To the Graduate Council:

I am submitting herewith a thesis written by John P. McNeely entitled "Data Compression Methods for Distributed EM-ML Reconstruction of PET Images." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Jens Gregor, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Associate Vice Chancellor
and Dean of the Graduate School

Data Compression Methods for Distributed EM-ML Reconstruction of PET Images

A Thesis
Presented for the
Master of Science Degree
The University of Tennessee, Knoxville

John P. McNeely
May 1998

Acknowledgments

My time at the University of Tennessee has been both challenging and rewarding. I am thankful to the faculty and graduate students in the Department of Computer Science for enriching my education experience. I would like to thank my Thesis Committee, Dr. Jens Gregor, Dr. Charles Romine, and Dr. Michael Thomason for their time, attention, and support.

Lastly, I want to thank my dearest wife, Sheila, for her patience and support over the course of my graduate studies.

Abstract

The expectation-maximization maximum-likelihood (EM-ML) algorithm for image reconstruction in positron emission tomography (PET) solves a linear system of equations in an iterative fashion. The computational, storage, and communication requirements for distributed computing with the EM-ML algorithm can be costly depending on the size of reconstruction data sets and the number of iterations required to achieve desired image quality. Partitioning of the iterative EM-ML algorithm requires a global reduction operation to consolidate and distribute locally computed results. The node-to-node communications required to complete a global reduction operation can introduce significant performance penalties depending on the type of network, number of nodes, and amount of data communicated. This thesis presents a study on improving EM-ML performance by introducing data compression methods into the global reduction communication components of EM-ML. Experiments were conducted on high-speed networks using a loosely-coupled cluster of parallel processing workstations and data collected from a CTI/Siemens ECAT 921 PET scanner as well as noise-free synthetic data. The findings here indicate that both statistical modeling and dictionary-based lossless compression methods do not improve performance of EM-ML distributed computations. Performance improvement is seen using lossy methods with fixed compression ratios and heuristic attributes resulting in much lower global reduction times.

Contents

1 INTRODUCTION **1**
- 1.1 Positron Emission Tomography 1
- 1.2 Expectation-Maximization 2
- 1.3 Distributed Computing 3
- 1.4 Focus of Attention 5
- 1.5 Objective 7
- 1.6 Thesis Outline 7

2 EM-ML DISTRIBUTED SYSTEM DESCRIPTION **8**
- 2.1 Distributed Computing Environment for EM-ML 8
- 2.2 Data Material 9
- 2.3 Message Passing in EM-ML 9
  - 2.3.1 Message Passing Interface System 9
  - 2.3.2 MPI Global Reduction Operation 10
- 2.4 Focus-of-Attention and Global Reduction Performance 15
  - 2.4.1 Call-graph Timing 15
  - 2.4.2 Focus-of-Attention Global Reduction Performance 18

3 LOSSLESS DATA COMPRESSION **24**
- 3.1 Statistical Compression Models 25
  - 3.1.1 Entropy and Coding 26
  - 3.1.2 Semi-Adaptive Huffman Coding 27
  - 3.1.3 Adaptive Huffman Coding 30
  - 3.1.4 Splay-prefix Coding 31
  - 3.1.5 Arithmetic Coding 33
- 3.2 Dictionary Compression Models 36
  - 3.2.1 LZ77 Dictionary Coding 38
  - 3.2.2 LZSS Dictionary Coding 39
  - 3.2.3 LZ78 Dictionary Coding 40
  - 3.2.4 LZW Dictionary Coding 41
- 3.3 Lossless Methods in Global Reduction 43

3.3.1 Compression Engine Performance 43
3.3.2 EM-ML Performance Results 47

4 LOSSY DATA COMPRESSION **52**
4.1 Transform Coding 52
4.2 Compression with Fixed-Point Arithmetic 54
4.3 Compression with Uniform Quantization 56
4.4 Distortion Measures and Image Quality 59
4.4.1 Brain Image Profiles 59
4.4.2 Region-of-Interest Measures 64
4.5 Lossy Methods in Global Reduction 70

5 CONCLUSIONS **75**

Bibliography **78**

Vita **83**

List of Tables

3.1 Compression Results for $\mathbf{M^k}$ - Brain at Normal Resolution. 44

3.2 Compression Results for $\mathbf{M^k}$ - Brain at three times Normal Resolution. 44

List of Figures

2.1 Illustration of MPI Global Reduction - Reduction 12

2.2 Illustration of MPI Global Reduction - Broadcast 14

2.3 EM-ML MPI Call Graphs 16

2.4 EM-ML Brain Reconstruction Performance - ATM 19

2.5 EM-ML Brain Reconstruction Performance - Ethernet 20

2.6 Synthetic Volume Global Reduction 22

3.1 Uniformity of $\mathbf{M^k}$ 46

3.2 EM-ML/Lossless Brain Reconstruction Performance 48

3.3 EM-ML/Lossless MPI Call Graphs 50

4.1 Normal and FPA Brain Images at Normal Resolution. 61

4.2 Normal and FPA Brain Images at 3 x Normal Resolution. 62

4.3 Normal and Quantized Brain Images at Normal Resolution. . . . 63

4.4 Normal and Quantized Brain Images at 3 x Normal Resolution. . 65

4.5 Simulated Phantom Data Set 67

4.6 ROI Measures - Fixed-Point Arithmetic 68

4.7 ROI Measures - Uniform Quantization 69

4.8 EM-ML/Lossy Brain Reconstruction Performance - ATM 71

4.9 EM-ML/Lossy Brain Reconstruction Performance - Ethernet . . . 72

4.10 Synthetic Volume Global Reduction - With Lossy Methods 74

Chapter 1

INTRODUCTION

1.1 Positron Emission Tomography

Positron Emission Tomography (PET) is a non-invasive method for examining the physiology of the human body. With PET a patient is injected with a radio-isotope which collects in the part of the body to be studied, then emits positrons as it decays. Each positron annihilates with an electron causing two photons to be emitted in opposing directions. Measurements of these photon pairs are collected by detector rings surrounding the patient. From these measurements a discretized emission map (image) that reflects the internal isotope concentration is reconstructed by computer. To reduce scatter, which is a significant noise component, septa are typically inserted between adjacent detector rings. This causes both the data acquisition and the image reconstruction to become two-

dimensional and is the scenario considered here.

1.2 Expectation-Maximization

Several iterative reconstruction algorithms have been proposed in the literature, such as algebraic reconstruction (ART) [12], expectation-maximization maximum-likelihood (EM-ML) [24, 17], Landweber [22], and conjugate gradient (CG) [16]. Common to all of these approaches is that they solve the linear system of equations

$$\boldsymbol{P}^T\boldsymbol{\lambda} = \boldsymbol{n}^* \tag{1.1}$$

where $\mathbf{n}^*$ is a $D \times 1$ vector representing the sinogram (that is, the photon emissions recorded by the detectors), $\boldsymbol{\lambda}$ is a $B \times 1$ vector representing the unknown image, and $\mathbf{P}$ is a sparse $B \times D$ matrix that describes the connection between the internal emission activity and the observed data.

The research for this thesis is based on the EM-ML approach [4]. The EM-ML algorithm concentrates on finding a $\boldsymbol{\lambda}$ that maximizes the likelihood of $P(\boldsymbol{n}^*|\boldsymbol{\lambda})$ based on a Poisson model of the emission process. The EM-ML algorithm solves the system of linear equations given in Equation 1.1 through an iteration scheme given by [24, 17]

$$\forall b : \lambda_b^{k+1} = \lambda_b^k \sum_d \frac{n_d^*}{\lambda_d^{*k}} p_{bd} \tag{1.2}$$

where k is the current iteration number, b denotes a given pixel, λ_b is the expected number of photon pairs emitted from pixel b, n_d^* denotes the number of photon pairs recorded by a tube d, λ_d^* denotes the expected number of photon pairs recorded by tube d, where $\lambda_d^* = \sum_b \lambda_b^k p_{bd}$, and p_{bd} is the conditional probability that a photon pair is recorded by tube d given that it originated from pixel b ($\sum_d p_{bd} = 1$). Important features of the algorithm are that $\boldsymbol{\lambda}^k$ remains non-negative for all k and $\sum_b \lambda_b^k = \sum_d \lambda_d^{*k} = \sum_d n_d^*$ for $k > 0$.

Equation 1.2 may also be expressed as $\lambda_b^{k+1} = \lambda_b^k M_b^k$ where multiplier M_b^k is shorthand for the right-hand side summation. In addition, the notation $\mathbf{n}^* = [n_d^*]$, $\boldsymbol{\lambda} = [\lambda_b]$, $\boldsymbol{\lambda}^* = [\lambda_d^*]$, $\mathbf{M} = [M_b]$, and $\mathbf{P} = [p_{bd}]$ is used when referring collectively to the entities in 1.2.

1.3 Distributed Computing

When considering how to distribute the iterative EM-ML algorithm in a distributed memory system, a more convenient form is given by:

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k \otimes (\mathbf{P}(\mathbf{n}^* \oslash (\mathbf{P}^{\mathbf{T}} \boldsymbol{\lambda}^k))) \tag{1.3}$$

where $\otimes$ and $\oslash$ denote the element-wise multiplication and division operations, respectively.

The computation for Equation 1.3 may be broken down as follows:

1. First the forward-projection $\boldsymbol{\lambda}^{*k} = \mathbf{P}^{\mathbf{T}}\boldsymbol{\lambda}^{k}$ is computed.

2. Next the vector $\phi^{k} = \mathbf{n}^{*} \oslash \boldsymbol{\lambda}^{*k}$ is computed.

3. Then the vector of multipliers is obtained with $\mathbf{M^k} = \mathbf{P}\phi^{\mathbf{k}}$.

4. Finally, $\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^{k} \otimes \mathbf{M^k}$ completes the computation.

With reference to Equation 1.3, two vector-matrix multiplications dominate the computation and is thus proportional to the number of nonzero matrix $\mathbf{P}$ elements. Each node in the distributed computation must communicate localized results with the other participating nodes at least once during each iteration, thus requiring a global reduction operation. Each node has a complete image vector $\boldsymbol{\lambda}$ and a subset of the $\mathbf{P}$ matrix and $\mathbf{n}^{*}$ vector. The global reduction operation then sums incomplete local copies of $\mathbf{M^k}$ which makes the amount of data communicated by each node proportional to B. Thus, the speed of the global reduction is dependent on the number of nodes participating, the speed of the underlying network, the speed of the nodes performing summation, and the amount of data to be reduced.

1.4 Focus of Attention

Prior research by [10, 9] has resulted in a preprocessing scheme which focuses attention on a subset of equations and unknowns in the EM-ML algorithm thereby reducing the overall storage, computation, and communication requirements. The focus-of-attention scheme modifies equation 1.2 by:

1. Computing an $\mathbf{n}^*$-mask that indicates the subset of relevant equations (tubes).
2. Computing a $\boldsymbol{\lambda}$-mask that indicates the subset of relevant unknowns (pixels).
3. Computing and storing matrix $\mathbf{P}$ for the tubes in $\mathbf{n}^*$-mask and the pixels in $\boldsymbol{\lambda}$-mask.
4. Reducing the count of boundary tubes in $\mathbf{n}^*$-mask.

