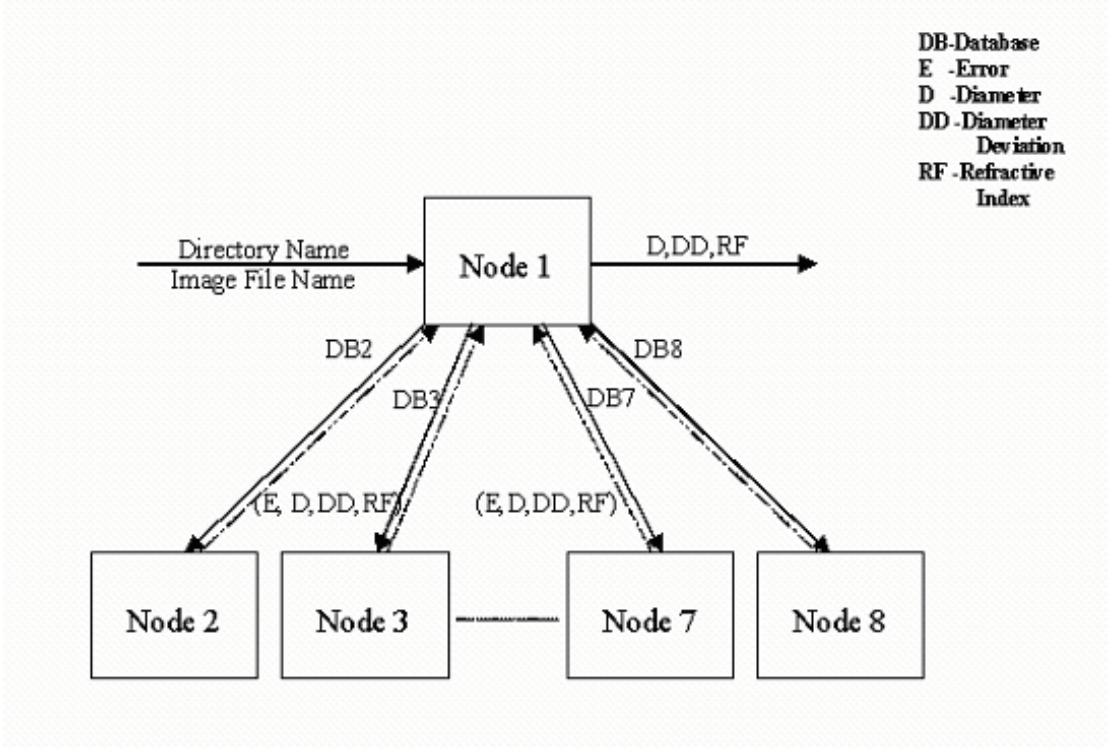

Figure 2-10. The LSS parallelization across database records. Node 1 is the master. Each processor accesses the databases depending upon its rank; the actual databases are accessed independently from each node's file system.

The master receives the input directory name and counts the number of database files in the directory, then assigns files to each processor to balance their load. Each processor calculates the fit for its own set of databases, given by number of files/number of processors. The processors send the local least square error and the corresponding diameter, diameter deviation and refractive index to the master processor in the form of an array. The master receives the local least square errors from each processor and in turn calculates the global least value among the errors received and finally returns the corresponding diameter, diameter deviation and the refractive index for the image.

The use of multiple independent lookup databases that are accessed through a conventional file system interface allows for 1) the partition of large datasets across multiple nodes, 2) the seamless execution of the program in conventional local-area and cluster-based environments used for MPI-based parallel executions, and 3) seamless integration with distributed Grid environments that are built on top of virtual file systems [11, 12]. These issues are addressed in Chapter 3.

CHAPTER 3
DATA MANAGEMENT SOLUTIONS

3.1 Computational View

As mentioned in Chapter 2, LSS spectral analysis relies on Mie-theory based inversion procedure. For the analysis to be highly accurate, databases of Mie-theory spectra have to be generated over many mean diameters, standard deviations and relative refractive indices. For example, a database considering a range of diameters from 0.1 um to 20 um in steps of 0.005 um, standard deviations from 0.1 um to 5 um in 0.005 um steps and average relative refractive indices from 1.02 to 1.1 in 0.0005 steps would result in a lookup table with 624 million records. The number of data points per record depends on the desired sensitivity and accuracy. Current LSS instruments generate a spectrum for wavelengths ranging from 410 nm to 680 nm. Hence to obtain high accuracy, Mie-theory spectra is considered over a wavelength range of 400 nm to 700 nm with steps of 1 nm. The storage requirements in this scenario demands TBytes of capacity for the lookup table.

The LSS spectral analysis compares each pixel of the image with each record of the database to obtain the best fit size and refractive index of the scatterers based on least-square approximation. The algorithm for the analysis is shown below. Let Np, Nr be the number of pixels in the image and the number of records in the database respectively. Let DPp, DPr be the number of data points per pixel and record respectively. The arrows in the algorithm show the number of operations needs to be computed.

```
For each pixel in image { ——————→ Np
        For each record in database { ——————→ Nr
                For each data point in pixel { ——————→ DPp
                        Sum = 0;
                        For each data point in record {——————→ DPr
                                If (record data point wavelength == pixel data
                                point wavelength) {
                        Sum = sum + sqr(pixel ΔI – Mie ΔI);
                                }                              ↘
                        }                                       5
                }
                error[record] = sqrt(sum);
        }
        return min(error);
}
```

The above algorithm shows that for each comparison of pixel with a record in database, 5*DPp*DPr operations are computed. If the number of data points per record is 300 and per pixel is 1040, each comparison would require 1.56×10^6 operations. Thus, to obtain high accuracy the total processing demands Peta-order number of operations. This level of accuracy demands computational requirements that are difficult to be met even with current high-end infrastructures. Hence parallel computing on cluster of workstations that has gained more attention in recent years is considered for implementing LSS analysis. High-speed general purpose networks and very powerful processors are reducing the performance gap between workstation clusters and supercomputers. Moreover, clusters can scale to large number of nodes and are less expensive than supercomputers. As the processors in workstation clusters do not share physical memory, interprocessor communication must be performed by passing messages among each other. Message-passing libraries such as MPI [9] and PVM [13] make the communication between processors possible. They provide functionality to easily pass

messages between processors. MPI is used as the basis for parallelizing the sequential programs in this thesis.

Message passing interface. Message Passing Interface is a communication library used in both parallel computers and workstation networks. It has been developed as a standard for message passing and related operations. The goal of the Message Passing Interface simply stated is to provide a widely used standard for writing message-passing programs. The interface attempts to establish a practical, portable, efficient, and flexible standard for message passing. It specifies the names, calling sequences, and results of subordinates to be called from FORTRAN programs, functions to be called from C programs, and the classes and methods that make up the C++ library.

In the message-passing model of parallel computation, the processes executing in parallel have separate address spaces. Communication occurs when a portion of one's process address space is copied into another process's address space. This operation is cooperative and occurs only when the first process executes a send operation and the second process executes a receive operation. The main advantages of establishing a message-passing standard are portability and ease-of-use.

3.2 Communication View

In current medical applications (e.g., computerized tomography) the physical area covered by the imaging device is large, thus requiring a large and expensive apparatus. Cost and area constraints limit the deployment of such devices to a few units in a medical facility. In addition, since the cost of the imaging device is high, a costly high-performance computing unit attached to the device is justifiable. In contrast, the physical area of tissue analyzed by an LSS imaging apparatus is typically small – of the order of square centimeters. LSS imaging can therefore be performed with smaller,

portable devices deployed in larger numbers across medical facilities. In this scenario, the use of a costly high-performance computer attached to the imaging devices is no longer attractive. However, it is important to perform high-performance computation to obtain a quantitative analysis of LSS images in quasi-real-time, allowing feedback to clinicians while the patient is under examination.

