

# Fine-Grain Parallel Reconstruction Algorithm for MR Images

Hyo Jong Lee

Div. of Electronic and Information  
Chonbuk National University  
Jeonju, Korea

Steven Potkin

Dept. of Psychiatry and Human Behavior  
University of California, Irvine  
Irvine, CA 92697

## Abstract

The role of magnetic resonance imaging (MRI) analysis is critical in medical diagnoses for performing a noninvasive diagnostic technique. The raw data of MRI consists of series of complex numbers that represent phases and amplitudes of signals. The reconstruction procedure of the MRI requires lengthy computational time and complicated procedure due to the heavy discrete Fourier transformation. A parallel algorithm is developed for the cluster of heterogeneous computers and analyzed based on its performance, focusing on communication overhead and granularity. Granularity and scalability are critical factors to determine the performance of parallel reconstruction algorithms. The performance of fine granularity is more efficient compared to the one of coarse granularity, if its input and output (I/O) processing is fast enough to handle messages from all slave processors.

## 1 INTRODUCTION

Magnetic resonance imaging (MRI) has been recognized as an inevitably important diagnostic tool in the field of medical science for more than two decades. Schizophrenia, Alzheimer's disease and other various mental disorders are examples of those that depend on the neuro-imaging techniques [6, 7]. However, the reconstruction procedures of MR images are not trivial due to the complicated nature of MRI [1]. Furthermore, the process is computationally intensive and takes a large space of memory due to the heavy discrete Fourier transformation and lengthy sequence of computation. Other medical imaging techniques, such as computerized tomography (CT) [3, 9], positron emission tomography (PET) [4, 8], and single photon emission tomography (SPECT) [2, 14] are implemented in parallel algorithms and their efficiencies are reported. Although it is demanded at many places, the functional MRI has not been parallelized except for Lee's research [12].

A sequence of MRI consists of 3-D volume data. The reconstruction is critical to the success of MRI

analysis because the greater the ratio of signal to noise (S/N) of the MRI is, the better the results are. Fortunately, the structure of raw data resembles the natural composition of the 3-D volume. A series of complex numbers represents the 2-D slice data from top to bottom. The 2-D slices are overlapped to construct the 3-D volume. These volume data are taken sequentially to capture the change of signal strength inside the body. Thus, the 200 volumes of data contain around 6000 or more slices.

Reconstruction of 3-D volume images from raw data requires exhaustive computation of discrete Fourier transformation with complex numbers. The processing time is closely related with the performance of systems due to lengthy processing. For example, Sun Ultra SPARC 5 took about 80 seconds to reconstruct a single volume, while a fast Sun server machine took about 30 seconds. If a single scan contains 200 volumes, and a diagnosis for one subject needs ten scans, 2000 volumes should be reconstructed before any analysis. To reconstruct all volumes will take almost two days with a slower processor. However, the data are natural to the parallel reconstruction method because computation of each volume is completely independent of each other.

The purpose of this research is two-fold: to develop a parallel reconstruction algorithm for MRIs, and to analyze the factors that affect its performance on the network of workstations. A parallel reconstruction algorithm for MRI is developed for networks of workstations and its performance is presented in this paper. The remainder of this paper is organized as follows: The problem with heterogeneous network is analyzed in Section 2. An adaptive parallel algorithm is described comparing coarse and fine granularity methods in Section 3. Experimental results are discussed in Section 4. Finally, Section 5 concludes the paper.

## 2 HETEROGENEOUS NETWORKS

It is common to add newer systems to an existing network of workstations. To build a larger network

in this manner is economically practical. The clustered networks of the workstation provide an excellent parallel processing environment with robust processing powers, memory and storage units. However, the network does not have homogeneity in terms of system resources, such as processor power, memory and network transfer speed.

Since the performance of parallel algorithms depends on those system resources, it is difficult to implement optimized parallel algorithms for heterogeneous networks. Communication delay, which varies from 120 ms, may cause a severe delay for computation due to the synchronization required during a sequence of computation. Furthermore, the system's load of each node within the network is different, depending on users from the outside because the target network is assumed to be opened to public. Thus, it is an eminent problem to design parallel algorithms that utilize every processor of a network of heterogeneous workstations.