These modifications are driven by observations which suggest that some parts of the linear system of equations described by Equation 1.1 are unnecessary for image reconstruction, as they do not actually contribute to the reconstruction process.

With respect to the distributed computing algorithm in (1.3), under the focus-of-attention model the nodes need only exchange information pertaining to the pixels covered by the $\boldsymbol{\lambda}$-mask. Thus, $\mathbf{M^k}$ can be locally compressed in a lossless fashion before the global reduction operation thereby reducing the amount of data to be communicated and summed.

The global reduction operation can be time consuming depending on such factors as how much data are communicated, speed of the underlying network bus, and the method chosen for reducing the data and broadcasting the results back to all the nodes. Even for relatively small data sets, like 2D slices, this reduction operation introduces a modest performance penalty. Applying EM-ML to larger data sets, such as a 3D volume consisting of 47 2D slices, may result in a considerable reduction in performance as CPU time increases with the communication and summation of a larger number of terms in $\mathbf{M^k}$. In addition, network saturation will introduce non-linear behavior in communication times should it occur.

The focus-of-attention model attempts to address this problem by *packing* $\mathbf{M^k}$ before the global reduction operation. This approach has shown effectiveness in speeding up both local computation and group communication. However, experimentation has shown that focus-of-attention becomes less effective when applying a zoom factor, thereby expanding the image boundaries, and when very little data may be discarded (ignored) during image reconstruction. That is, if $\mathbf{M^k}$ becomes dense, then the focus-of-attention packing operation is unable to discard much, if any, of the data.

1.5 Objective

The objective of this thesis is to determine if alternative lossless and lossy data compression methods can be applied to reduce the amount of time spent communicating between nodes during global reduction operations and thereby reduce the overall computation time of EM-ML in a distributed computing environment. The data compression methods may exploit inherent redundancy, data patterns, or data formats to effectively reduce storage size and communication times.

1.6 Thesis Outline

Chapter 2 provides a description of the EM-ML system environment, use of message passing for global reduction, and a baseline of EM-ML communication performance. Chapter 3 examines lossless data compression and provides experimental results for both statistical and dictionary-based algorithms. Chapter 4 examines lossy data compression methods and provides experimental results for some heuristic approaches. Chapter 5 presents the conclusions of this thesis.

Chapter 2

EM-ML DISTRIBUTED SYSTEM DESCRIPTION

2.1 Distributed Computing Environment for EM-ML

The computing environment used for EM-ML experimentation is a network of 12 Sun Microsystems Ultra II Model 2170 workstations, each configured with two 167 MHz UltraSPARC-1 processors and 256 MB of RAM. During performance tests, only one processor is utilized by EM-ML. The workstations are connected to a 155 Mbps ATM (OC-3) switch as well as a 100 Mbps switched Ethernet. The EM-ML algorithm is parallelized by letting each workstation (node) maintain a local copy of $\boldsymbol{\lambda}^{k}$ while distributing $\mathbf{n}^{*}$ and the corresponding columns of the $\mathbf{P}$-matrix across the nodes. After each iteration, the nodes communicate localized results, $\mathbf{M}^{\mathbf{k}}$,

using the a global reduction operation.

2.2 Data Material

A CTI/Siemens ECAT 921 PET scanner is the source of most data used for these experiments. The ECAT 921 PET scanner generates 384 x 80 sinograms and 128 x 128 images for arbitrary offsets and zoom factors. The physical gantry radius is 41.2 cm with an effective full-view radius of 25.1 cm and default pixel width of 4.55 mm. All EM-ML data are stored in IEEE double-precision format. Both real and simulated phantom data sets are used for experimentation and include:

- Brain - FDG head sinogram.
- Phantom - noise-free sinogram obtained by projecting a version of the Shepp-Vardi phantom [24] via the **P**-matrix.

2.3 Message Passing in EM-ML

2.3.1 Message Passing Interface System

From Equation 1.3, $\mathbf{M^k}$ must be communicated between nodes in order to make localized computations known to all nodes participating in the distributed computation. Communication between nodes in EM-ML is handled by the Message Passing Interface (MPI) system [6]. MPI is a widely used standard for writing

message-passing programs and for programming parallel processors. MPI supports many computer architecture types, data formats, communication topologies, and collective communications — such as global reduction.

EM-ML uses MPI in the *Single Program Multiple Data* (SPMD) paradigm. Under SPMD the same program is executed on different processors and execution decisions are made based on the *rank* of the process within the collective group of processes, or *process group*. In MPI a process group may be configured into different topologies such as Cartesian (grid) or graph topology. A topology in MPI is simply a set of constraints placed on a process group. By default a process group topology is a linear rank ordering of the process from 0 to $N-1$ where N is the number of processes. The process with rank 0 is considered the root process in the process group. This special ranking is used in the global reduction operation.

2.3.2 MPI Global Reduction Operation

The global reduction operation in MPI performs a collective operation on a process group and broadcasts the results back to each process in $\lceil \log_2 N \rceil$ time. The root process (rank=0) is designated as the process that will ultimately gather all the collective results and then be the point from which the results are broadcast back out to the process group.

The reduction stage algorithm uses a masking operation to determine send and

receive process pairs. Initially the mask is set to 1 and each process performs a bitwise-AND operation using the mask against the process rank. If the result is zero then the process is a receiver. If the result is non-zero then the process is a sender. A receiver, with rank $rank_R$, then receives from the process with rank $rank_S = (rank_R \vee mask) \bmod N$. Once the receiver receives the data, the receiver process performs the given operation and accumulates the results. A sender, with rank $rank_S$, sends to the process with rank $rank_R = (rank_S \wedge \neg mask) \bmod N$. Once a sender finishes sending data, the sender then goes into a broadcast receive state waiting for the root process to send the collective results. All receivers next update the mask by shifting one bit to the right, effectively taking the mask to the next higher power of 2, and another reduction iteration is performed. The reduction is finished when the mask value is $>= N$. It should be noted that it is possible that the $rank_S$ calculated by a receiver will be $>= N$, in which case the receiver is idle for that reduction iteration. Figure 2.1 illustrates the reduction stage of the MPI global reduction operation for 12 nodes. Note the mask value at each step and that the reduction is completed in $\lceil \log_2 N \rceil$ time.

The broadcast stage of the MPI global reduction is implemented using an iterative tree-based algorithm that takes $\lceil \log_2 N \rceil$ iterations to complete. Processes are divided into participants and extra nodes. The idea is to send the data to enough participants so that on the last broadcast iteration there are enough participants

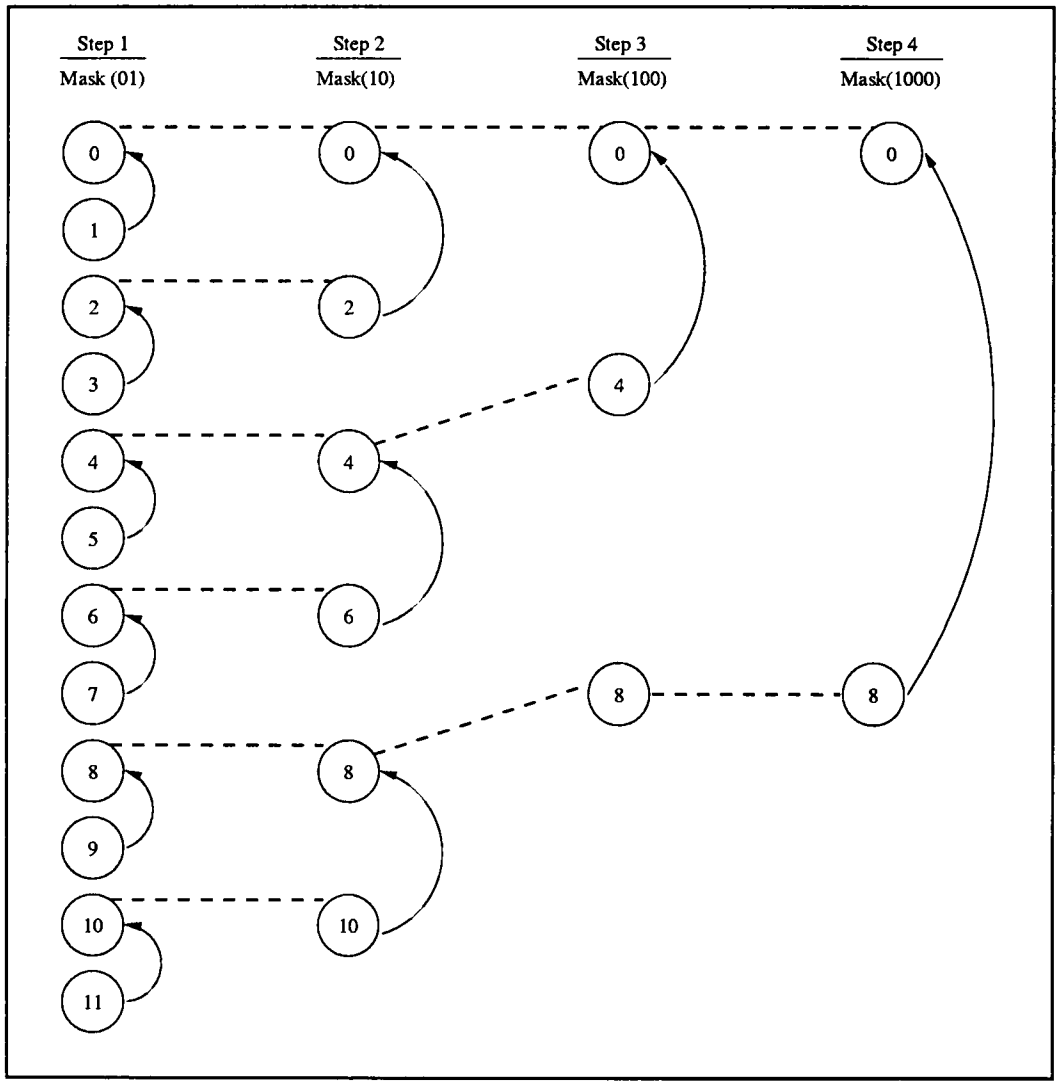


Figure 2.1: Illustration of reduction stage in the MPI global reduction operation. A 12 node reduction is presented with processes represented by the circles and send/receive pairs denoted by the directed arrows. The incident end of the arrow indicates the receiver.

to send to all of the extra processes. For process groups, with a size of $N > 1$ the number of participants, B_p, will be $B_p = \lfloor \log_2 N - 1 \rfloor$ and the number of extra processes, B_e, will simply be $B_e = N - B_p$. The root process is always a participant and starts the broadcast. Once a receiver receives data, the receiver becomes a logical root (root of subtree) and will send on the next broadcast iteration. At each participant broadcast iteration, i, there are 2^{i-1} processes capable of sending. On the last iteration, all extra processes receive data from some subset of participant processes. Figure 2.2 illustrates the broadcast stage of the MPI global reduction operation for 12 nodes. Note that the number of processes capable of sending does grow exponentially at each step and that the broadcast is completed in $\lceil \log_2 N \rceil$ time. In addition, the number of participant processes is $B_p = \lfloor \log_2 11 \rfloor = 8$ and the number of extra processes is $B_e = 12 - 8 = 4$.