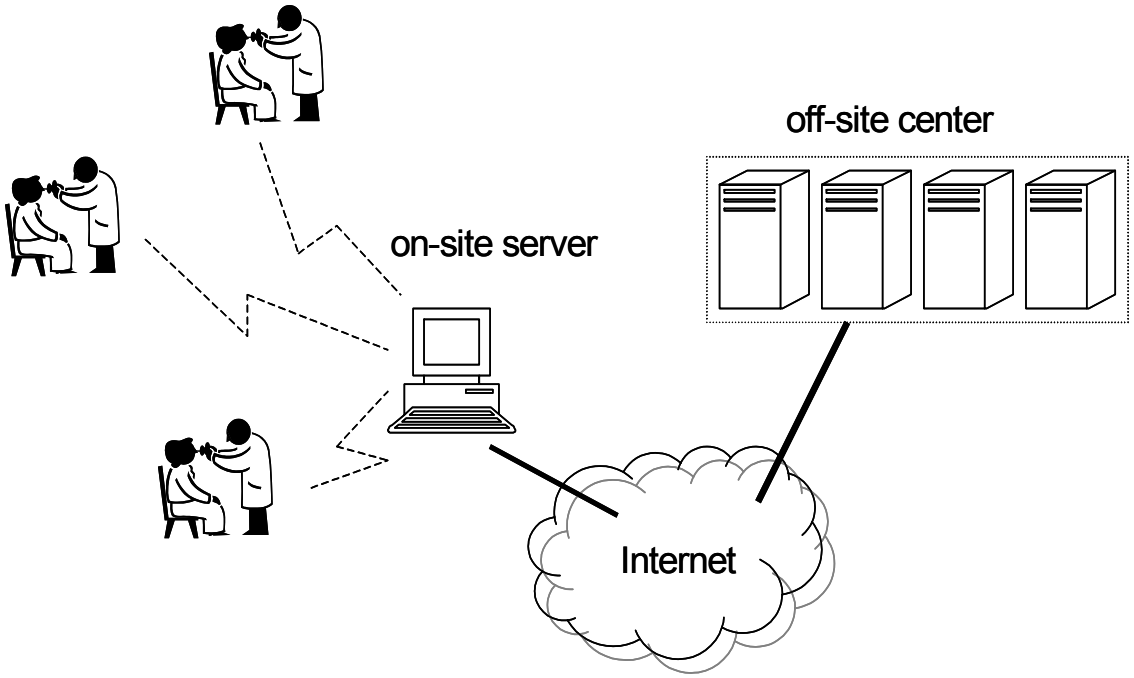

Figure 3-1. Environment envisioned for the deployment of in-vivo LSS imaging: multiple instrumentation devices connect to on-site computing services (e.g., via the wireless links represented by dashed lines), which have access to off-site computing centers.

Medical applications such as LSS analysis can benefit from an environment where a large number of network-enabled instrumentation systems are applied to data collection and visualization, while processing takes place in remote resources (Figure 3-1). There are several motivations for this solution: the hardware cost of a 'thin' device can be reduced because of its small processing requirements, and the corresponding software support can be performed at computing facilities (where extensive IT expertise is

available) rather than at a medical facility. Furthermore, improvements in the analysis software (algorithms and associated knowledge databases) can be deployed without requiring reconfiguration or replacement of instrumentation devices.

Under such network-computing model, the processing of data occurs on remote high-performance resources, while data is produced and displayed at the instrumentation device. A key issue that needs to be addressed by the network-computing system is the communication of data from/to instruments and processing nodes. In addition, the need for quasi real-time response times arises in this application domain: it is important for clinicians to obtain feedback from analysis results while the patients are under examination. This requirement, in turn, poses challenges to the design of data management solutions – they must allow communication to occur at high performance, possibly subject to high-latency and low-bandwidth constraints of networks available in medical environments.

In this environment, data associated with a patient needs to be transferred from a medical facility to an off-site computing center with guaranteed privacy. In addition, the least-square error minimization of an LSS image against a database of Mie-theory spectral fits demands high-bandwidth access to data that may reside in a data center not co-located with the computing center. These data management requirements are solved using virtual file system [10, 11] across different administrative domains. A virtual file system with support for cross-domain data transfers and application-transparent disk caching allows both the images and databases to be accessed with the intuitive abstraction of a conventional, local NFS file system, thus simplifying application development. LSS can also benefit from virtual file system caching, since the typical

access pattern to the database is read-only, and there is spatial/temporal locality of accesses when multiple images are minimized against the same database.

Virtual file system. Virtual file system provides file-system abstraction by considering a virtualization layer on top of NFS. It allows data to be transferred on-demand between storage and compute servers for the duration of a computing session [10]. This functionality is realized via user-level extensions to existing NFS implementations that allow reuse of unmodified clients and servers of conventional operating systems. Such implementations use middleware-controlled proxies to map identities between dynamically-allocated logical accounts and transparently broker a user's access to files across administrative domains. It leverages NFS implementations and does not require any modifications to either operating systems or applications: NFS client maps application system calls to RPC requests, and the NFS server maps RPC request to I/O accesses.

Secure RPC-based connections are established through the use of tunneling. A tunnel allows the encapsulation and encryption of datagrams at the client side, and corresponding decryption and de-capsulation at the remote site. It supports private communication channels in application-transparent manner. Tunneling of RPC-based connections is established through SSH, the de-facto standard for secure logins. Thus SSH tunnels allow privacy and session-key authentication without requiring modifications to underlying NFS implementations.

Furthermore, data transfer in the distributed virtual file system is on demand and transparent to the user. This behavior is inherited from the underlying NFS protocol, which allows for partial transfer of files on a block-by-block basis (typically 4K to

32Kbytes in size). This property is important when supporting applications that access large files, but not necessarily in their entirety. In addition to supporting on-demand transfers and dynamic identity mappings, middleware-controlled proxies support performance and functionality extensions (such as client-side caching and meta-data handling) that makes the virtual file system suitable for wide-area Grid applications.

Caching improves the performance of computer systems by exploiting temporal and spatial locality of references and providing high-bandwidth, low-latency access to the cached data [14]. Large read/write disk-based user-level caches can be employed on a per-application basis to complement typical kernel memory buffers: memory provides a small but fast first level cache while disk cache acts as a relative slower but larger second level cache. This 2-level memory hierarchy helps in the reduction of capacity and conflict misses. Disk caching, implemented by the file system proxy, eliminates the overhead of a network transaction. The disk cache holds frames in which data blocks and cache tags can be stored. The cache banks are created on the local disk by the proxy on demand. The hashing function is used to exploit spatial locality by mapping consecutive blocks of a file into consecutive sets of a cache bank. The proxy cache supports both write-through and write-back policies. These write policies can be determined also on a per-application basis. Write-back caching is important in wide-area environments to hide long write latencies.

Virtual file system addresses limitations of conventional approaches to data management in Grid environments which typically rely on mechanisms for data transfer that either requires explicit naming of files to be transferred by users and/or applications (GASS [15]) or special libraries linked to applications to support remote I/O. The

approach also addresses limitations of conventional distributed file systems: non-virtualized native NFS implementations that rely on single-domain, local-area network environments for authentication and user identification [16] are not well-suited for a cross-domain Grid deployment. These limitations are addressed via user-level, middleware controlled distributed file system proxies that intercept, modify and forward NFS remote procedure calls. File system proxies enable Grid-oriented extensions implemented completely at the user level, without requiring any kernel-level modifications to existing O/Ss; these extensions include Grid-controlled identity mapping for cross-domain authentication, locality enhancements via per-application disk-based caching policies, and data communication privacy via per-session encrypted channels.

3.3 Variable Granularity

Quasi real-time constraints also have an impact on the computational model that is specific to the analysis of LSS images. Highly-accurate LSS analyses are necessary in situations where precancerous lesions need to be inspected in detail, but can be prohibitively time-consuming in situations where a coarse-grain inspection of a large area of tissue is desired. These constraints have implications on the information processing requirements (data and computation) for this application.

The LSS application exhibits tradeoffs between computation time and accuracy: ideally, LSS imaging should be performed in quasi real-time to allow clinical feedback while patients are under examination; however, a detailed analysis may be too expensive – in terms of response time or the cost of utilizing remote resources – to be performed in all cases. A variable-grain solution that seamlessly supports multiple execution models – short response time (and possibly low accuracy), and high accuracy (at the expense of large response time) – is therefore desirable. The LSS application benefits from the

capability of operating on its data sets at different granularities – for example, by sampling down a dataset for a coarse-grain first-order analysis, and considering the entire dataset for high-resolution analyses.

CHAPTER 4
EXPERIMENTAL EVALUATION

4.1 Introduction

This chapter summarizes the results from experimental analyses and presents a comparative evaluation of the performance of the LSS algorithms discussed in Chapter 2. The chapter is divided into 3 parts. The first part describes the experimental results obtained for sequential and parallel implementations of LSS algorithms in a local disk. In the second part, the performance of the LSS algorithms in a distributed file system environment is evaluated. The final part describes the experimental results obtained for the variable granularity implementation.

4.2 Speedups on Local Disk

4.2.1 Experimental Setup

The experiments have been performed on a 32-node Linux-based cluster. Each physical node is configured as follows: 2.4 GHz Pentium-4, 1.5 GB RAM, 18 GB disk, Gbit/s Ethernet, Red Hat Linux 7.3. Experiments have been conducted in a virtual-machine based Grid [17] with VMware GSX 2.5 VMs (256MB memory, 4GB virtual disk, Red Hat Linux 7.3).

The implementation of MPI for the cluster is based on LAM/MPI 6.5.9. LAM allows processors to communicate information among them via explicit calls within C/C++/FORTRAN programs. The directory containing database files is mounted via a virtual file system [11] on all nodes.