In this section the effect of heterogeneity of networks is described. Some criteria that should be considered for the parallelizing of algorithms in this situation are described. In particular, the focus will be on factors that affect the performance of the algorithm and the system environment.

## 2.1 Network description

The platform considered in this research is a group of mixed high and low performance workstations that are connected with an ordinary internet. This is the chosen target for a few reasons. First, this type of networked workstations is commonplace. Second, supercomputers are expensive, with limited availability as compared to the network of workstations. It has been reported that most of the workstations are idle up to 80% depending on time of day [15]. These interconnected workstations provide emulated parallel machines with a large aggregate processing power.

Systems that performed the current research consist of Sun UltraSPARC 1s, 5s and IIe. Their clock speed ranges from 165 Mhz to 500 Mhz and their memory sizes vary from 64Mb to 1Gb. The workstations are interconnected with inexpensive switching hubs of maximum transfer speed of 10Mbps. The current configuration consists of 30 processors. Each has a remote file service through NFS and share common file system. The system runs the MPI on the top of the SunOS. The network performance varies depending on the workstations. Figure 1 shows bandwidths for slower and faster workstations.

Maximum bandwidth of slower workstations barely passes 1Mbytes/sec, while the maximum bandwidth of faster workstations reaches approximately 10Mbytes/sec. The size of the messages which cause

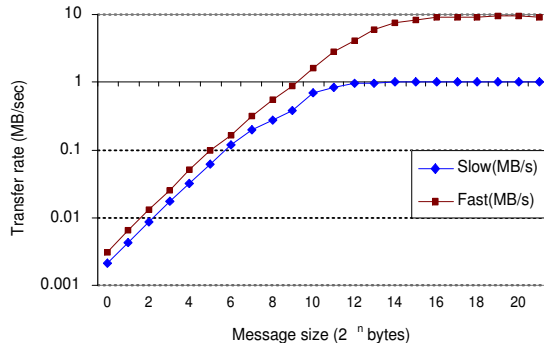


Figure 1: Bandwidth for slow and fast workstations

network saturation are  $2^{12}$  and  $2^{16}$  bytes for slower and faster workstations, respectively. It is clear that the message size of  $2^{12}$  bytes or smaller will be transferred without major delay.

The different completion time of each workstation is also a serious problem of parallel algorithm development. The execution time for reconstructing 80 volumes ranges from approximately 32 minutes to 95 minutes. The difference in speed is caused by both the heterogeneity in system resources, such as different clock speed and memory, and the difference in workstation load. In conclusion, the workstations have typical characteristics of a heterogeneous system with low capacity of network transfer.

## 2.2 Performance criteria

The system described in the previous section is reliable and has an advantageous cost to performance ratio. However, different system resources, unpredictable system load, and low network bandwidth are the serious disadvantages. The performance of the parallel reconstruction depends on how those disadvantages are handled. Criteria considered to improve performance are message passing, load balancing and scalability.

### 2.2.1 Message passing

The parallel tool used in this research is MPI(Message Passing Interface) [5], which performs a parallel task based on message passing among processors. Every data and control of a program is processed by message passing. This method is not favorable for networks with low bandwidth. The size of raw data for a 150-volume image is over 470 Mbytes, which requires too much transmission time and the serious communication overhead will degrade efficiency.

### 2.2.2 Load balancing

The goal of load balancing is to maximize the power of every workstation. Two factors, granularity and job distribution methods, can control load balancing. The granularity of parallelism is a good control factor to achieve load balancing. Piotrowski et al. [4] conducted experiments on performance sensitivity with variable granularity. This may introduce additional complexity in the program flow control, but specific task size can be distributed to a specific processor. However, their conclusion is that a fixed granularity performed better compared to varied granularity.

Task distribution in different sizes can help load balancing. The structure of the reconstruction algorithm is a hierarchical tree. One scan for a specific subject consists of multiple volume; each volume consists of multiple slices; and each slice consists of multiple Fourier transformations on each column. Parallelizing scans, volumes, slices, and columns results in finer granularity, respectively. The magnitude of the granularity in terms of processing times is 5200, 70, 20, and 0.4 seconds, respectively.

Two different approaches are implemented. The first approach is to divide the tasks in equal sizes. The second approach is to apply the concept of the processor pool. Thus, each slavery processor will request whenever it has completed the allocated task. The algorithms are described in Section 3 in detail.