This dissection of the reduction and broadcast stages of the global reduction operation is germane to understanding where data compression logically fits into the structure of the global reduction algorithm. With the broadcast stage, data can be compressed by the root process prior to sending and the decompressed by the receiving processes. The reduction operation is slightly more complex. There are two general cases where compression may be applied: i) before a process sends, the data are compressed and must then be decompressed by the receiver before the reduction operation can be performed; and ii) before each process enters into

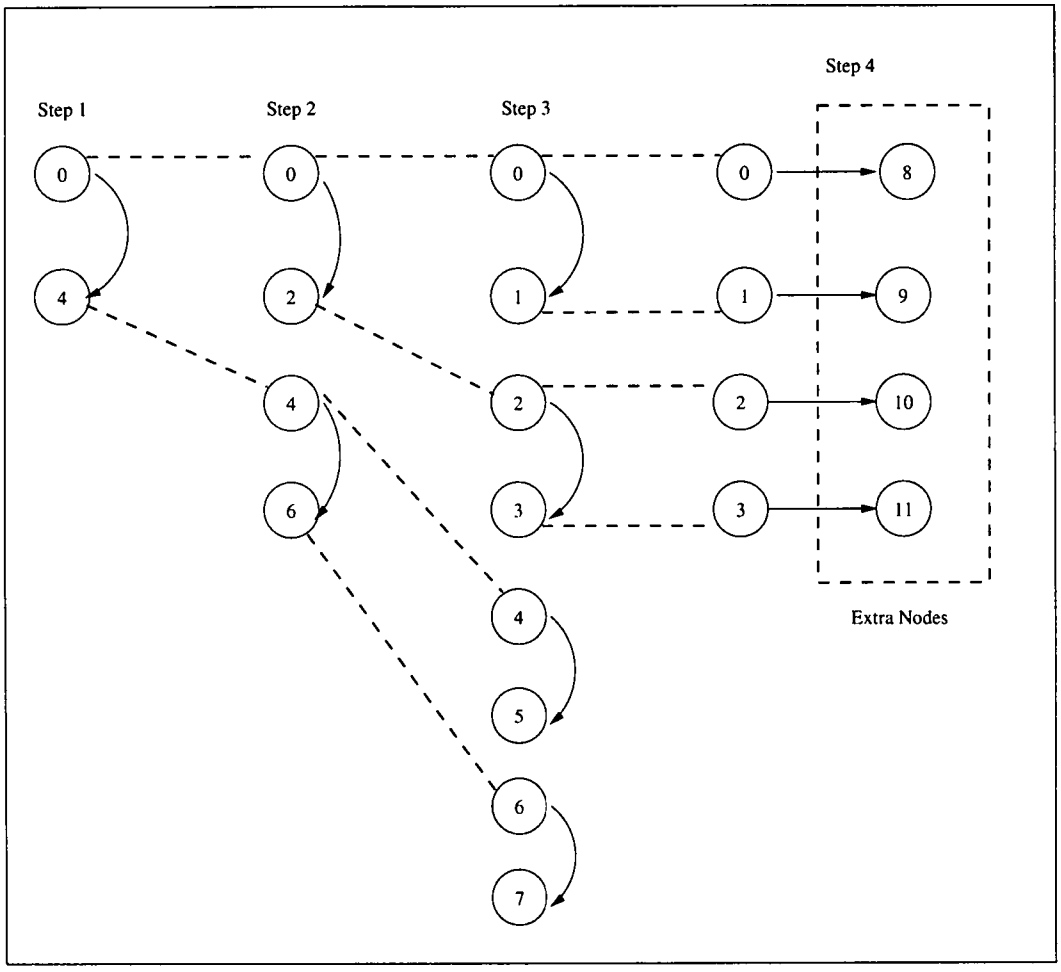


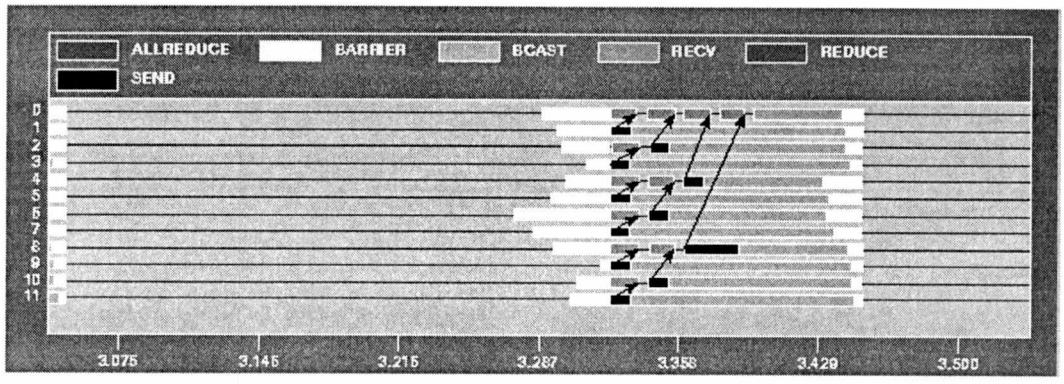
Figure 2.2: Illustration of broadcast stage in the MPI global reduction operation. A 12 node broadcast is presented with processes represented by the circles and send/receive pairs denoted by the directed arrows. The incident end of the arrow indicates the receiver.

the reduction loop, the data are compressed into a form whereby the reduction operation can be performed directly on the compressed data. The packing mechanism in the focus-of-attention model in [10, 9] is an example of case (ii). The choice of compression location is critical to the overall per-iteration global reduction communication time. In case (i), there are $2*(N-1)$ compression and decompression operations during the reduction stage and N compression and decompression operations during the broadcast stage for a total of $2*(N-1)+N$. Even though these operations are executed in parallel as dictated by the algorithms described above, there are fewer operations in case (ii). In case (ii) there are N compression operations and N decompress operations for a total of $2N$ compress/decompress operations. Hence, each node performs one compression and one decompression operation and all nodes perform each operation in one parallel step.

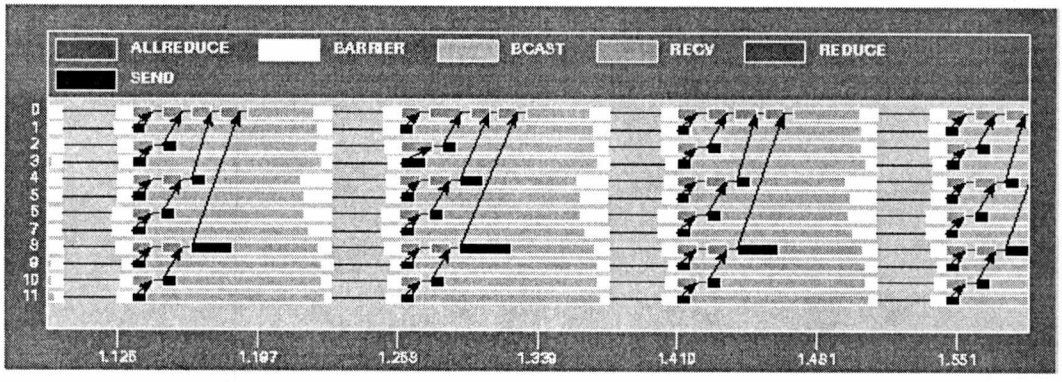
2.4 Focus-of-Attention and Global Reduction Performance

2.4.1 Call-graph Timing

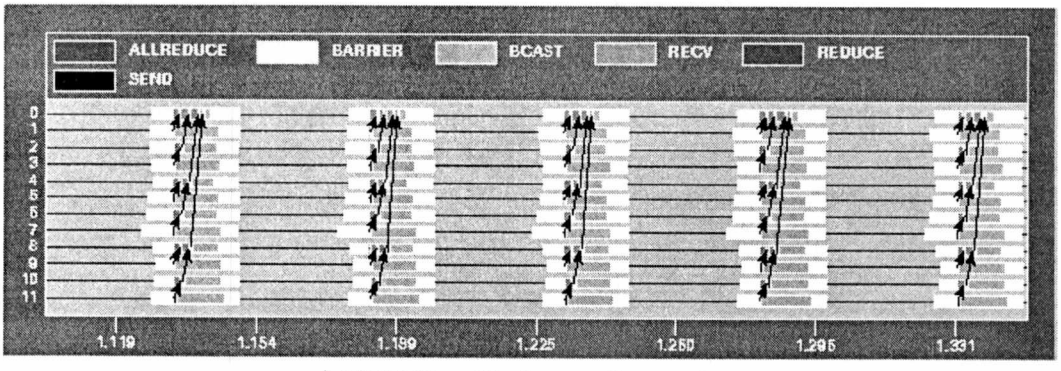
Figure 2.3 shows visual call-graphs with timing information from different EM-ML runs. Figure 2.3-a shows ordinary EM-ML; that is, without applying the focus-of-attention scheme. Figure 2.3-b shows EM-ML with focus-of-attention but without packing in order to show just compute effects. Figure 2.3-c shows EM-ML with focus-of-attention and packing. These runs used a 12-node configuration with all



a) Ordinary EM-ML.



b) EM-ML with focus-of-attention and no packing.



c) EM-ML with focus-of-attention.

Figure 2.3: EM-ML MPI call graphs for global reduction during a brain sinogram image reconstruction on the ATM network.

communications carried over the ATM network. Each call-graph shows 12 nodes vertically, rank 0 through 11, and time, in seconds, horizontally. Each row depicts the communication and computation activity of each node, respectively, for the given time-frame. The directed arrows indicate a send and receive pair. The incident end of the arrow indicates the receiver. The thin line between shaded blocks indicates non-MPI activity, in this case CPU times. In figure 2.3-a, one full iteration for ordinary EM-ML is shown. The thin lines depict the time spent computing localized results for λ^k and M^k. The global reduction is depicted by the cluster of shaded blocks. In the global reduction, node zero receives four times from nodes 1,2,4, and 8. In between each receive, the solid line indicates the summation operation. Once all summations are completed, node zero initiates the broadcast operation.