A database of LSS spectra is generated over a range of diameters (5.65 um to 5.97 um in steps of 0.0005 um), diameter deviations (0.005 um to 2.5 um in steps of 0.005 um) and constant refractive index (0). This results in a database with 320000(640*500*1) records approximately. As discussed in Section 2.3, the Mie function takes on average 20 seconds to compute one record, thus the sequential generation of the complete database would take approximately 74 days. As each record can be generated independent of each other, the database is generated in parallel. The parallel generation of the database with 32 processors takes 3 days. Each process writes to a separate file resulting in 32 database files. The databases occupy 1.9 GB of disk space.

The LSS backscattered spectra of polystyrene beads (diameter = 5.8 um and diameter deviation = 0.02 um) suspended in water is used as the image for the experiments. The purpose of the beads is to test and ensure the proper calibration of the LSS instrument.

4.2.2 Results and Analysis

The execution time for LSS analysis against each of the 32 databases is, on average, 46.5 seconds. Table 4-1 shows the least-square error and the best fit parameters for each database. The sequential implementation for obtaining the best fit for the bead image across the 32 databases took 22 minutes. The best data fit for the experiment image is obtained at a diameter of 5.796 um, diameter deviation of 0.025 um and refractive index of 0. Figure 4-1 shows the image scattering intensity as a function of wavelength and the corresponding Mie-theory fit.

Table 4-1. Least-square error and the best fit parameters for each database

Database	Error	Parameters		
		Diameter	Diameter Deviation	Refractive Index
1	3.674617	5.6500	0.270	0
2	3.789568	5.6600	0.585	0
3	3.814236	5.6700	0.580	0
4	3.239030	5.6890	0.070	0
5	3.149745	5.6935	0.070	0
6	3.322689	5.7000	0.070	0
7	3.793431	5.7100	0.240	0
8	3.803646	5.7200	0.585	0
9	3.291548	5.7390	0.040	0
10	2.898698	5.7440	0.045	0
11	3.203210	5.7500	0.050	0
12	3.451358	5.7600	0.100	0
13	3.847614	5.7700	0.095	0
14	3.901837	5.7890	0.025	0
15	**2.898156**	**5.7960**	**0.025**	**0**
16	2.993846	5.8000	0.025	0
17	3.131888	5.8100	0.075	0
18	3.710292	5.8200	0.065	0
19	3.969656	5.8300	0.190	0
20	3.711047	5.8490	0.050	0
21	3.190115	5.8550	0.050	0
22	3.425422	5.8600	0.050	0
23	3.687394	5.8700	0.105	0
24	3.971831	5.8800	0.165	0
25	4.137124	5.8900	0.165	0
26	3.970392	5.9090	0.070	0
27	3.862937	5.9140	0.075	0
28	4.061786	5.9220	0.135	0
29	4.147652	5.9300	0.135	0
30	4.243860	5.9400	0.195	0
31	4.287716	5.9520	0.545	0
32	4.304501	5.9600	0.540	0

The average execution time for each database is 46.5 with a standard deviation of 0.496

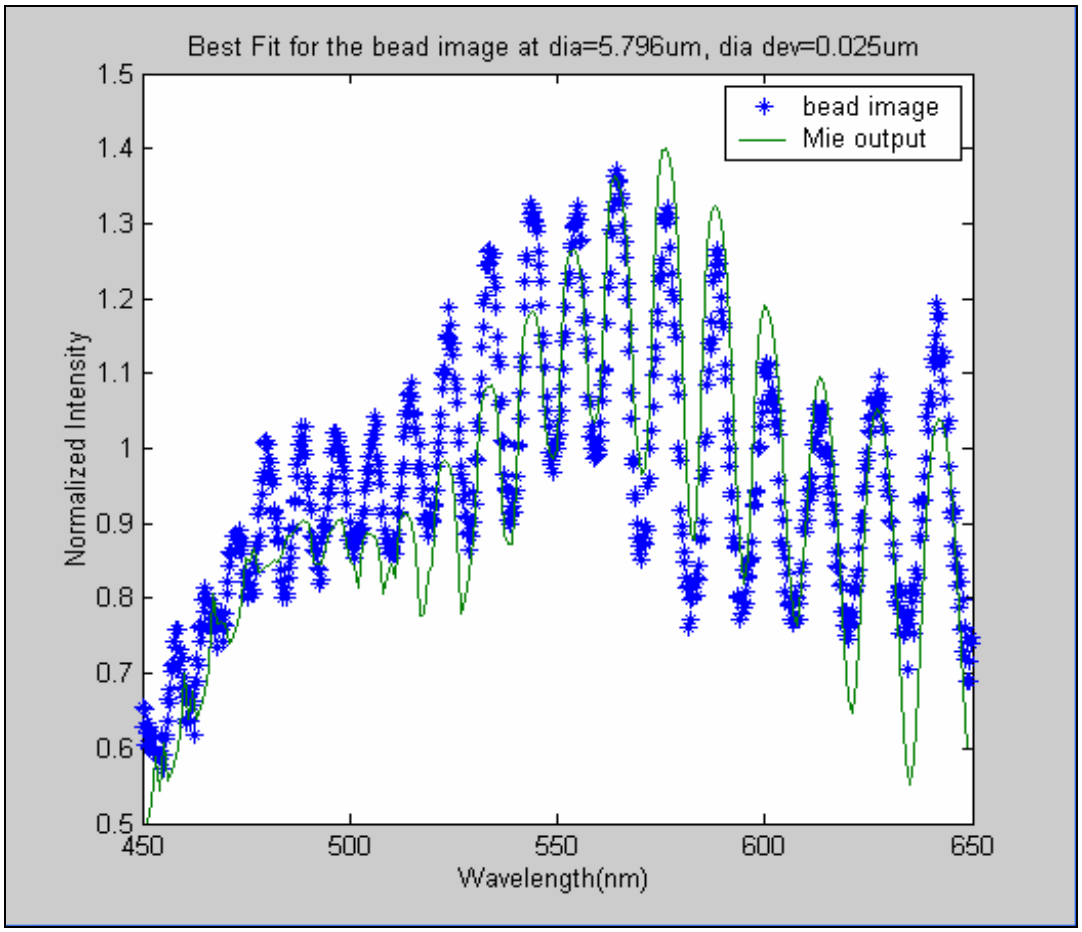

Figure 4-1. Spectral image obtained from polystyrene beads (diameter=5.8um, stdev = 0.02um) suspended in water, and least-square error fit (diameter =5.796um, stdev = 0.025um)

The above image has been analysed using the database-parallel algorithm on a 16 node cluster. The execution time to obtain the parameters is measured as a function of number of processors. Table 4-2 shows the execution times for the bead image. The results show that the parallel LSS analysis achieves speedups of up to 13.3.

Table 4-2. The LSS analysis execution time as a function of number of processors

No. of processors	Execution Time	Speed Up
1	1318	N/A
2	664	1.98
4	333	3.96
8	172	7.66
16	99	13.31

A database is generated with diameter ranging from 5.7 to 5.85 in steps of 0.005um, diameter deviation ranging from 0.005 to 0.085 in steps of 0.005um and constant refractive index (0) across different number of processors on a local disk. The time taken to generate database in each case is listed in Table 4-3. The results show that the speedup is almost linear as the parallel database generation is independent of each other. It can also be seen that speedups of 15.98 can be achieved with 16 processors.

Table 4-3. Time required for generating the database across different number of processors and corresponding speedup values

No. of processors	Execution Time	Speed Up
1	8839	N/A
2	4417	2.00
4	2212	3.996
8	1104	8.00
16	553	15.98

4.3 Speedups using Virtual File System

The data in this section is obtained by evaluating the performance of LSS algorithms for four different environments: local disk, LAN, WAN, WAN+C. With respect to the location of the databases, the following scenarios have been considered:

1. **Local:** The databases are stored in a local-disk file system.

2. **LAN:** The databases are stored in a directory NFS-mounted from a server in the local area network. File system proxies are used to forward RPC calls and do not have caching.

3. **WAN:** The databases are stored in a directory NFS-mounted from a server across the wide area network. File system proxies are used to forward RPC calls and do not have caching enabled.

4. **WAN+C:** The databases are stored in a directory NFS-mounted from a server across the wide area network. File systems proxies are used to forward RPC calls and support client-side disk caching.