### 2.2.3 Scalability

In some cases, the number of subjects may be only one, while in other cases, the number could be more than twenty. For example, Biomedical Informatics Research Network (BIRN) [16] often needs to analyze multiple subjects at multiple sites with different scanning. In this case, the number of scans is easily over 400 subjects. The algorithm requires good scalability to handle different data sizes. It should also be able to handle different sizes of workstations. The adaptive algorithm will perform best in any case.

## 3 PARALLEL RECONSTRUCTION

Currently, there are several different kinds of reconstruction methods available [1, 13]. A heuristic reconstruction algorithm has also been developed [10]. The methods depend on the specific MRI hardware system, acquisition protocol (software), and analysis purpose. However, a general reconstruction method has common procedures. This section will describe the general reconstruction method, followed by the description of two parallel reconstruction methods.

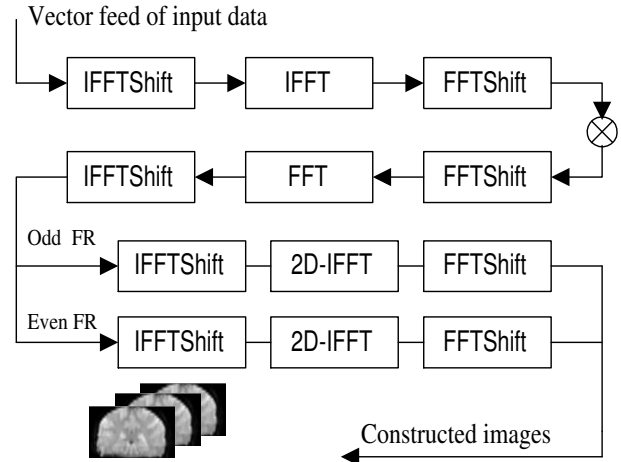


Figure 2: Flowchart of reconstruction

### 3.1 Sequential reconstruction method

A core of sequential reconstruction method is depicted in Figure 2. MRI scans often acquire partial  $k$ -space to reduce scan time by reducing the number of phase encodings. The partial  $k$ -space can be recovered by either filling or padding zeros or phase correction and conjugate synthesis. The raw data is fed into FFT-Shift operation followed by the 1-D Fast Fourier Transform (FFT) as the stream of 1-D inputs and builds correlation outputs.

The phase correction has to be made before the  $k$ -space symmetry can be exploited to synthesize the skipped data. Generally new data are synthesized by the conjugate symmetry for the missing data, such as  $M(kx, ky) = M * (-kx, -ky)$ . The phase correction function used is a unit amplitude image with a phase that is the conjugate of  $m_x(x, y)$ , which is the magnitude at a given location.

Then, the correlation result updates the  $k$ -space and is fed into the 2-D FFT after inverse FFT has been performed. Image volumes are not in order for most of the scanners because images are acquired in an interleaved fashion. The idea is to separate the odd volumes from the even volumes before passing through the 2-D FFT. Then, the FFT shift operations are applied to the outputs of 2-D FFT. Finally, the resulting volumes are combined together in proper order to build a 3-D volume data.

### 3.2 Parallel reconstruction method

Both coarse-grain and fine-grain algorithms are developed. Assuming the size of scan data is large, two methods may be considered: a volume-based and a slice-based parallel algorithm. The size of a single volume is about 1Mbytes for  $128 \times 128 \times 32$  resolution.

A slice is considered 2-D data representing one slice at the Cartesian coordinate. Since the order is not sequential from the raw scanning, the computation of slices slightly interferes with other slices. The typical size of a single slice would be 32Kbytes for the above resolution. Details of algorithms are described in the following sections.

### 3.2.1 Processor utilization

It is apparent performance would be improved at a higher processor utilization. The simple way to parallelize reconstruction method is to distribute equal number of tasks to each processor. This method can be implemented easily by distributing data in equal sizes. However, the utilization of fast processors will be low. Thus, it is expected to see unbalanced load due to different processing power and preoccupied outside tasks in the event the network is opened to public.