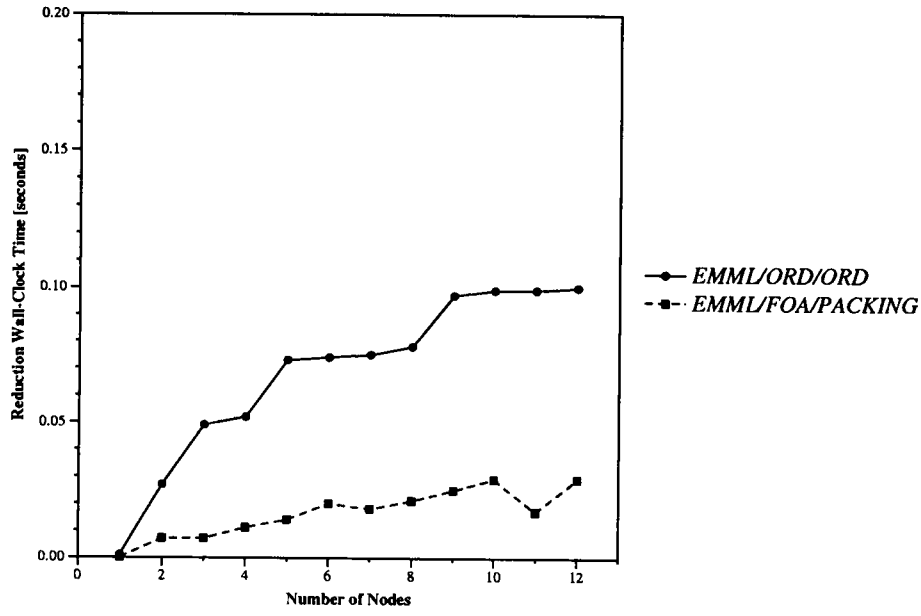
In ordinary EM-ML the time spent computing for an iteration clearly takes more time than communicating the results. With focus-of-attention applied, the compute time for each iteration can be reduced. The results of applying focus-of-attention can be seen in Figure 2.3-b. Depicted here is roughly the same amount of time as in Figure 2.3-a — 0.5 seconds. However, three full iterations are completed. Figure 2.3-c shows even better performance when packing is applied to the global reduction communications. The time resolution is roughly one-half that of figures 2.3-a and 2.3-b and yet five full iterations are completed giving roughly a

10:1 improvement over ordinary EM-ML. Note the reduction in time to complete a global reduction operation. In this case the brain image was reconstructed at normal resolution and nearly 90% of the data are ignored through focus-of-attention. But as will be seen later, this reduction in data and improvement in performance does not hold as the data used for image reconstruction becomes more dense, that is, the focus-of-attention method is unable to reduce the set of equations and unknowns for reconstruction.

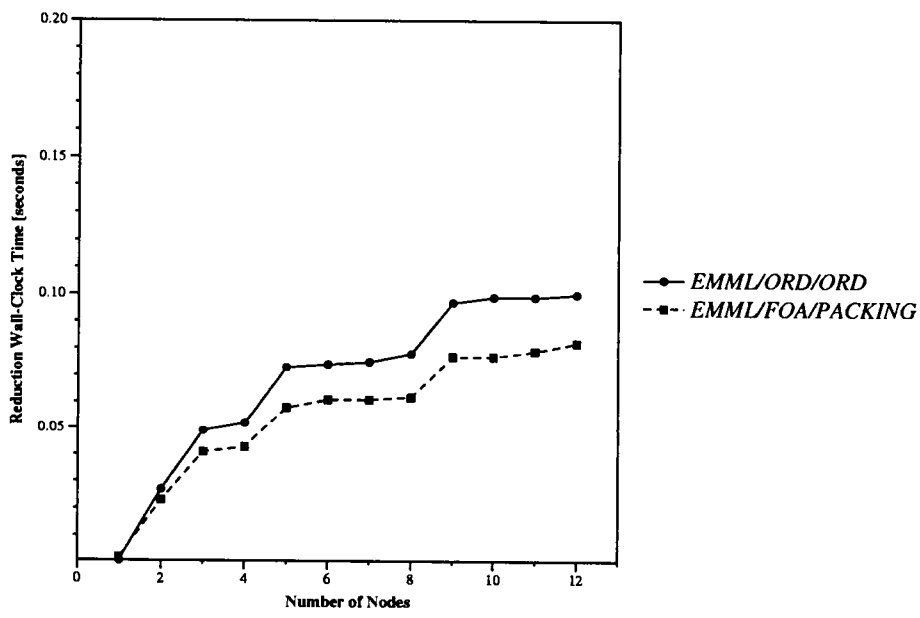
2.4.2 Focus-of-Attention Global Reduction Performance

Focus-of-attention, with packing, can be used to reduce per-iteration communication times. The packing operation uses the λ-mask to send and operate on a subset of the data; that is, data being *focused* on. At levels where the focus-of-attention scheme works well, packing can be quite effective — 90% compression for a brain image reconstruction at normal resolution. Packing provides significant savings in per-iteration communication time as can be seen in Figure 2.4-a and 2.5-a. As ordinary EM-ML begins to experience performance degradation, the packing of $\mathbf{M^k}$ keeps the communication impact on the reconstruction process very low.

However, as can be seen in Figures 2.4-b and 2.5-b where image reconstruction is performed at three times the normal resolution, packing becomes less effective which results in higher communication costs on both ATM and Ethernet. So the

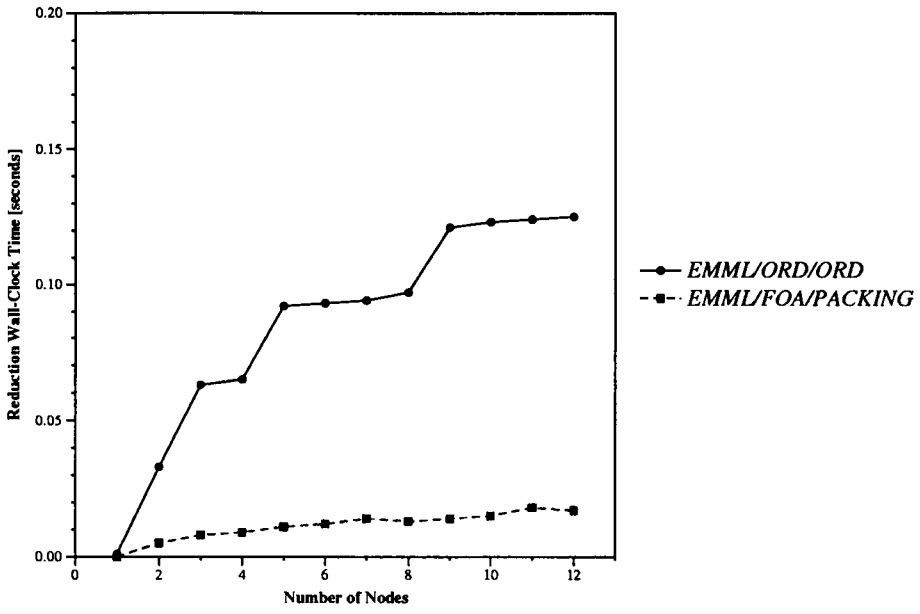


a) EM-ML Performance at normal resolution.

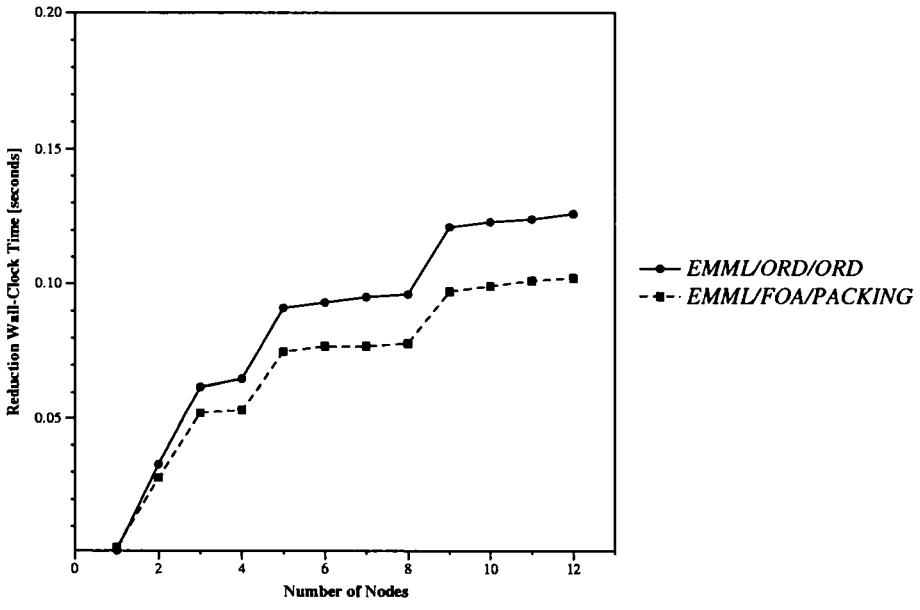


b) EM-ML Performance at 3 x normal resolution.

Figure 2.4: Performance of EM-ML on the ATM. The brain image was reconstructed at normal resolution in (a) and 3 x normal resolution in (b).



a) EM-ML Performance at normal resolution.

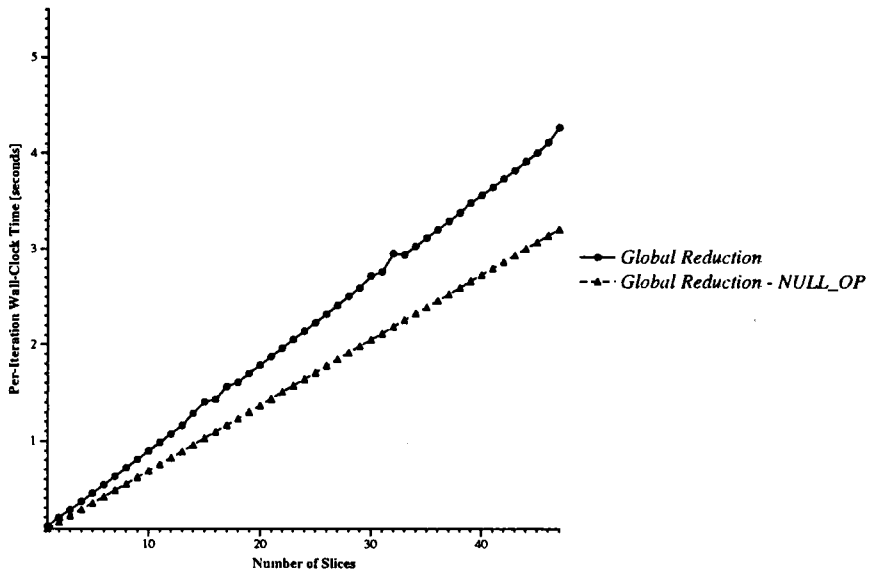


b) EM-ML Performance at 3 x normal resolution.