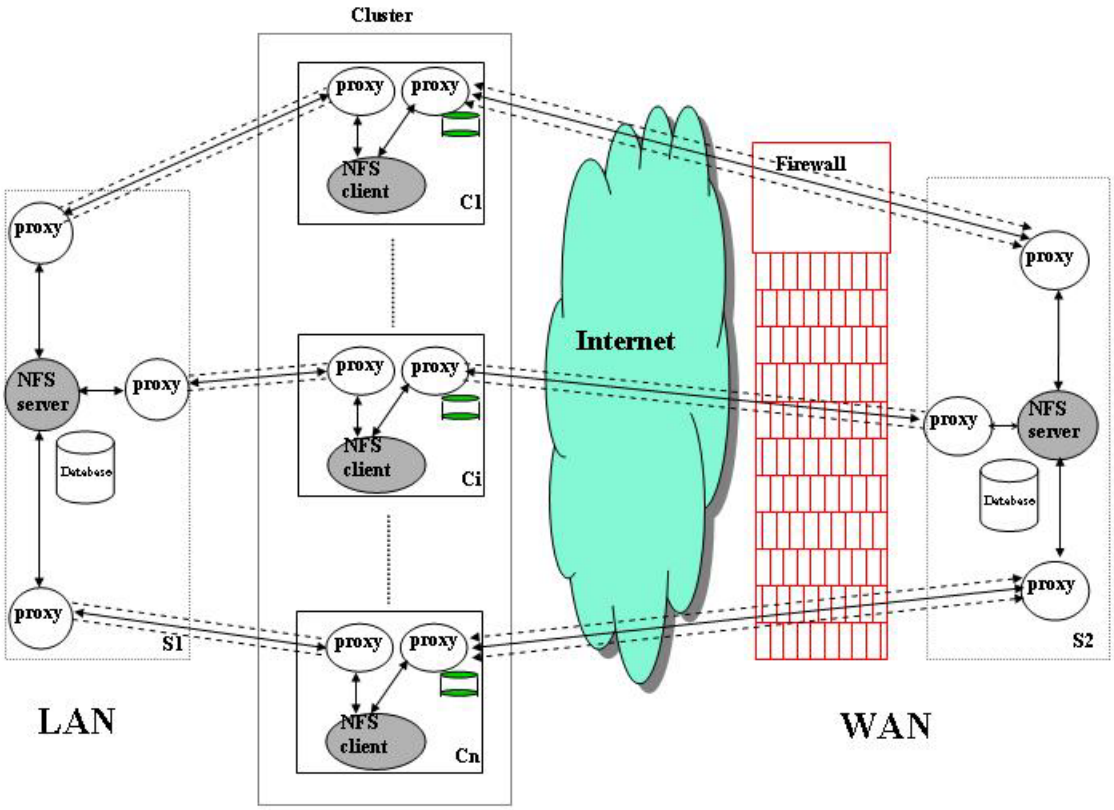

Figure 4-2. Experimental setup. The client cluster and server S1 are connected by a 100Mbps switched LAN at University of Florida. Server S2 is behind a firewall at NorthWestern University and connected by a WAN.

4.3.1 Experimental Setup

The experimental setup is illustrated in Figure 4-2. The LAN file server S1 is a dual Pentium-3 1.8GHz server with 1GB RAM and 8-disk/500GB SCSI RAID5 array. The WAN file server S2 is a dual Pentium-3 1GHz server with 1GB RAM and 46GB IDE RAID0 array. Experiments consider both local-area and wide-area virtual file systems. The WAN experiments are based on connections between University of Florida and Northwestern University through Abilene. The NFS traffic in both local area and wide area environments is tunneled through an SSH based private virtual file system. The proxy cache uses NFS version 2 with 8KB buffer size. The cache at the client side is configured with 1GB size, 512 file banks which are 16-way associative.

In the NFS-mounted cases, two scenarios are considered with respect to the state of the kernel client buffer memory cache: one where the file system is re-mounted (1st run), and one where the file system is not remounted (2nd run). The memory buffers are flushed by unmounting/remounting, and proxy caches are cleaned by removing cached data from the disk. All execution times cover the entire run of an application, and are measured at physical machines.

4.3.2 Results and Analysis

Least-square analysis. Table 4-4 shows the execution times for the parallel LSS application in different scenarios; Figure 4-3 shows the corresponding speedup plots. In the table, 1st run and 2nd run represent the first and second executions following an NFS mount. For WAN+C 2nd run, the proxy cache is "warm" in both mount and unmount configurations. It can be seen that the performance has improved as the number of processors is increased. The performance in the 2nd run is better because of caching of NFS blocks in the memory buffer cache. The advantage of client-side disk caching (WAN+C) in the presence of temporal locality becomes apparent when the number of processors is increased. The performance overhead of WAN+C with respect to local disk reduces significantly from 459% in case of a single processor to 12.5% with 16 processors. This can be explained by the increase in aggregate cache capacity stemming from the availability of independent proxy caches in each node. As the number of processors is increased, the working set size per each node is reduced and fits the proxy cache, resulting in high hit rate in the client-side disk cache.

Table 4-4. Execution times (seconds) for LSS analysis. Scenarios where databases are stored in local disk, LAN and WAN files servers are considered.

#Proc	Local Disk	LAN		WAN		WAN +C		
		1st run	2nd run	1st run	2nd run	1st run	2nd run	
							mount	unmount
1	1318	1404	1396	13473	11860	12465	7001	7369
2	664	735	718	5961	5883	5979	2204	2225
4	333	432	397	2992	2986	3044	674	1496
8	172	301	269	1993	1482	1580	228	317
16	99	234	203	817	755	804	111	183

The results summarized in Figure 4-3 support important conclusions. First, the hit-time overhead (with respect to local disk) introduced by the proxy-based virtual file system layer is small for the 16-processor case when the application exhibits temporal locality. Second, it can be observed from the performance difference between the WAN+C and WAN 2nd run scenarios that that the kernel-level buffer cache does not have sufficient capacity to hold the working dataset of an LSS database. The proxy-level disk cache behaves as a second-level cache to the kernel buffers; in fact, the 16-processor WAN+C scenario also achieves better performance than both LAN cases because kernel buffer misses are served by proxy disk cache accesses. It is important to point out that the design of the NFS call-forwarding file system proxy allows for a series of proxies, with independent caches of different sizes, to be cascaded between client and server, supporting scalability to a multi-level cache hierarchy (e.g., a two-level hierarchy with GBytes of cache storage space in a node's local disk, and TBytes of storage space available from a LAN disk array server).

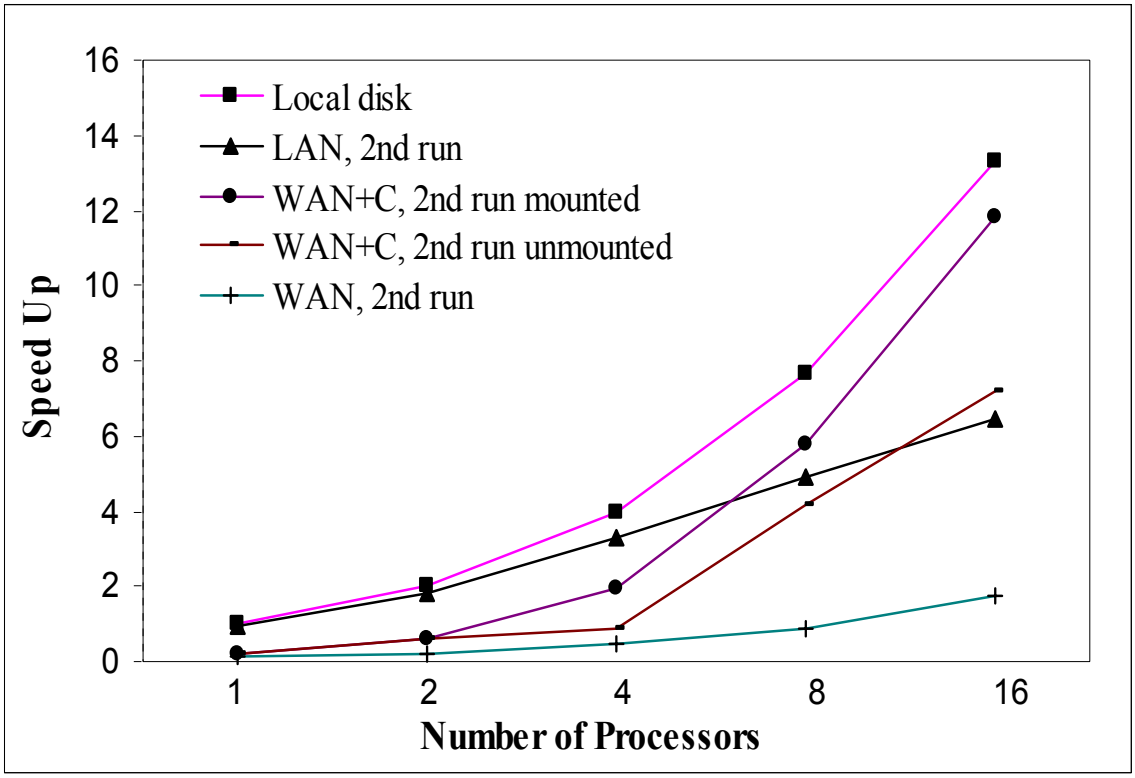

Figure 4-3. Speedup plot for parallel LSS analysis application

Table 4-5. Profile of the virtual file system proxy cache in the case of WAN+C scenario.