The processor farming technique is an excellent tool to create higher processor utilization. This technique has been used by Lee and Lim [11]. The higher utilization can be achieved by distributing tasks whenever a slave processor requests a new task. A master processor controls the distribution and considers the group of slave processors as a processing farm. Both coarse-grain and fine-grain methods adapt the processing farm technique to achieve better utilization of processors.

### 3.2.2 Coarse-grain method

This method parallelizes reconstruction algorithm by distributing volumes, a relatively large task. The detail algorithm is described in Figure 3. The pseudocode of the coarse-grain method consists of two blocks: an *if* block for a master processor and a *while* block for a slave processor. Initially, the master allocates a volume to each slavery processor sequentially by sending volume number *k*. Then, the master processor waits requests from slaves. As soon as it receives a message from a slave processor, it distributes a new volume number. It repeats the same steps until all volumes are distributed. After all volumes are distributed, the master distributes a *DONE* signal to each slave processor.

The *while* block in the pseudocode specifies for the actual reconstruction. Initially, every slave processor will get at least one task inside the first *if* block of Figure 3. Since it has already received a volume identification number *k* from the master node, it can reconstruct a volume by calling *getVolume()*. The *getVolume()* moves the file pointer to the corresponding *k*-th volume location and constructs each slice sequentially. The single execution time of *getVolume()* is ap-

proximately 0.8sec, which makes the algorithm coarse. Once each volume is reconstructed, *getVolume()* writes the result image after combining and reordering every slice. Then, the slave processor sends its own processor identification number to the master, instead of receiving a next volume number. A slave processor reconstructs volumes until the received volume number is not the *DONE* signal. Thus, it did not need to send any message to a master. However, a mas-

```

Procedure coarseGrainMethod
Input rawdata
Output resultImg
if( node==master ) { /* dedicated master */
  /* initial distribution */
  for( p=0; p<noProc; p++ )
    send( p, p )
  /* distribute jobs to requesting nodes */
  for( p=nProc; p<=noFrame; p++ ) {
    recv( des, ANY$_SOURCE );
    send( p, des );
  }
  for( p=0; p<nProc; p++ ) /* all done */
    send( DONE, p )
}
else { /* slave processor */
  while (true) {
    recv( k, MASTER )
    if ( k==DONE )
      break
    for( j=0; j<noSlice; j++ )
      getVolume( k, j, image ) /* get kth frame */
    for( j=0; j<noSlice; j++ ) {
      combineImage( resultImg, image )
      writeImg( resultImg )
    }
    send( myId, MASTER )
  }
}
}

```

Figure 3: Pseudocode for the coarse-grain method

ter is waiting for a slave's request and allocates new volumes only to the requesting processors in the adaptive method. Thus, this allocation method keeps every slave processor continually busy.

The flow control of the adaptive method is guaranteed not to enter either a deadlock or a starvation situation. Every slave processor will receive at least one task initially. As soon as the slave processor completes the allocated task, it sends the next volume request. If there are more tasks to be distributed, the master will send a volume number to the requesting processor. If all volumes are already allocated to other slave processors, the slave processor will receive a *DONE* signal and close the entire process properly.

### 3.2.3 Fine-grain method

In this method the distributed task is divided into a smaller size than the coarse-grain method. Although the number of volumes is relatively large, the execution time for parallelizing volumes is long, as the difference of time duration between slow processors and fast processors is critical to overall performance. A single volume data consists of multiple slices. The reconstruction of slices requires little interaction with other slices. Thus, the reconstruction algorithm can be further parallelized at the slice level of reconstruction. The pseudocode in Figure 4 describes the functions of a master and a slave processor of the fine-grain method.

Each slice within a specific volume is distributed to the requesting nodes. Distributed slices may not belong to the same volume. In order to handle multiple slices from different volumes, a data type, *taskInfo* is defined. It holds information about a slice number, a volume number, and a task type. Initially, the master processor distributes slice numbers to all slave processors in the first for block. Then it waits until the earliest slave processor returns the constructed slice image in the second for block.

Slave processors receive task information and construct slices until the task type is an end signal *DONE* in the while block. Once slice and volume numbers are obtained, the slave processor constructs slices that are assigned to themselves called *calcImg()*. Then they return the constructed slices with corresponding task information. If a single volume consists of 28 slices, 28 processors are involved to reconstruct that single volume. The algorithm is considered as a fine-grain, because a single volume has been reconstructed by a single processor in the previous algorithm.