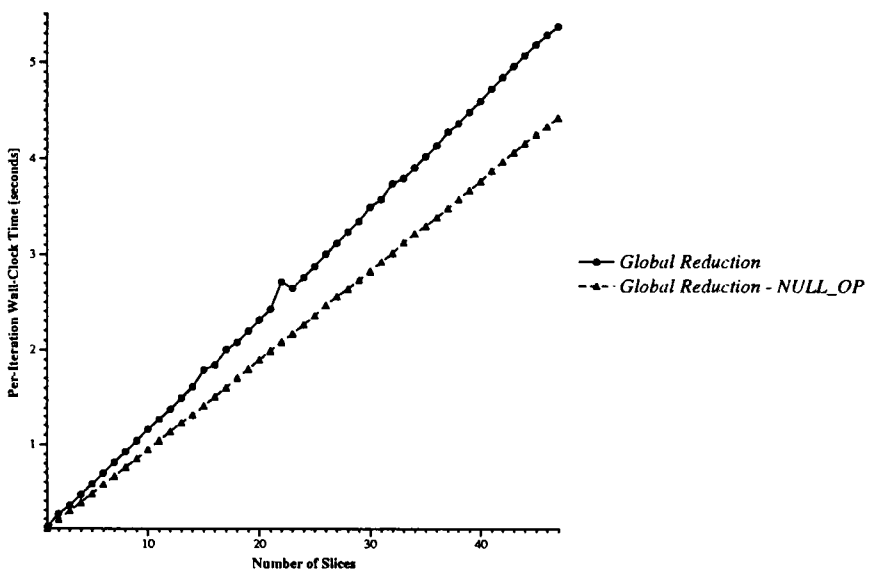
Figure 2.5: Performance of EM-ML on the Ethernet. The brain image was reconstructed at normal resolution in (a) and 3 x normal resolution in (b).

impact of packing on reducing communication time is dependent upon the boundaries established by focus-of-attention on the sinogram object(s) to be reconstructed. However, traditional data compression methods are not limited in this way.

As packing becomes ineffective, the global reduction performance returns to the level of ordinary EM-ML (figures 2.4-b and 2.5-b). While there may not be a significant impact on the high speed networks these experiments were conducted on, the same is not necessarily true for slower speed networks or in the case of moving to volume image reduction. Volume image reduction consists of 47 2D slices. Thus, as a volume, $\mathbf{M^k}$ would grow in size from 128 KB to 5 MB, assuming each term is a double-precision number taking 8-bytes. To get an idea of performance under such memory constraints, an experiment was run on both the ATM and Ethernet networks whereby a global reduction was run using a 12-node configuration for 32 iterations, and executing a summation operation on synthetic data for $\mathbf{M^k}$. The results are shown in Figure 2.6. The y-axis shows per-iteration time, and the x-axis shows the number of slices used during the reduction. Both the ATM and the Ethernet networks demonstrate linear performance indicating that the networks are not being saturated. The lines denoted by "Global Reduction NULL_OP" demonstrate performance for performing global reduction without actually executing the operation (summation). The difference between the nor-



a) Synthetic Volume Performance - ATM.



b) Synthetic Volume Performance - Ethernet.

Figure 2.6: Performance of global reduction with large data sets. All runs were conducted on a 12-node configuration for 32-iterations.

mal global reduction and the NULL_OP is the amount of CPU time (on average) the nodes are spending performing summation. Hence, most of the global reduction time is spent communicating.

Chapter 3

LOSSLESS DATA COMPRESSION

Data compression is a branch of information theory which relies on inherent redundancy in an information source to derive an alternative coding using less space. Data compression consists primarily of modeling combined with coding. The fundamental purpose of modeling is to generate symbol probabilities for use by a coding method. The fundamental purpose of a coding method is to optimally leverage the symbol probabilities given by a model of an information source into an alternate encoding which approaches the entropy of the source.

3.1 Statistical Compression Models

Statistical compression involves using information source statistics, such as frequency of occurrence, for coding. Symbols within an information source may be considered by a statistical model independently (for example, finite context models) or dependently (for example, Markov processes). The term order-n will be used to denote the context of a model. Order-0 modeling examines the current symbol regardless of context. Order-1 modeling examines the current symbol, s_i, within the context of the previous symbol s_j, $P(s_i|s_j)$.

Statistical models may be either static, semi-adaptive, or adaptive. Static methods use previously derived information source statistics as a model for coding by off-line analysis. Semi-adaptive models pre-scan the input source to derive statistics requiring a two-pass algorithm. Adaptive techniques adjust the probabilities as the input source is scanned and coded (or decoded) by starting with a *zero-frequency* assumption on all symbols. As symbols are processed, the model is updated.

In order to achieve compression, a statistical model needs to do two things [20]:

1. accurately predict the frequency/probability of symbols in the input stream.

2. generate symbol probabilities which deviate from a uniform distribution.

The accurate prediction capability of models is essential to the coding process. Coding methods attempt to assign smaller codewords to more frequently seen data

patterns. Thus, the better the model predicts, the more likely an encoding method will achieve higher compression. But, the information source must deviate from a uniform distribution; otherwise, the encoding process assignment of codewords will tend to average out and produce minimal compression, or worse, expansion of the information source.

3.1.1 Entropy and Coding

Entropy is a measure of the amount of uncertainty in an information source [1]. Considering an information source, S, as a collection of symbols $s_1, s_2, \ldots, s_q$ with probabilities of occurrence $p_1, p_2, \ldots, p_q$, then for any discrete probability distribution, entropy may be calculated as:

$$H(S) = -\sum_{i=1}^{q} p_i \log p_i \tag{3.1}$$

The function $H(S)$ measures the amount of *information* contained in the probability distribution p_i. The function $H(S)$ (or entropy) is important in that it provides a lower-bound on the average codeword length for a given information source and probability distribution. Hence, the closer a compression method can get to $H(S)$, the smaller the codewords will be on average and better compression will be achieved.

For encoding schemes using fixed-length or variable-length codewords, the average

codeword length, $\bar{L}_c$, is given by:

$$\bar{L}_c = \sum_{i=1}^{q} p_i c_i \tag{3.2}$$

where c_i is the codeword length, in bits, associated with the source symbol s_i.

Redundancy is the measure of efficiency of an encoding [23]. Redundancy is defined as the difference between the entropy of the input source and the average length of the associated encoding. Thus the formulation of redundancy, R, becomes

$$R = \bar{L}_c - H(S) \tag{3.3}$$

3.1.2 Semi-Adaptive Huffman Coding

Huffman coding is one of the most widely known and used methods in data compression. Huffman coding was first presented in [13] and is also discussed in [11, 2, 20, 23]. The Huffman algorithm generates variable-length codes with an integral number of bits. The generated codes also have the property of *unique decodability.* This provides perfect reconstruction of the input source by ensuring that resulting sequence of codewords can be decoded in one, and only one, way without the need for end-markers. More specifically, Huffman codes are prefix codes. That is, no code is a prefix of any other code in the codeword set.

The Huffman algorithm builds a binary coding tree, derived from source statistics,

for use in generating the variable-length codewords. An important objective of the Huffman scheme is to assign longer codewords to low-probability input symbols and shorter codewords to high-probability input symbols. This is reflected in the construction of the coding tree. The coding tree is constructed from the bottom up. The tree construction algorithm essentially works as follows:

1. First designate all input symbols as weighted leaf nodes of the tree. The weight is simply the frequency or probability of the source symbol. All the leaf nodes are marked as *free.*

2. Find the two free nodes with the lowest weights.

3. These two nodes are combined into one by creating a parent node and assigning the parent node a weight equal to the sum of the weights of the selected nodes.

4. One branch of the parent node is assigned the value of the zero-bit and the other branch the one-bit. This assignment is done consistently throughout the tree construction.

5. The two selected nodes are marked as no longer free.

6. The parent node is marked as a free node.

7. Repeat steps 2 - 6 until only one free node is left. This free node is designated the root of the tree.

Once completed, the binary tree contains all the variable-length codewords. A codeword for a given source symbol is contained in the unique path from the root node to the leaf node associated with the source symbol. Since the coding tree is not optimized for symbol lookup, another fast search data structure is usually constructed from the coding tree. Decoding (or expansion) works by looking up the symbol based on the codeword whereby the individual bits are used to navigate the code tree.

The Huffman algorithm has been shown to generate minimum-redundancy codes [11]. Given the probability distribution of an information source, the Huffman scheme produces the shortest possible average code length, $\bar{L}_c$, for any integral bit-width coding scheme. In fact, the redundancy of Huffman codes is bounded. If p_{max} is the highest probability of S, then for $p_{max} < 0.5$ the Huffman code is bounded by the half-open interval $[H(S), H(S) + p_{max})$; otherwise, the Huffman code is bounded by $[H(S), (H(S) + p + 0.086))$ [23].

The semi-adaptive Huffman coder used for experimentation was adapted from [20] and uses a two-pass implementation with an 8-bit symbol length, order-0 model. The first-pass derives the probability distribution from the input source. The second-pass performs the encoding based on the symbol frequencies generated from the first-pass. Decoding (expansion) is a one-pass operation. An alternative is to generate the frequency statistics off-line. However, the two-pass implementa-

tion was chosen since the probability density function for $\mathbf{M^k}$ changes with each iteration as each term gets closer to 1.0. In addition, the frequency distribution for $\mathbf{M^k}$ can be easily skewed by numerous zero values depending on the level of reconstruction.

3.1.3 Adaptive Huffman Coding

Adaptive Huffman coding is a data compression method where no prior probability of the information source is derived. All symbols in an information source are assumed to have zero-frequency. Thus, unlike semi-adaptive Huffman coding, the binary coding tree is constructed and updated as the information source is read. Both compression and expansion start with the same model; i.e., zero-frequencies. By starting with the same model and updating the model and coding tree, no encoding information need be transmitted with a compressed stream.

The coding tree used in adaptive Huffman coding differs from semi-adaptive Huffman in that a *sibling property* is preserved. Every node in the tree has a weight and every node except the root node has a sibling, that is, another node which shares the same parent node. Leaf nodes are weighted with the probability of occurrence of the associated symbol and the interior nodes are weighted by the sum of the weights of the two sibling nodes. If the nodes at each level in the tree are ordered by increasing weight and if every node appears adjacent to its sibling, then the tree exhibits the sibling property [20, 7].

After a new symbol is read and encoded (or decoded) the tree is updated by bumping the frequency count of the node representing the symbol. In order to preserve the sibling property, the affected leaf node may need to be exchanged with another leaf node or possibly moved up the tree. The overall effect will be that leaf nodes with higher frequencies will move closer to the root node, and thus have shorter codewords, while lower frequency leaf nodes will migrate farther from the root node and have longer codewords.

Adaptive Huffman coding suffers from i) starting with less than optimal statistics, and ii) costs associated with updating the coding tree. The Adaptive Huffman coder used for experimentation was adapted from [20]. As will be shown later, the costs associated with maintaining the coding tree in the adaptive Huffman algorithm are too costly for use in global reduction.

3.1.4 Splay-prefix Coding

The splay-prefix algorithm is an adaptive data compression method which uses a splay tree data structure for prefix-coding. The splay-prefix coding mechanism can be used with different compression schemes such as Huffman or arithmetic coding [15]. The algorithm considered in this research uses a splay-prefix coding tree in a manner similar to the Huffman coding algorithm described in Section 3.1.2.