No. of Processors		Number of calls	Cache hits	Cache hit rate	Total bytes transferred per node (r/w)	Total time servicing requests (s)	Average bandwidth (B/s)
1	1st run	235136	511	0.002	1926098944	3.41E+03	5.65E+05
	2nd run	234624	111163	0.474	1921904640	2.84E+03	6.76E+05
2	1st run	117330	33	0	961101824	1.10E+03	8.73E+05
	2nd run	117283	84224	0.718	960716800	9.20E+02	1.04E+06
4	1st run	58656	28	0	480477184	9.80E+02	4.90E+05
	2nd run	58639	57440	0.98	480337920	1.50E+02	3.20E+06
8	1st run	29062	28	0.001	238059520	8.30E+02	2.87E+05
	2nd run	29062	29062	1	238059520	2.06E+01	1.15E+07
16	1st run	14682	24	0.002	120266752	7.05E+02	1.71E+05
	2nd run	14404	14404	1	117989376	3.20E+00	3.68E+07

Table 4-5 shows the profile for the virtual file system for the NFS read attribute collected in the case of WAN+C scenario. The table lists the number of NFS blocks being transferred, number of hits in the cache, total bytes transferred for different number of processors. The results show that the cache hit increases and reaches 1 as the number of processors is increased. The total bytes transferred per each node decreases as the data set is distributed across multiple nodes.

Database generation. Table 4-6 shows the time taken to generate the databases across different nodes in different scenarios; Figure 4-4 shows the corresponding speedups for the parallel database generation process. The database generation is a computational-intensive process that constantly exercises the file system – the legacy program that computes the Mie function reads from input files, and generates output files that are processed to generate each database entry. Therefore, this process generates many write requests that are subsequently invalidated by a file's removal. This experiment thus considers proxy-based configurations (WAN+C) that implement both a typical NFS write-through (WT) policy and an alternative write-back policy (WB) of blocks in the proxy disk cache.

Table 4-6. Execution times (seconds) for LSS database generation. Scenarios where databases are generated in local disk, LAN and WAN files servers are considered.

#Proc	Local Disk	LAN		WAN		WAN +C (WT)		WAN + C (WB)
		1st run	2nd run	1st run	2nd run	1st run	2nd run	1st run
1	8839	8988	8977	22914	22765	22935	22752	9016
2	4417	4491	4488	11693	11550	13002	11776	4493
4	2212	2253	2251	5910	5790	5971	5775	2249
8	1104	1134	1132	2954	2849	2894	2854	1137
16	553	583	576	1685	1595	1503	1445	570

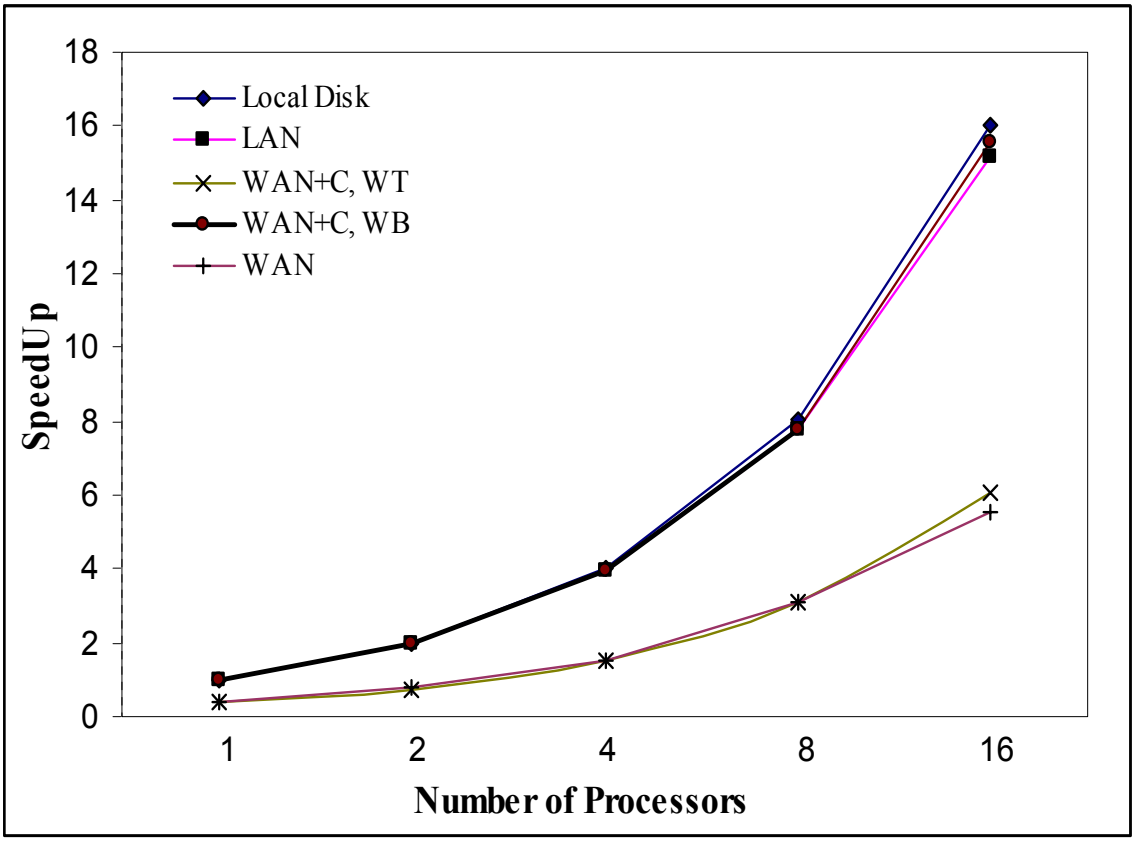

Figure 4-4. Speedup plot for parallel database generation. Only 1st run results are shown. The difference between 1st and 2nd runs is very small because data is mostly written.

The speedup plot shows that performance is close to linear with respect to number of processors for local-disk, LAN and WAN+C/WB. The performance gained from using a write-back cache policy in user-level proxies (as opposed to native, write-through schemes) is evident from the figure. The performance overhead in WAN+C scenario with local write back cache varies from 2% to 3% relative to the local disk configuration. The WAN+C write-back cache scheme performs slightly better than LAN scenario because it avoids network transfers for data that is written and then removed (such as the output Mie function files).

Table 4-7 shows the profile of the virtual file system proxy for NFS read and write attributes collected in the case of WAN+C scenario with Write-Back cache. The results

show that the cache hit rate is in the order of 97% for the NFS create call in the case of 16 processors. It can also be seen that the bandwidth offered by the cache is comparable to that of local disk access. The flush time of the Write-Back cache is 6546s with a single processor and in the order of 502s per processor in the case of 16 processors.

Table 4-7. Profile of the virtual file system proxy cache in the database generation process in the scenario of WAN+C with Write-Back cache

Number of processors	1	4	16
Read			
Number of calls	1391	381	212
Cache hits	1391	381	212
Cache hit rate	1	1	1
Total bytes transferred (r/w)	5697536	1560756	1347584
Total time servicing requests (s)	2.97E-01	9.13E-02	1.11E-03
Average time servicing a request (s)	2.14E-04	2.40E-04	3.23E-04
Average bandwidth (B/s)	1.92E+07	1.71E+07	1.52E+07
Write			
Number of calls	196843	50254	12082
Cache hits	196170	49858	11756
Cache hit rate	0.997	0.992	0.973
Total bytes transferred (r/w)	1589844558	405885129	97495879
Total time servicing requests (s)	5.32E+01	1.36E+01	1.75E+01
Average time servicing a request (s)	2.70E-04	2.70E-04	1.45E-04
Average bandwidth (B/s)	2.99E+07	2.99E+07	2.58E+07

4.4 Variable Granularity

Tradeoffs in accuracy and speed of LSS image analysis come about when the pixel resolution and the configuration of look-up databases are chosen. Low accuracy analysis of LSS images can be performed by generating the databases using larger steps between discrete parameters or by sampling down the databases with smaller steps. Low accuracy analysis helps in reducing the information processing requirements (data and computation) for the application. Low accuracy analysis can be performed earlier across a wide range of databases to obtain the close fit, and then high accuracy analysis can be

done by considering the complete databases around the close fit. A program is designed that performs low accuracy analysis for the image by sampling down the databases. The implementation takes the sampling interval along with the general parameters required for least-square analysis of LSS images. A sampling interval of "n" indicates that "n" records are skipped before reading another record in the database. Table 4-8 shows experimental results that consider variable-granularity executions of the LSS application.

Table 4-8. Error, WAN execution time and number of NFS data blocks transfers for database sampling

Sampling Interval	LSS Error	Time (s)	Number of Blocks	Number of Bytes Transferred	Average Bandwidth
1	2.899	793	14666	120143872	1.75E+05
5	2.9	700	14662	120094720	1.77E+05
10	2.902	432	6894	56434688	1.77E+05
20	2.916	323	3622	29614080	9.68E+04
40	2.934	152	1856	15147008	1.07E+05

The results shown in the table are based on the executions performed on 16 nodes in the WAN configuration. It can be seen that the least-square error has increased and the execution time has decreased as the sampling interval is increased. The 4th column in the table indicates the number of NFS blocks being transferred from the file server. It can be seen that the virtual file system transfers the data partially on demand for the low-accuracy case, reducing the overall transfer size by a factor of 8 and execution time by a factor of 5.2 (with respect to whole-file transfer). The reduction in the number of blocks transferred does not follow a linear relationship with respect to the sampling interval due to NFS client-side read-ahead (prefetching) implemented in the kernel. Nonetheless, the reduction in transfer requirements is substantial, and is handled by the virtual file system in an application-transparent way.