The number of message passing is also 28-times. Another difference between this method and the previous method is that the master processor writes down the slice images. Thus, each slave processor can contribute a whole processor time to the computation of reconstruction. However, the master processor must be designated as a dedicated master, which distributes slice numbers, receives data, and writes down results.

The size of a single slice depends on the image resolution. Both communication and I/O overhead affect the efficiency of algorithms. The pseudocode in Figure 4 may be changed so that each slave processor writes down the reconstructed data instead of sending it to a master processor, if the communication overhead is larger than the I/O processing time. The *recv(slice, taskInfo)* inside the *if* block of a master section and the *send(slice, MASTER)* inside the *while* block of a slave section in Figure 4 transfer data from a slave processor to a master processor. If those two

```
Procedure fineGrainMethod
Input rawdata
Output resultImg
if ( node==master ) { /* dedicated master */
  /* initial distribution */
  for ( p=0; p<noProc; p++ ) {
    update(taskInfo)
    send( taskInfo, p )
  }
  /* distribute jobs to requesting nodes */
  for ( s=nProc; s<noFrame+nProc; s++ ) {
    recv( taskInfo, ANY$_SOURCE )
    /* receive the earliest one*/
    recv( slice, taskInfo.src )
    if( s<noFrame*noSlice ) /* checke whether */
      task.type = SLICE /* more data exist */
    else
      task.type = DONE
    update( taskInfo ) /* update slice/volume */
    send(s, taskInfo.src)/* assign a new task */
    writeSlice( slice );/* write down a slice */
  }
}
else { /* slave processor */
  while (true) {
    /* get a new slice # */
    recv( taskInfo, MASTER )
    if ( taskInfo.type==DONE )
      break
    /*get a single frame */
    slice = getSlice( taskInfo, image )
    calcImg( slice ) /* construct 2D image */
    send( taskInfo, MASTER ) /* send slice # */
    send( slice, MASTER ) /* send a result */
  }
}
}
```

Figure 4: Pseudocode for the fine-grain method

statements are removed and the *writeSlice(slice)* statement is moved from a master section to a slave section, the pseudocode favors improved I/O processing over communication overhead.

## 4 EXPERIMENTS

Each method has been implemented with MPICH 1.2.5 and tested on the network of various Sun UltraSPARC workstations described in Section 2.1. Input image is selected from a set of functional MRI of 80 volumes. Each volume consists of 28 slices in this set. Although the test execution was performed during non-busy hours, some fluctuations of the system's load were observed. A coronal and a sagittal view of a reconstructed volume are shown in Figure 5.

Figure 6 shows the execution times of the coarse-grain and the fine-grain parallel algorithms for dif-

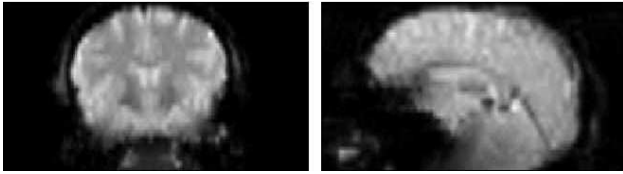


Figure 5: Sample of reconstructed MRI

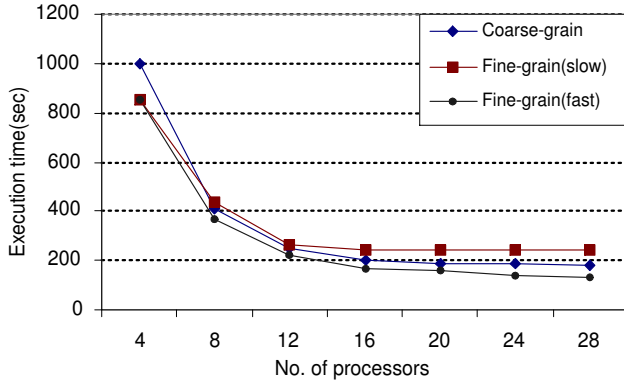


Figure 6: Execution time of algorithms

ferent numbers of processors. Dramatic reduction in execution time of parallel algorithms is found around twelve processors. The pattern of execution time of each method shows good parallelism.