Splay trees were originally presented as a form of self-balancing binary tree in [25] and [26]. When a node in a splay tree is accessed, the tree is splayed by making the accessed node the root node through a series of rotations. It has been shown that this produces an amortized search cost of $O(\log n)$ on an n-node tree. Single searches could take as long as $O(n)$. However, the splay-prefix algorithm uses a bottom-up compression technique so that worst case search times are not an issue.

Splaying typically applies to trees with data stored in the internal nodes and leaf nodes. But a prefix-code tree stores symbol information in the leaf nodes. The splay-prefix algorithm uses a semi-splaying technique which works with prefix-code trees. With this technique, the target node is not moved to the root of the tree; instead the path from the target node to the root node is reduced by a factor of two. In addition, the lexicographic ordering constraint is eliminated. By removing this constraint, the rotation operation may be simplified to only swapping two links in the tree and is referred to as a semi-rotation. The semi-rotation operation maintains the same distance reduction as would be achieved with a full rotation.

Although semi-splaying does not necessarily maintain a well balanced tree, the amortized search cost of $O(\log n)$ is still achieved within a constant factor. The semi-splaying operations move more frequently accessed nodes closer to the top of the tree and less frequently accessed nodes towards the bottom. By assigning

bit-codes to the left and right branches of the tree, the splay tree provides a prefix-coding data structure. Hence, codewords are derived by performing a bottom-up search. Likewise, expansion of codewords is done top-down using the codeword bit pattern as a navigator to a leaf node.

In splay-prefix coding, both the compression and expansion functions start with the same zero-frequency model. Thus, no coding information need be transmitted between compression and expansion. The splay-prefix tree is initialized as a balanced tree with N leaf nodes, where N is the number of source symbols. As symbols are compressed, or codewords expanded, the splay semi-rotation operations are performed. Note that the transformations performed by compression and expansion are identical for a given information source, since the splay-prefix tree is initialized in the same manner. Although the splay-prefix coding algorithm does not achieve optimal compression, the performance of splay-prefix coding is comparable to Huffman coding but does degrade on high-entropy sources, as will be shown later.

3.1.5 Arithmetic Coding

Arithmetic coding is an adaptive data compression method which replaces a group of input symbols with a single real number that falls in the interval $[0, 1)$. This single number can then be used to perfectly reconstruct the original group of input symbols. It has been shown that, in principle, arithmetic coding matches

the entropy (in terms of bits/symbol) for a message with respect to the model used for coding [2], whereas Huffman coding is optimal when symbol probabilities are integral powers of $\frac{1}{2}$ (that is, $2^{-1}, 2^{-2}, \ldots$).

In arithmetic coding, symbols are assigned to subintervals within the $[0, 1)$ range. Each subinterval is sized to match the probability of the symbol. Thus the subinterval assignment represents a cumulative frequency distribution of the information source. The cumulative symbol probabilities are used by the encoding process to continually restrict the range $[0, 1)$ into successive subinterval ranges, that is subintervals within subintervals. The decoding process uses the symbol probabilities to then expand the the subinterval ranges into a value which lies within a subinterval associated with a specific symbol.

Let r_L and r_H denote the low-end and high-end of the encoding range, respectively. Initially, arithmetic encoding starts with $r_L = 0$ and $r_H = 1$ representing the half-open interval $[0, 1)$. As each symbol, s_i, is processed the range is narrowed proportionally by the subinterval for s_i. The final value of r_L can then be used by the decoding process as the decoder does not need to know the low and high values of the final range. A special end-of-coding symbol is used so the decoder knows when to stop. For each symbol read, the encoding process is given by:

$$r_H = r_L + F_H \times (r_H - r_L) \tag{3.4}$$

$$r_L = r_L + F_L \times (r_H - r_L) \tag{3.5}$$

where F_L and F_H denote the low-end and high-end cumulative frequencies for the symbol s_i, respectively.

The decoding process does not need to know the low and high ranges used by the encoder. Just as in the encoding process, the decoding process starts with $r_L = 0$ and $r_H = 1$. Given an arithmetically encoded message, V, the decoding process repeatedly checks to see which subinterval V lies in. A symbol, s_i, is decoded by finding an s_i which satisfies the predicate:

$$\exists s_i (F_L \leq \frac{(V - r_L)}{(r_H - r_L)} < F_H) \tag{3.6}$$

There can be only one s_i which satisfies the predicate. Once s_i is found, the decoder outputs the symbol and then updates r_L and r_H just as the encoding process does in Equations 3.4 and 3.5. Also note that the fraction, $\frac{(V-r_L)}{(r_H-r_L)}$, in the inequality of (3.6) is a subinterval range expansion which is proportional to the probability of each symbol as it is extracted. Thus, Equation 3.6 reverses the effects of the encoding process. That is, just as the encoding process continually

reduces the subinterval range with each symbol encoded, the decoding process expands the subinterval ranges. The decoding process stops when the end-of-coding symbol is decoded.

Arithmetic coding implementations actually have several factors which can complicate the simplicity of the encoding and decoding processes described above. Such factors include incremental transmission and reception and fixed-precision arithmetic. The precision required to represent the range values grows with the length of the message. This quickly becomes a problem since computers do not have infinite precision in floating-point number operations. Implementations deal with this through a combination of incremental transmission (blocking the input stream) and using integer arithmetic with shifting operations. An extensive treatment of implementation issues for arithmetic coding can be found in [28], [2], and [20].

Arithmetic coding supports static, semi-adaptive, and adaptive modeling. The implementation used for experiments in this thesis uses adaptive modeling.

3.2 Dictionary Compression Models

Dictionary-based compression schemes attempt to incorporate the structure of data composing the information source in order to achieve compression. These methods build lists of recurring data patterns and use tokens, or list indexes, to

encode the data.

Dictionary-based schemes support static, semi-adaptive, and adaptive methods of compression. Static methods are pre-compiled dictionaries which do not change during compression or expansion. Digram coding is an example of using a static dictionary. Semi-adaptive methods typically perform two-passes over the information source and create a dictionary which must be transmitted with the encoded stream. Adaptive dictionary-based algorithms encode variable-length strings of symbols as tokens. The tokens form pointers into phrase dictionaries. The idea is to create tokens that are shorter in length than the strings which they replace. The decoding process is then reduced to expanding the tokens with the dictionary phrases.

The early works of [18] and [19] provide the foundation of most adaptive dictionary encoders today. This family of algorithms is referred to as LZ coding. There are two primary branches of LZ coding referred to as LZ77 and LZ78. These two methods differ in fundamental ways. Each method has different implementation issues, strengths, and weaknesses. A good review of dictionary-based methods and the LZ family of coding algorithms may be found in [28].

3.2.1 LZ77 Dictionary Coding

The LZ77 family of algorithms uses a fixed-length phrase dictionary where the phrases are derived from the symbols found in a fixed-length sliding window — a search window and a look-ahead window. The search window is typically thousands of bytes in length while the look-ahead window is generally smaller. The LZ77 algorithm attempts to match the contents of contiguous symbols (strings) in the look-ahead window with strings in the search window. Tokens are emitted in place of strings in the look-ahead buffer. A token may represent a matching string or a single symbol. In any case, each token emitted consists of the tuple $< o, l, C(s_i) >$, such that o is the offset from the first symbol of the string in the look-ahead window to the corresponding first symbol in the search window, l is the length of the match, and $C(s_i)$ is the code of the first symbol that follows the string in the look-ahead window. After emitting a token, the windows are advanced $l + 1$ positions. For expansion of the compressed source, the LZ77 algorithm simply expands each token by emitting the string referenced by $o + l$, expanding $C(s_i)$ and then advancing the windows $l + 1$ positions. In comparison, expansion tends to be much faster than compression as compression requires efficient search methods to achieve better performance, whereas expansion consists primarily of indexing and copying operations.

A problem for LZ77 encoding is in determining the window size. If the window

is too small, the encoding may not match recurring periodic patterns where the period is longer than the size of the window. Simply increasing the window size is not always the solution since a larger window size increases the substring search times for find matching patterns. In addition to slower search times, the bit size of the offset and length pointers may also increase.

3.2.2 LZSS Dictionary Coding

The LZSS coding algorithm is a variation on LZ77 [3]. LZSS improves on LZ77 by adding an additional data structure to help manage phrases in the sliding windows and by using a binary tuple representation for tokens.

LZSS uses a binary search tree to store previously seen string matches as the search and look-ahead windows are shifted. This effectively reduces the search time for finding longest matching phrases to $(\log_2 W) \times l$, where W is the size of the search window.

LZSS defines a token as the tuple $< b : o, l >$. This tuple may be interpreted as either an offset/length pair, where o and l have the same meaning as in LZ77, or as a single symbol for output. LZSS uses a bit-field prefix, b, to distinguish which interpretation is needed. The LZ77 token uses a triple representation in order to handle the special case where no matching phrase is found. In this case LZ77 emits the token $< 0, 0, C(s_i) >$ to output the current symbol as-is. The LZSS

approach emits the token $< b : 0, C(s_i) >$ for a single symbol.

The LZSS coder used for experimentation was adapted from an implementation in [20]. Experiments revealed the LZSS coder was one of the slower compressors but had very fast decompression times. The compression ratios are comparable to Huffman, but the LZ78 approach to dictionary coding is generally faster.

3.2.3 LZ78 Dictionary Coding

The LZ78 family of algorithms do not use sliding windows, rather the input symbols are parsed into phrases, where each phrase is the longest matching phrase seen so far plus the current symbol. This approach effectively removes the assumption found in LZ77 that patterns occur close together. In LZ78 both the encoder and decoder must build and maintain identical dictionaries. In each case, the dictionaries consist of one entry, the null-string at index 0. The LZ78 encoder emits tokens as binary tuples of the form $< i, c >$ such that i is an index into the dictionary of the longest matching phrase and c is either the codeword for the current symbol if there is no matching phrase or an index into the dictionary of the symbol which immediately follows the matched phrase. This is semantically similar to LZ77 in that the token represents the currently matching phrase and the symbol which broke the match.

A problem for the LZ78 algorithm is that the dictionary grows without bound.

Thus, some method must be devised to mitigate an ever expanding dictionary. A simple method is to stop adding phrases to the dictionary once a size threshold is reached. The UNIX compress program monitors the compression ratio and when the ratio begins to deteriorate, the dictionary is expunged and the process of adding phrases starts anew [20]. Another problem with the LZ78 approach is that both the encoder and decoder must maintain identical dictionaries. In LZ77, only the encoder maintained a dictionary. LZ77 expansion is generally a simpler and faster process than in LZ78.

3.2.4 LZW Dictionary Coding

The LZW coding algorithm is a variation on LZ78 (see [27]). LZW improves on LZ78 by employing a technique for removing the necessity of c in emitted tokens. With this technique only the index of the matching phrase need be emitted. LZW requires that the dictionary be initialized with entries for all the input symbols, s_i, of an information source, S. This approach is an intuitive step when considering the following observations in LZ78: i) the first time a symbol is seen in the input information source it is added to the dictionary, and ii) when emitting a token for a phrase the dereference of c is always going to be a single symbol. Thus, priming the dictionary with all the input symbols allows the immediate coding of phrases rather than just individual symbols.