CHAPTER 5
INTEGRATION WITH GRID ENVIRONMENTS

This chapter describes the integration of the LSS application with In-VIGO. In Section 5.1, the necessity of integrating LSS application with computational grid framework is explained with the help of an example. The architecture of In-VIGO and the integration of LSS application with In-VIGO are described in Section 5.2. The performance for parallel database generation in computational grid environments like In-VIGO is analyzed in Section 5.3.

5.1 Framework for Distributed Processing of LSS Images

As discussed in section 2.4, the existing algorithm for the spectral analysis of LSS images relies on the Mie-theory-based inversion procedure to predict major spectral variations of light scattered by cell nuclei. Closed-form analytical solutions to this inverse problem are not known; existing algorithms obtain approximate solutions by performing look-ups on a database of LSS spectra over a range of mean diameters and determining the best-case fit. Tradeoffs in accuracy and speed of LSS image processing come about when the pixel resolution and the configuration of look-up databases are chosen. LSS imaging techniques analyze three-dimensional input data sets (two surface coordinates, and frequency). In a scenario where a 1.3x1.3cm image is processed with very high accuracy, 25x25μm (512x512) pixels containing 40 spectral data points from 20 different frequencies need to be processed, and, for each pixel, 640 million-entry look-up table (combinations of 4K diameters and 1K standard deviations, 160 refractive indices) needs to be searched. The total processing in this scenario demands a Peta-order number of

operations, MBytes of storage for each image and GBytes of storage for the look-up table.

5.2 Integration with Web-based Grid Portal

A user interface for the LSS application has been developed in In-VIGO, an infrastructure that supports the on-demand creation and management of virtualized end-resources for grid computing. In-VIGO is designed to support computational tools for engineering and science research [12]. In-VIGO is built on virtualization technologies that have three fundamental capabilities: polymorphism, manifolding and multiplexing.

The integration of LSS application with In-VIGO is achieved via one of such virtual technology known as virtual application (VAP). A virtual application is different from the actual application, such that multiple executions can be run simultaneously over the same physical application. The VAP specifies the interface exported to In-VIGO users (eg. a Web-based portal as depicted in Figure 5-1).

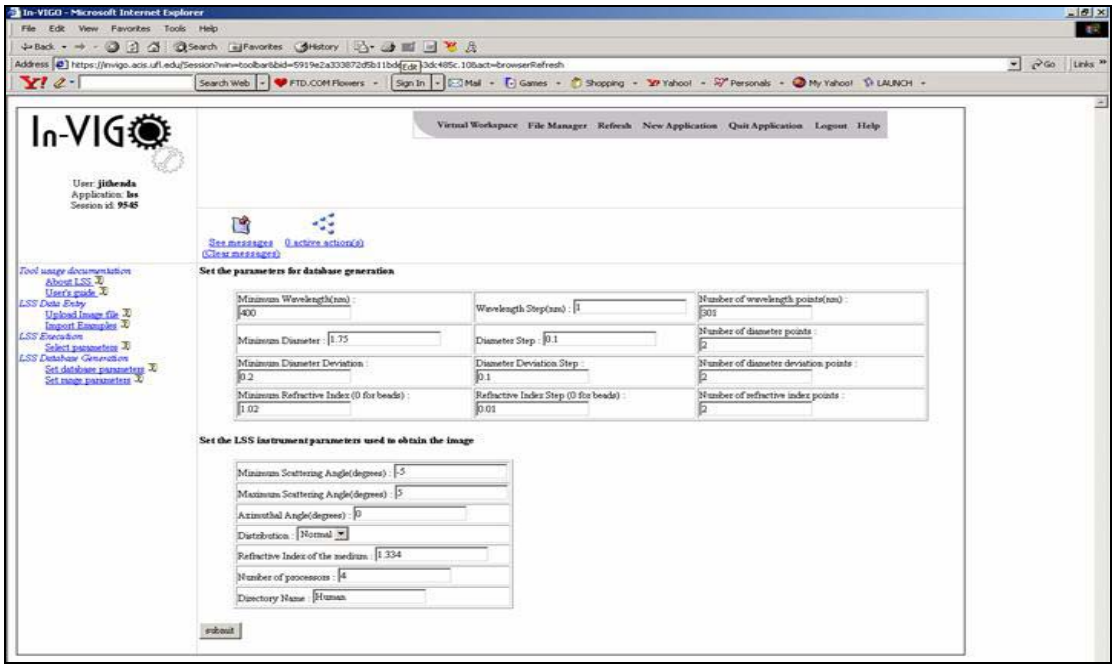

Figure 5-1. Snapshot of LSS application in In-VIGO

The virtual file system is implemented via extensions to existing NFS implementations that allow reuse of unmodified clients and servers of conventional operating systems. The modifications are encapsulated in a software proxy that is configured and controlled by the grid middleware. The virtual file system discussed in Section 3.2 establishes a dynamic mapping between users' working directories and their home directories on the data server.

The integration of LSS application in In-VIGO requires creation of a configuration file and Java classes (known as rules) to control the application behavior. The configuration file contains the tags which determine the user interface in the browser. The In-VIGO rules are associated with actions initiated by users (e.g., upload files, execute application). The configuration file is in XML format and the In-VIGO rules are written in Java. A snippet of the configuration file is shown in Appendix A. The configuration file contains the links to the application documentation and certain actions, such as upload files and set parameters, through which a user can control and interact with the application. It also contains rules tag which defines the implementation of the action. The configuration file is described by a Documentation Type Definition (DTD). In-VIGO rules represent the control and resource view of the virtual application in In-VIGO. That is, they specify the resource requirements (operating system, architecture), prerequisites of the execution (directory structure, files) and the execution logic corresponding to the user's input. The In-VIGO rule to execute the LSS application in In-VIGO is shown in Appendix B. In-VIGO supports different rules such as:

- LoadFileRule – used to load files to the user session
- ImportFileRule – used to import the example files to the user session
- DataEntryRule – used to set parameters for the application
- ProcessRule – used to run the application

In-VIGO supports ranging. Ranging in In-VIGO is a simple implementation of parameter sweeping. The parameter that needs to be ranged has three values: Begin, End and Func. The VAP generates all the values from Begin to End in steps of Func, unfolds the value into the parameter and executes the In-VIGO rule as many times as the number of values in the range. In this way multiple executions of the same executable are performed with different parameters.

The following facilities have been provided for the LSS application in In-VIGO

- **Upload Files** – Users can upload the image file from their computer. This facility is supported by LoadFileRule.
- **Import Examples** – Users can import the sample database directory and image files into their working directory from the repository. This facility is supported by ImportFileRule.
- **Generate Databases** – User can generate his own database files in parallel across different number of processors. A form is provided for the user to select the range of diameters, diameter deviations, refractive indices and wavelengths that are required for the database generation. The form also contains the input parameters specific to an LSS instrument such as scattering angle, azimuthal angle etc. The databases are written into a directory specified by the user. The parallel generation of databases discussed in Section 2.3.2 is achieved through ranging in In-VIGO.
- **Execute LSS** – User can perform analysis of LSS images against a directory of databases. The user has been provided with a form to select the database directory, the image file and the range of wavelength in which LSS analysis has to be performed. The user can obtain the best fit parameters for the image and a figure that shows the best fit.

The above facilities have enabled the user to execute the application in two ways

- Upload the image file and obtain the best fit parameters by performing LSS analysis against the example databases.
- Upload the image file, generate the database files and obtain the best fit parameters by performing LSS analysis against the generated databases.

Currently the In-VIGO prototype is not integrated with MPI based parallel environments; hence the LSS application has been integrated only in its sequential form.

Efforts will be made in the near future to run In-VIGO on a cluster where we can specify the number of processors and execute the parallel version of LSS program too.

5.3 Performance Analysis

A database is generated with diameter ranging from 5.7 um to 5.85 um in steps of 0.005 um, diameter deviation ranging from 0.005 um to 0.085 um in steps of 0.005 um and constant refractive index (0) using ranging in In-VIGO. The generation of the database in a single node took 9553 seconds (averaged over 3 executions) and when ranged across 8 nodes, it took 1296 seconds (again averaged over 3 executions) giving a speedup of 7.37. The decrease in speed up as compared to Table 4-3 can be attributed to the overhead involved in automatic creation of file system sessions and the resource management in In-VIGO.