The fine-grain method distributes slices to slave processors instead of volumes. In this method, a master processor receives and writes every slice image onto a destination disk. The master processor will be utilized 100% if the slave processors send slices continuously. Two curves of fine-grain, slow and fast represent slow and fast master processors, respectively. The faster processor is quicker in both I/O processing and computational power. It is obvious the execution time of the fast master is shorter than the slower one. Figure 6 also shows that fine granularity is not always better than coarse granularity. If the number of processors is eight or more, the coarse-grain algorithm is even better for the fine-grain algorithm at a slow master processor because the slow master processor cannot receive slices from a large group of slave processors.

Speed-up of each algorithm in Figure 6 is drawn in Figure 7. The speed-up of the fixed method slows after 24-processor, while others increase continuously. The fine-grain algorithm of a slow master processor does not show a distinct increment after 20-processor. The speed-up of the fine-grain is dramatic, if the master processor can handle slices from slave processors.

It is worthwhile to note that the pattern of the speed-ups for the two methods shows the super-linear speed-up, for which the performance is better than the ideal speed-up. For example, speed-ups for 20-processor are about 30 and 35 for each method. This

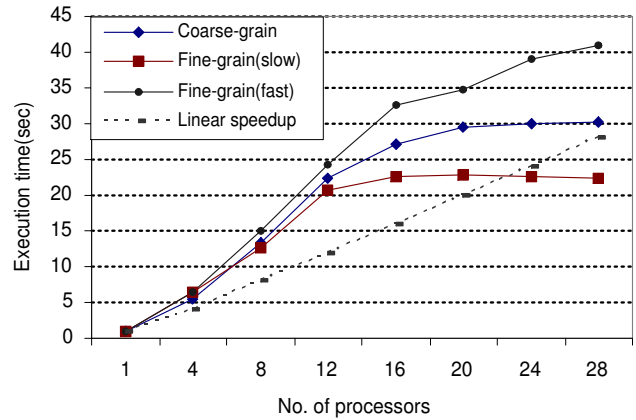


Figure 7: Speed up of each algorithm

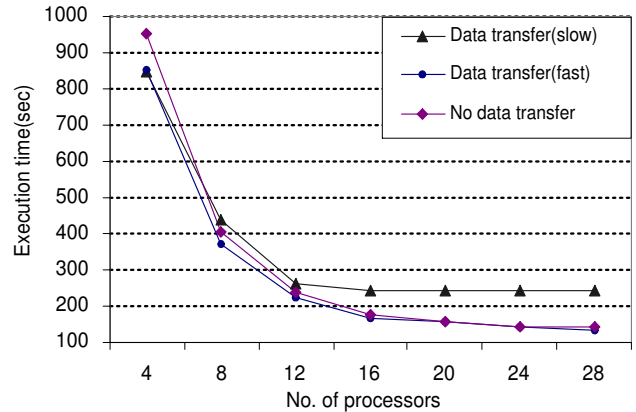


Figure 8: Execution time excluding data transmission

is possible because some slave nodes are faster than the first slave node, which executed the single processor data.

It is important to discover the effect of the communication overhead between the master processor and slave processors. Data transferring time can be estimated with the saturated bandwidth diagram shown in Figure 1. Transfer time for a single slice, for which the size is 32Kbytes, is estimated between approximately 25ms and 250ms on the experimented local network. The execution time is also measured after removing the send and recv for data transmission in the pseudocode described in Figure 4. Each slave processor writes down the computed slice image directly to a file system instead of sending it to a master processor. Figure 8 displays the execution times of three cases without data transmission. Curves for *Data transfer* show the measured execution time using *send/recv* operators for the fast and slow master processor, respectively. A curve for *No data transfer* shows the execution time for direct storing of slices by slave processors. Direct storing takes longer for a small number of processors, but its performance improves as the number of processors increases. This is perfectly logical. If

the number of slave processors increases, the master processor may become a bottle neck to write down all slice images. It suggests that distributed I/O must be sought, if the number of processor nodes is large.