The coding process for LZW is very similar to LZ78. When encoding, some

number of input symbols are read which match an existing phrase in the dictionary. If we let the matching phrase be P_i and the symbol which breaks the match be s_j, then the encoder emits an index, i, for the matching phrase and then adds the new phrase $P_i + s_j$ to the dictionary, where $+$ is the concatenation operator. The decoding process is similar except that an index, k, is read as input and the phrase P_k is emitted. Then, in order to construct a dictionary identical to that used by the encoding process, the decoder must maintain a new phrase string, P_i. P_i is initially a null-symbol. After emitting P_k, the decoder concatenates P_k onto P_i, that is, $P_i = P_i + P_k$. If P_i exists in the dictionary, then the decoding of the input stream continues; otherwise, P_i is added to the dictionary and P_i is set back to the null-symbol. There is a single exception case to this algorithm. When the information source contains the sequence '$s_i + P_j + s_k + P_l + s_m$', the encoder will output a code which the decoder will not have added as a phrase to the dictionary. The decoder handles this special case by using the first symbol from the previously seen phrase and appending it onto the current phrase and adding it to the dictionary.

The LZW coder used for experimentation was adapted from an implementation in [20]. Experiments revealed the LZW coder was one of the fastest compressors but compression ratios suffer on high-entropy information sources.

3.3 Lossless Methods in Global Reduction

3.3.1 Compression Engine Performance

Tables 3.1 and 3.2 contain mean compression/expansion results for the various coding methods applied to $\{\mathbf{M^k}\}_{k=1}^{32}$ in a 12-node brain image reconstruction. All times are measured in seconds. The compression ratio, given in column *Ratio*, is calculated by $(100 - (out * 100/in))$ where *in* equals the input size of $\mathbf{M^k}$ and *out* equals the compressed size of $\mathbf{M^k}$. The input size of each $\mathbf{M^k}$ for the brain data set is 128x128 terms where each term is represented by an 8-byte value in IEEE floating point format for an input size constant of 128KB. The column labeled $Time_c$ gives the time to compress and the column labeled $Time_d$ gives the time to decompress. The $Time_t$ is simply the total time; that is, $Time_c + Time_d$. Table 3.1 provides performance data for a normal resolution brain image while Table 3.2 provides performance data for a reconstruction at three times normal resolution.

The first half of each table gives performance values for $\mathbf{M^k}$ generated using the focus-of-attention scheme. The second-half of the table, those columns marked with *, were generated using ordinary EM-ML. Note the extreme differences between using focus-of-attention and ordinary EM-ML for similar resolutions, and differences between the resolutions using similar schemes (that is, focus-of-attention and EM-ML). A fairly high-level of compression is achieved when compressing $\mathbf{M^k}$ at normal resolution with focus-of-attention applied. This is due

Table 3.1: Compression Results for $\mathbf{M^k}$ - Brain at Normal Resolution.

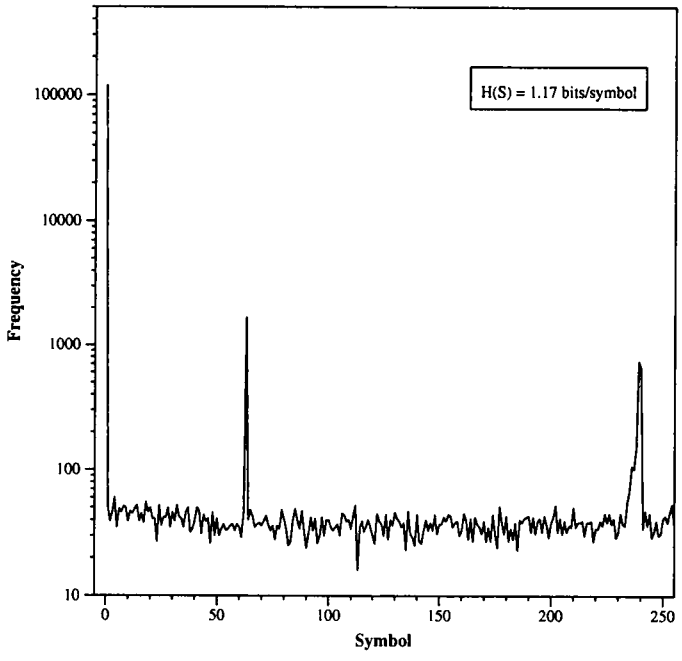
	$Time_c$	$Ratio$	$Time_d$	$Time_t$	$Time_c^*$	$Ratio^*$	$Time_d^*$	$Time_t^*$
SHUFFMAN	0.066	78.00%	0.068	0.134	0.159	09.12%	0.222	0.381
LZW	0.080	87.60%	0.062	0.142	0.238	-21.80%	0.238	0.476
SPLAY	0.096	78.00%	0.099	0.195	0.357	-9.06%	0.348	0.705
ARITHMETIC	0.149	89.78%	0.144	0.293	0.689	08.28%	0.811	1.500
AHUFFMAN	0.156	79.00%	0.158	0.314	0.425	09.09%	0.448	0.873
LZSS	0.267	78.94%	0.062	0.329	0.694	-2.69%	0.190	0.884

Table 3.2: Compression Results for $\mathbf{M^k}$ - Brain at three times Normal Resolution.

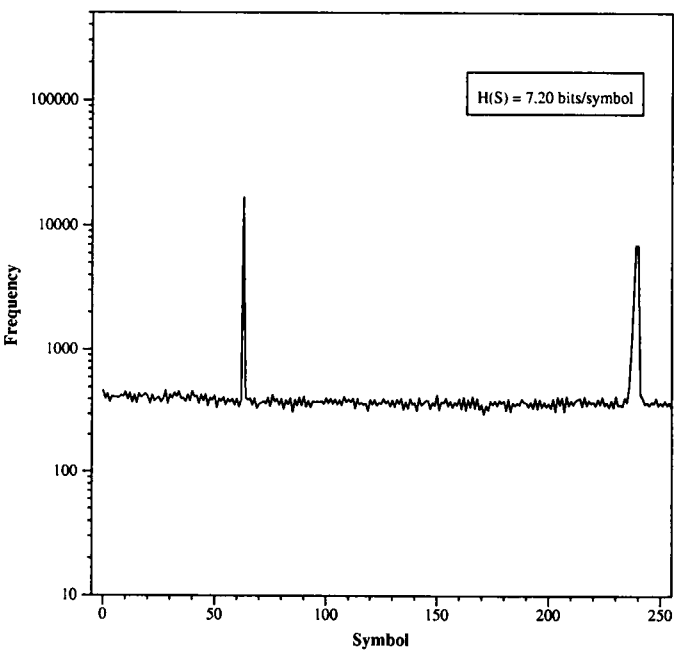
	$Time_c$	$Ratio$	$Time_d$	$Time_t$	$Time_c^*$	$Ratio^*$	$Time_d^*$	$Time_t^*$
SHUFFMAN	0.148	19.22%	0.202	0.350	0.160	09.34%	0.221	0.381
LZW	0.204	04.00%	0.196	0.400	0.242	-20.60%	0.236	0.478
SPLAY	0.297	10.78%	0.291	0.588	0.356	-8.38%	0.345	0.701
AHUFFMAN	0.379	19.47%	0.399	0.778	0.422	09.34%	0.445	0.867
LZSS	0.735	16.44%	0.160	0.895	0.696	-2.03%	0.191	0.887
ARITHMETIC	0.568	24.28%	0.663	1.231	0.677	08.75%	0.804	1.481

primarily to the large number of zero-byte values introduced by the focus-of-attention algorithm. The semi-adaptive Huffman coder provides the best total performance. But as the $\mathbf{M^k}$ vector becomes more dense, the effectiveness of entropy-coding quickly diminishes — with each method. Note that some algorithms actually expand the data. In the splay algorithm this is most likely due to the rotation of nodes on every access. Since the information source is fairly uniform, on average the codeword length is longer than the input symbol length. A similar effect is seen with the dictionary coding methods. In this case there is a large number of tokens generated for a correspondingly small string of symbols. Hence, the length of the token exceeds the length of the string of symbols and inflation occurs.

Figure 3.1 shows frequency graphs of $\mathbf{M^k}$ for both EM-ML with focus-of-attention (a) and without (b). With focus-of-attention, there is a significantly large number of zeros introduced. As is noted in Figure 3.1-a, the entropy for $\mathbf{M^k}$ is 1.17 bits/symbol. Thus, given an 8-bit input symbol length there is a large potential for compression. But note that with the exception of a couple of high-frequency spikes, the remaining symbols are fairly uniform. Figure 3.1-b shows $\mathbf{M^k}$ under ordinary EM-ML. In this case the vector $\mathbf{M^k}$ is more dense since much of the background noise has not been removed as it is under the focus-of-attention method. This is illustrated in Figure 3.1-b by the higher average symbol frequency as



a) Focus-of-Attention.



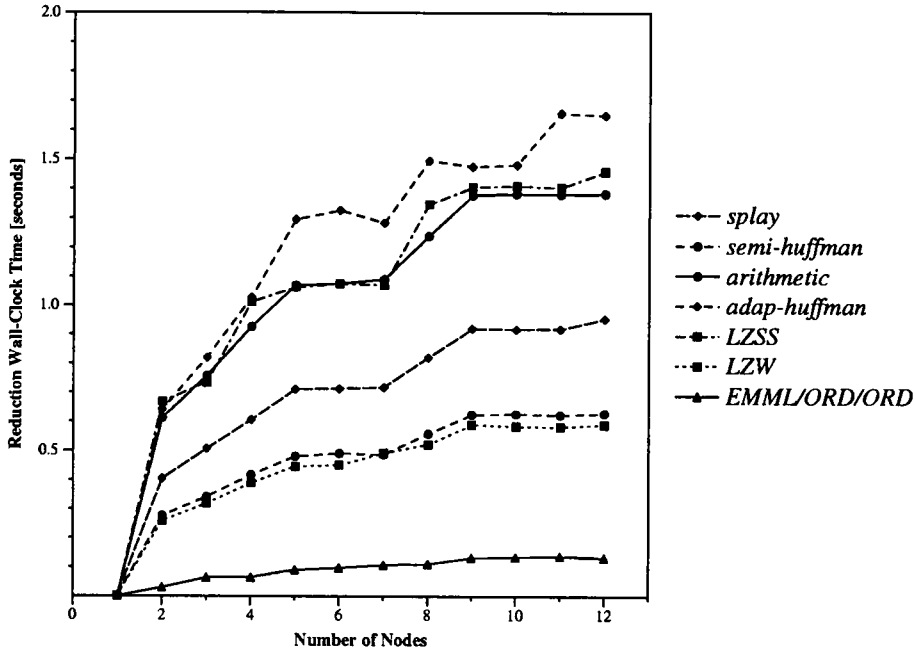
b) Ordinary.