CHAPTER 6
RELATED WORK

6.1 Summary

Gregor von Laszewski et al. [6] have demonstrated a grid-enabled real-time analysis, visualization, and steering environment for micro tomography experiments and developed a portable parallel framework that supports on-line three-dimensional tomography image reconstruction - and subsequent collaborative analysis - of data from remote scientific instruments. However, the visualization is built on hardware-optimized libraries.

Grid Enabled Medical Simulation Services (GEMSS) [18] present a Grid middleware which provides grid services for medical applications based on Grid technology standards. GEMSS mainly focuses on the computational services for the applications rather than accesses to data over distributed systems.

A remote Access Medical Imaging System (ARAMIS) [19] provides an object-based user interface with a transparent access to remote sources. However, the system is based on input parallel libraries making it application specific. The implementation of ARAMIS is similar to the network model deployed for LSS. The system propose two levels of network: high-speed, fast-access network to support transport of large volumes of data (between databases and servers) and a low bandwidth network for transport between the servers to user's workstation.

Grid data management has been investigated in previous efforts in the context of distributed instrumentation [20-22]. However, proposed techniques have focused on a

model where support for communication is explicitly provided by the application and/or grid middleware – typically, the data is "staged" from instrument to a remote computing node (and back). In addition, existing techniques are geared towards cases where substantial computational infrastructure (hardware and network capacity, and software expertise) is available at the site where data collection is performed (e.g., a national research center). In contrast, medical applications such as LSS imaging benefit from a different model, where support for communication is handled transparently from applications, hence reducing programming complexity, and where the computational infrastructure support consists of commodity computers and networks typical of a medical facility.

Current grid solutions typically employ file staging techniques to transfer files between user accounts in the absence of a common file system [15]. As indicated earlier, file staging approaches require the user to explicitly specify the files that need to be transferred, or transfer entire files at the time they are opened. This poses additional application programming challenges (the programmer must explicitly identify all data that may be necessary to perform computation so that it can be transferred prior to execution) and may lead to unnecessary data transfers (e.g., data needed for high-accuracy analysis that is not used in a low-accuracy computation). These requirements hinder the deployment of solutions that can dynamically adapt computation based on run-time requirements (since the choice of the working data set is statically determined before execution). In contrast, the architecture based on a virtual file system allows for transfers of data on-demand, and on a per-block basis. Hence, the amount of data transferred is determined by the amount of data actually used in computation, and

decisions regarding the data used in computation can be efficiently performed at run-time. This is important in applications such as LSS, where the size of working sets used in computation can vary dynamically based on accuracy requirements.

Kangaroo [23] utilizes remote I/O mechanisms from special libraries to allow applications to access remote files. It also employs RPC-based agents. However, unlike VFS, Kangaroo does not provide full support for the file system semantics commonly offered by existing NFS/UNIX deployments (e.g., delete and link operations).

Legion [24, 25] employs a modified NFS daemon to provide a virtual file system. From an implementation standpoint, this approach is less appealing than NFS call forwarding: the NFS server is customized, and must be extensively tested for compliance, performance and reliability. In contrast, VFS works with unmodified application binaries and native O/S clients/servers.

CHAPTER 7
CONCLUSIONS

In this final chapter, we summarize the work presented throughout this thesis and propose some ideas for future work.

7.1 Summary

This thesis presents a case study for the use of Grid environments for biomedical applications. A nascent application from the medical imaging, Light Scattering Spectroscopy, is used as a basis for this study. The computational requirements of the problem are met by parallelizing and distributing the computation across multiple nodes. The communication requirements are solved by the use of virtual file system.

In terms of application development, the virtual file system allowed for the rapid design and deployment of the LSS code. Since all inputs and outputs were treated as regular files, the virtual file system allowed the same code to seamlessly migrate from a local-area to a wide-area environment without modifications. The file system approach to data management supports on-demand transfers at the O/S layer, requiring no application modifications. Such support is especially important for applications tat access data at different granularities. The performance of the virtual file system can be improved by using user-level disk caches and write policies.

The database generation code is a legacy sequential Mie-function generation code, and the LSS analysis code has an MPI component that is responsible solely for synchronization and gathering of least-square errors reported by each process. For the latter, an alternative implementation using MPI file-I/O support (e.g., ROMIO) could be

conceived with extra programming effort; however, such a solution might be difficult to deploy across administrative domains, since it assumes a shared file system across compute nodes. The LSS application uses the shared virtual file system across different nodes to access the database files rather than partitioning the database for sending it to all nodes. In contrast, database partitioning via a message-passing interface could complicate programming and might also increase the communication cost when large datasets are multi-cast from master to slave nodes. Although performance comparisons have not been done, we believe the virtual file system approach can provide superior performance, benefiting from the user-level disk caches that are persistent across LSS executions, especially in a wide-area environment.

The automatic creation and destruction of virtual file system sessions is also a key issue. In the context of PUNCH [11] and In-VIGO [12], middleware allow for the setup of virtual file system sessions on a per-user, per-application basis. The setup starts with the execution of server-side proxies for NFS; once the proxies are running, the sessions are established by a client-side mount operations initiated by the middleware. In In-VIGO, this setup takes a few seconds to complete and is done in an application-transparent manner, hiding the complexity of setting up the file system from application writers and end users of a problem solving distributed environment.

7.2 Future Work

This section discusses some of the possibilities of future research that may be applied to the work presented in this thesis:

1. The present implementation of the LSS analysis determines the best fit based on least-square error minimization. Curve fitting techniques can be used to determine the best fit more efficiently.

2. Brute force algorithm is used to compare each record of the database with each pixel in the image to obtain the best fit. The performance can be improved by using more optimized and efficient algorithms.

3. The present deployment of LSS application in In-VIGO only supports the sequential implementation of LSS analysis. Efforts would be made in the future to integrate the parallel version too.

APPENDIX A
CONFIGURATION FILE FOR LSS APPLICATION IN IN-VIGO

The configuration file snippet for the LSS application is shown below:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configurationFile SYSTEM
"http://www.acis.ufl.edu/~acissoft/netcare/configuration_file.dtd">

<configurationFile>
<credits>

<![CDATA[
    <p>Description of the application </p>
  ]]>
</credits>

<permissionGroups>
  <permGroup name="Run">
         Permission groups allow the users to be divided into groups.
  </permGroup>
</permissionGroups>

<values>
  <value name="Distribution">Used for lookup list. </value>
</values>

<groups>
   <group name="MieParameters" columns="3" rows="4">
        Used for grouping the parameters into a table.
   </group>
</groups>

<categories>
   <category name="00Documentation">
        Specifies the category for the actions.
   </category>
</categories>

<actions>
  <action name="RunLSS">
     <actionText>Text to be displayed on the browser</actionText>
```

```
    <actionCategory>Selecting the category</actionCategory>
    <actionDescription>Brief description of the action</actionDescription>
    <actionType>special/action</actionType>
    <actionHelp>Brief description</actionHelp>
    <actionPredicate> The action would appear only if the predicates evaluate
        to true.</actionPredicate>
  <actionAction>Corresponding Rule</actionAction>
  </action>
</actions>

<rules>
   <loadFileRule name="LoadDatabasefileRule">
    <fileSpecs name="DatabaseFile">
    <fileName>Specify the file name, if restricted.</fileName>
    <fileDirectory>The directory where the file would be uploade </fileDirectory>
    <fileLabel>Label for the upload button</fileLabel>
    <fileDescription>Description of the action </fileDescription>
    </fileSpecs>
   </loadFileRule>

   <dataEntryRule name="DataEntryRule">
    <dataElement name="Directory">
       <dataDefault> Select default value, if any</dataDefault>
       <dataLabel>Select database directory </dataLabel>
       <dataType>Type of the data element</dataType>
       <dataValidationPredicate>True</dataValidationPredicate>
       <dataValidationErrorText>Invalid Entry</dataValidationErrorText>
            <dataListPredicate>getFileList(System_workingDir,"Dir_*")
                    </dataListPredicate>
    </dataElement>
    <validationRule>
          <validationPredicate>True</validationPredicate>
       <validationErrorText>Invalid data entry</validationErrorText>
    </validationRule>
   </dataEntryRule>

  <processRule name="RunLSSRule">
    <processName>specifies the path to the rule</processName>
  </processRule>

</rules>

</configurationFile>
```

APPENDIX B
JAVA RULE FOR EXECUTING LSS APPLICATION IN IN-VIGO

The rule code for the execution of the application in In-Vigo is shown below:

```
package invigo.rules.lss;

import java.lang.*;
import java.io.*;
import invigo.invigoutilities.*;
import invigo.invigorule.*;
import invigo.virtualapplication.VirtualApplication;
import invigo.resourcemanagement.InVigoJobRequest;
import org.apache.log4j.*;
import invigo.resourcemanagement.QOSInfo;


public class VAP_LSS_RunLSSRule extends InVigoRule {

   private  Logger logger = Logger.getLogger(VAP_LSS_RunLSSRule.class);

   public VAP_LSS_RunLSSRule() {
   }

   public void runRule() throws RuleExecutionInterrupt {
      String baseDir=null;
      String toolBaseDir = null;
      String jobName = null;
      String LSSPath = null;
      String envPath = null;

      logger.debug("Started execution of VAP_LSS_RunLSSRule !");
      importFiles("/home/users/acissoft/software/Linux/LSS","fit.m");

      try {
         baseDir=getSymbolValue(VirtualApplication.properties[VirtualApplication.SYS
TEM_WORKINGDIR]);
         LSSPath= "/home/users/acissoft/software/Linux/LSS";
         envPath = "/usr/bin";

         logger.debug("baseDir= "+baseDir);
         logger.debug("LSSPath= "+LSSPath);
```