## 5 CONCLUSION

Parallel reconstruction algorithms for functional MRI were developed and implemented with MPI for the network of Sun workstations. Primary goals of algorithm development are to reduce unnecessary communication, to achieve good load balancing, and to maintain scalability. The performance of parallel algorithms indicates improved speed-up. The algorithms mimic the processor farming model by distributing tasks to available slave processors. The processor farming technique works well for the heterogeneous networks. The performance of algorithms reflects the behavior of good linear speed up. It is expected that the speed up will continually increase linearly, if every slave processor has an equal amount of computing power, because the algorithm uses coarse granularity. However, this cannot occur in the network of low cost workstations. The experiment utilized two different granularities. The fine-grain method showed better performance, if the master processor is capable of handling slices received from slave processors, while the coarse-grain method is better when the master processor is a slow speed processor. The performance of the *no-data-transfer* method was more efficient than the *data-transfer* method. This is due to distributed I/O processing, along with the reduced communication overhead. It is expected the best performance will be achieved with fine-grain algorithm with a powerful master processor in terms of I/O processing.

## Acknowledgements

This research was supported by the Functional Imaging Research in Schizophrenia Testbed Biomedical Informatics Research Network (BIRN, <http://www.nbirn.net>), which is funded by the National Center for Research Resources at the National Institutes of Health (NIH). (5 MOI RR 000827)

## References

- [1] M. H. Bounocore and D. C. Zhu. Image-based ghost correction for interleaved epi. *Magnetic Resonance in Medicine*, 45:96–108, 2001.
- [2] R. R. Brechner and M. Singh. Iterative reconstruction of electronically collimated spect images. *IEEE Trans. On Nuclear Science*, 37(3):1328–1332, 1990.
- [3] C. M. Chen, S. Y. Lee, and Z. H. Cho. A parallel implementation of 3-d ct image reconstruction on hypercube multiprocessor. *IEEE Trans. On Nuclear Science*, 37(3):1333–1346, 1990.
- [4] C. M. Chen, S. Y. Lee, and Z. H. Cho. Parallelisation of em algorithm for 3-d pet image reconstruction. *IEEE Trans. On Medical Imaging*, 10(4):513–522, 1991.
- [5] MPI Consortium. Message passing interface, 2004.
- [6] V. A. Diwadkar, M. D. DeBellis, J. A. Sweeney, J. W. Pettegrew, and M. S. Keshavan. Abnormalities in mri-measured signal intensity in the corpus callosum in schizophrenia. *Schizophrenia Research*, 67(2-3):277–282, 2004.
- [7] P. M. Doraiswamy. Magnetic resonance markers in alzheimer’s disease clinical trials. *Molecular Imaging and Biology*, 6(2), 2004.
- [8] D.W.Shattuck, J.Rapela, E. Asma, A. Chatzioanou, J. Qi, and R. M Leahy. Internet2-based 3d pet image reconstruction using a pc cluster. *Physics in Medicine and Biology*, 47:2785–2795, 2002.
- [9] C. A. Johnson and A. Sofer. A data-parallel algorithm for tomographic image reconstruction. In *Proceedings of the 7th symposium on the Frontier of Massive Parallel Computation*, volume 67, pages 1–10, 1999.
- [10] Hyo Jong Lee. A heuristic method for the reconstruction of functional magnetic resonance images. In *Proceedings of International Conference on Imaging Science, System, and Technology*, 2004.
- [11] Hyo Jong Lee and B. H. Lee. Parallel ray tracing using processor farming model. In *Proceedings of Parallel Processing Workshop*, 2001.
- [12] Hyo Jong Lee, Jessica Turner, and Steven Potkin. Scalable parallel reconstruction algorithm for magnetic resonance images. In *Proceedings of Parallel and Distributed Processing Techniques and Applications*, 2004.
- [13] K. J. Lee, D. C. Barber, M. N. Paley, I. D. Wilkinson, N. G. Papadakis, and P. D. Griffiths. Image-based epi ghost artifact reduction using iterative phase correction. In *Conference for Medical Image Understanding and Analysis*, 2001.
- [14] F. Munz, T. Stephan, U. Maier, T. Ludwig, A. Bode, S. Ziegler, S. Nekolla, P. Bartenstein, and M. Schwaiger. Now based parallel reconstruction of functional images. In *Proceedings of International Parallel and Distributed Processing Symposium*, 1998.
- [15] M. Mutka and M. Livny. The available capacity of a privately owned workstation environment. *Performance Evaluation*, 12(4):269–284, 1991.
- [16] Biomedical Information Research Network. Birn project homepage, 2004.