```
            jobName = envPath + "/time" + " " + LSSPath + "/lss_serial_directory " + " " +
getSymbolValue("Directory") + " " +  getSymbolValue("ImageFile") + " " +
getSymbolValue("MinWL") + " " + getSymbolValue("MaxWL");

            logger.debug("jobName= "+jobName);

        } catch (RuleException e) {
            logger.error("Failed to obtain base variables: "+e.getMessage());
        }


        InVigoRuleTask t = null;

        try {
            t=executeJob(jobName,
            null,
            true,
            null,
            "Result.out",
            "Time.out",
            null);
        } catch (RuleException e) {
            logger.error("LSS Analysis job failed to execute: "+e.getMessage());
        }
    }
}
```

LIST OF REFERENCES

1. V. Backman, R. Gurjar, K. Badizadegan, R. Dasari, I. Itzkan, L.T. Perelman and M.S. Feld, "Polarized Light Scattering Spectroscopy for Quantitative Measurement of Epithelial Cellular Structures In Situ", IEEE Journal of Selected Topics in Quantum Electronics, Vol. 5, No. 4, August 1999, pp. 1019-1026.

2. V. Backman, L.T. Perelman, J.T. Arendt, R. Gurjar, M.G. Muller, Q. Zhang, G. Zonios, E. Kline, T. McGillican, T. Valdez, J. Van Dam, M. Wallace, K. Badizadegan, J.M. Crawford, M. Fitzmaurice, S. Kabani, H.S. Levin, M. Seiler, R.R.Dasari, I. Itzkan, and M. S. Feld, "Detection of Preinvasive Cancer Cells In Situ", Nature, Vol. 406, No. 6791, July 2000, pp. 35-36.

3. V. Backman, V. Gopal, M. Kalashnikov, K. Badizadegan, R. Gurjar, A. Wax, I. Georgakoudi, M. Mueller, C.W. Boone, R.R. Dasari, and M.S. Feld, "Measuring Cellular Structure at Submicron Scale with Light Scattering Spectroscopy", IEEE Journal of Selected Topics in Quantum Electronics, Vol. 7, No. 6, December 2001, pp. 887-893.

4. V. Backman, R. Gurjar, L.T. Perelman, I. Georgakoudi, K. Badizadegan, R. Dasari, I. Itzkan, and M.S. Feld, "Imaging of Human Epithelial Properties with Polarized Light Scattering Spectroscopy", Nature Medicine, Vol. 7, No. 11, November 2001, pp. 1245-1248.

5. V. Backman, Y. L. Kim, Y. Liu, R.K. Wali, H.K. Roy, M.J. Goldberg, A.K. Kromine and K. Chen, "Simultaneous Measurement of Angular and Spectral Properties of Light Scattering for Characterization of Tissue Microarchitecture and its Alteration in Early Precancer", IEEE Journal of Selected Topics in Quantum Electronics, Vol. 9, No. 2, April 2002, pp. 243-256.

6. G. von Laszewski, M-H. Su, J. Insley, I. Foster, J. Bresnahan, C. Kesselman, M. Thiebaux, M. Rivers, S. Wang, B. Tieman, and I. McNulty, "Real-Time Analysis, Visualization, and Steering of Tomography Experiments at Photon Sources", Proc. of the 9th SIAM Conference on Parallel Processing for Scientific Computing, San Antonio, Texas, March 22-24, 1999.

7. S. Smallen, W. Cirne, J. Frey, F. Berman, R. Wolski, M. Su, C. Kesselman, S. Young, and M. Ellisman, "Combining Workstations and Supercomputers to Support Grid Applications: The Parallel Tomography Experience", Proc. of the 9th Heterogeneous Computing Workshop (HCW 2000), Cancun, Mexico, May 1, 2000, pp. 241-252.

8. A. Apostolico, H. Benoit-Cattin, V. Breton, L. Cerrutti, E. Cornillot, S. Du, L. Duret, C. Gautier, C. Guerra, N. Jacq, R. Medina, C. Michau, J. Montagnat, G. Norstedt, A. Robinson, C. Odet and M. Senger, "Requirements for Grid-Aware Biology Applications", DataGrid WP10 Workshop, September 2001.

9. MPI: A Message Passing Interface Standard, http://www.mpi-forum.org/index.html

10. R. Figueiredo, "VP/GFS: An Architecture for Virtual Private Grid File Systems", Technical Report TR-ACIS-03-001, ACIS Laboratory, Department of Electrical and Computer Engineering, University of Florida, May 2003.

11. R. Figueiredo, N. Kapadia and J. Fortes, "The PUNCH Virtual File System: Seamless Access to Decentralized Storage Services in a Computational Grid", Proc. of the 10th IEEE International Symposium on High Performance Distributed Computing, San Francisco, California, August 7-9, 2001, pp. 334-344.

12. S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu. "From Virtualized Resources to Virtual Computing Grids: The In-VIGO System", Future Generation Computing Systems, special issue, Complex Problem-Solving Environments for Grid Computing, David Walker and Elias Houstis, Editors (to appear).

13. G. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, "PVM 3.0 User's Guide and Reference Manual", Oak Ridge National Laboratory, ORNL/TM-12187, February 1993.

14. J. Henessy and D. Patterson, "Computer Architecture: A Quantitative Approach", 3rd edition, Morgan Kaufmann, 2002.

15. J. Bester, I. Foster, C. Kesselman, J. Tedesco and S. Tuecke, "GASS: A Data Movement and Access Service for Wide Area Computing Systems", Proc. of the 6th Workshop on I/O in Parallel and Distributed Systems, Atlanta, Georgia, May 1999, pp. 78-88.

16. B. Callaghan, "NFS Illustrated", Addison-Wesley, 2002, ISBN 0201325705.

17. R. Figueiredo, P. Dinda and J. Fortes, "A Case for Grid Computing on Virtual Machines", Proc. of the 23rd International Conference on Distributed Computing Systems (ICDCS), Providence, Rhode Island, May 19-22, 2003.

18. G. Berti, S. Benker, J.W. Fenner, J. Fingberg, G. Lonsdale, S.E. Middleton and M. Surridge, "Medical Simulation Services via the Grid", Proc. of HealthGrid Workshop 2003, Lyon, France, 2003.

19. D. Sarrut and S. Miguet, "ARAMIS: A Remote Access Medical Imaging System", Proc. of 3rd International Symposium on Computing in Object-Oriented Parallel Environments, San Francisco, USA, December 1999, pp. 55-60.

20. A. Chervenak, I. Foster, C. Kesselmann, C. Salisbury, S. Tuecke, “The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets”, Journal of Network and Computer Applications, Vol. 23, No. 3, July 2000, pp. 187-200.

21. W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger and K. Stockinger, “Data Management in an International Data Grid Project”, Proc. of the 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000), India, December 2000.

22. J.S. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany and R. Wolski, “The Internet Backplane Protocol: Storage in the Network”, NetStore '99: Network Storage Symposium, Seattle, Washington, October 1999.

23. D. Thain, J. Basney, S-C. Son, and M. Livny, “The Kangaroo Approach to Data Movement on the Grid”, Proc. of the10th IEEE International Symposium on High Performance Distributed Computing (HPDC10), San Francisco, California, August 2001, pp. 325-333.

24. B. White, A. Grimshaw, and A. Nguyen-Tuong, “Grid-based File Access: the Legion I/O Model”, Proc. of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC’00), Pittsburgh, Pennsylvania, August 2000, pp.165-173.

25. B. White, M. Walker, M. Humphrey and A. Grimshaw, “LegionFS: A Secure and Scalable File System Supporting Cross-Domain High-Performance Applications”, Proc. of Supercomputing (SC2001): High Performance Networking and Computing, Denver, Colorado, November 10-16, 2001.

BIOGRAPHICAL SKETCH

Jithendar Paladugula was born in Warangal, Andhra Pradesh, India in 1980. He received his bachelor's degree with Roll of Honor in electrical and electronics engineering from Regional Engineering College Warangal, India in May 2001.

He joined the University of Florida to pursue a master's degree in electrical and computer engineering. In Fall 2002, he joined the Advanced Computing and Information Systems (ACIS) Lab as a graduate research assistant under the guidance of Dr. Renato J. Figueiredo. His research at the ACIS lab forms the basis of this work. His research interests include parallel computing, distributed computing and computer networks.