Figure 3.1: Uniformity of $\mathbf{M^k}$ for (a) focus-of-attention and (b) Ordinary EM-ML.

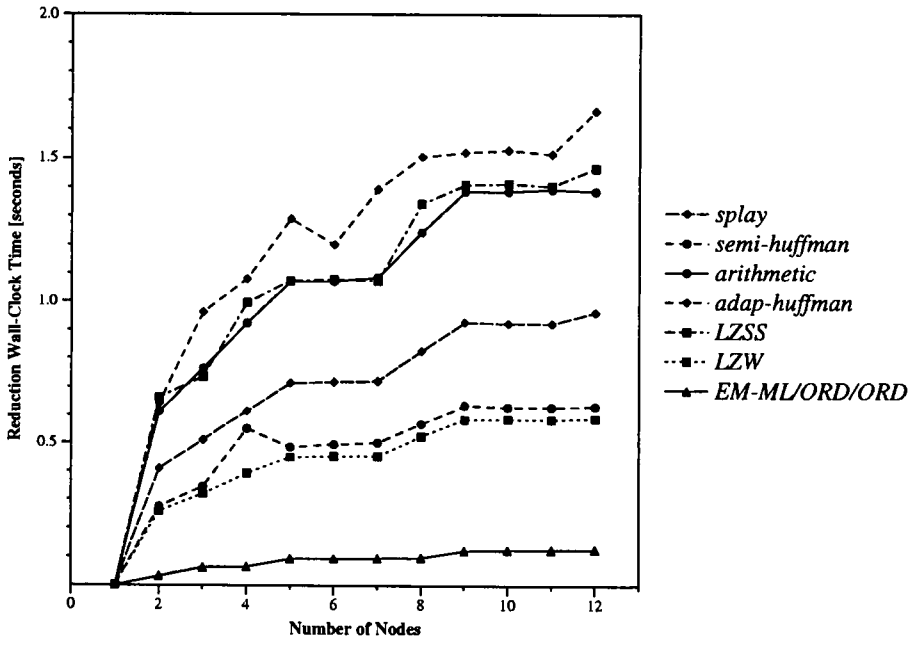
well as the calculated entropy of 7.20 bits/symbol. Thus, there is less potential for data compression. The same effects are seen at three times normal resolution even with focus-of-attention applied since the entropy of $\mathbf{M^k}$ is tied directly to the density of the vector. That is, entropy is inversely related to the density of $\mathbf{M^k}$.

3.3.2 EM-ML Performance Results

Figure 3.2 shows EM-ML global reduction performance for brain image reconstruction at normal resolution under different compression schemes for both ATM and Ethernet networks. The plot for EM-ML is *not* using focus-of-attention. The plots for the lossless methods are using the focus-of-attention method in order to achieve the high-compression ratios given in Table 3.1. The intent is to show that even when the worst case performance for EM-ML is compared to the best case performance of EM-ML with lossless methods, that is high compression ratios, the introduction of lossless methods do not improve global reduction times. The plots for semi-adaptive Huffman and LZW confirm these are the faster methods. LZW is slightly faster than semi-adaptive Huffman which may be attributable to the higher compression ratio achieved and thus less data to communicate. In any case, the lossless methods only serve to hamper the reconstruction process by introducing severe delay in per-iteration communication times.



a) Compression Performance at normal resolution - ATM.

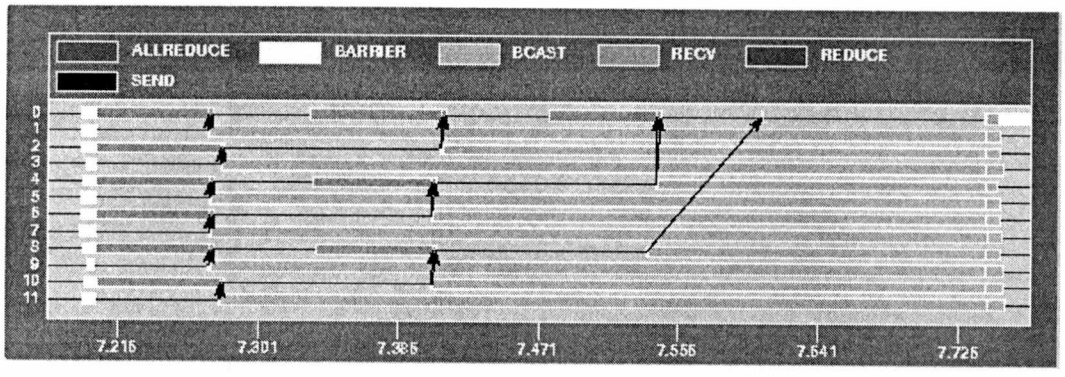


b) Compression Performance at normal resolution - Ethernet.

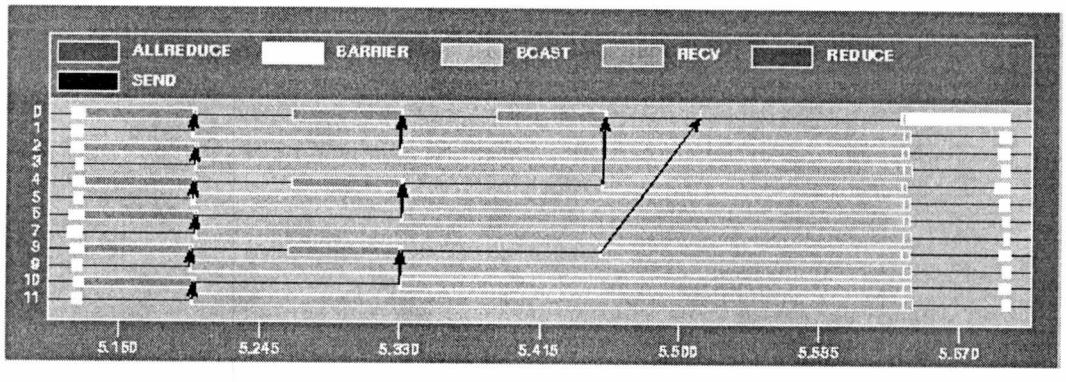
Figure 3.2: Performance of EM-ML with lossless compression on the ATM and Ethernet Networks. The brain image was reconstructed at normal resolution.

Figure 3.3 provides visual call-graphs for semi-adaptive Huffman (a) and LZW (b) coding. In both instances long periods of compression and decompression activity can be seen between send and receive calls thus prolonging the completion of the global reduction. Note that some of this time is spent summing terms but a much larger portion of time is spent in compression. Also note the near non-existence of send blocks. The send blocks are only detectable by the originating end of the arrows. This indicates a low utilization of the network as each node is spending more time compressing $\mathbf{M^k}$ to 10% - 20% of original size. However, on the high-speed networks the cost of compression and decompression is higher than just sending $\mathbf{M^k}$ as-is.

Experimentation reveals that there are two fundamental problems with applying entropy-coding compression algorithms to EM-ML global reduction operations. i) In order for an $\mathbf{M^k}$ to be summed it must first be compressed by the sender and then decompressed by the receiver. Thus, every summation operation between two nodes requires at least two compression operations. Recall from Sect. 2.3.2 that the total number of compression operations will be $2 * (N - 1) + N$. For a 12-node global reduction, the total compress/decompress cost for LZW would be $34 \times 0.134 = 4.556$ seconds not including actual communication time. ii) Whenever $\mathbf{M^k}$ becomes *dense*, the probability distribution tends to be uniform. This renders traditional entropy-coding methods ineffective. The focus-of-attention method



a) Semi-Adaptive Huffman.



b) LZW.

Figure 3.3: EM-ML MPI call graphs for global reduction during a brain sinogram image reconstruction at normal resolution on the ATM network.

helps to induce higher compression ratios due to the fact the area outside the region of interest is set to 0.0. However, this only helps in those cases when there is a relatively small region-of-interest.

Chapter 4

LOSSY DATA COMPRESSION

Lossy compression is an encoding of an information source with an average codeword length, $\bar{L}_c$, less than the entropy, $H(S)$. This form of encoding introduces distortion, or less than perfect reconstruction. The amount of acceptable distortion, or information loss, is based on some quality criteria. For EM-ML, experiments have shown that $\mathbf{M^k}$ can tolerate some data loss as long as $\boldsymbol{\lambda^k}$ is normalized to 1.0 after each iteration.

4.1 Transform Coding

Transform coding is described by [14] as a common method for utilizing linear dependencies in input data for efficient digitization (and compression). Transform coding algorithms achieve compression by combining a quantization strategy with a back-end lossless (error-free) compression algorithm, such as those described in

Chapter 3. The steps in transform coding can be summarized as follows:

1. Transformation: apply transformation algorithm projecting the input data onto the transformation space (domain). This is normally a lossless step.

2. Quantization: apply quantization (or thresholding) algorithm to transformed data and achieve mapping of many input values into a smaller set of output values. This step becomes lossy.

3. Entropy Coding: apply lossless compression to quantized results.

There are many transform-based coders and techniques, e.g., Discrete Cosine Transform, Fourier Transform, Wavelet Transform. But since these transforms require the use of a back-end lossless compression method (step 3) performance may be degraded. From Chapter 3, lossless methods encounter difficulty in both computation time and with high-entropy data sets. Applying a transform coding method which requires the three steps outlined above will only add to the CPU costs associated with lossless compression, even though the transformation of a high-entropy information source may result in a more highly compressible form in the transformed space (e.g., the frequency domain of the Fourier method). A lossy method which does not require the use of a lossless encoder is needed for application to distributed computations EM-ML.

4.2 Compression with Fixed-Point Arithmetic

The fixed-point arithmetic (FPA) approach involves a single pass over the input data, $\mathbf{M^k}$, and transformation to a fixed-point representation. The summation of the global reduction then uses fixed-point (integer) arithmetic rather than floating-point arithmetic. At the end of the global reduction operation, each node transforms the fixed-point result back to a floating-point format. Compression is achieved by reducing the space required to store each term. Assuming an 8-byte double precision input format, if a 16-byte short datatype is used as the output format, the compression ratio is 8:2 (75%). If a 4-byte integer datatype is used, the compression ratio is 8:4 (50%). These compression ratios will also be constant regardless of the size or composition of the input data. The FPA approach assumes $0.0 \leq M_b < 2.0$ for all b, and enforces this boundary restriction as it is known that as λ_b^k converges to a fixed point solution M_b^k goes to 1.0 [21].

The IEEE-754 standard format for representing double-precision numbers uses 1-bit for sign, 11-bits for the exponent, and 52-bits for the fraction with an implied one-bit [5]. This represents a floating-point number as:

$$\pm 1.\{frac\} \times 2^{exp}. \tag{4.1}$$

It should be noted that the sign bit should always be zero (+) for each term in $\mathbf{M^k}$ and that the exponent is biased (1023 for double).

The FPA method transforms numbers between the IEEE floating-point format and the custom fixed-point format, and accommodates the necessary normalization and alignments (see [8]). The conversion consists of taking n-bits from the fraction, adding the implicit one-bit in the proper position, and shifting based on the magnitude of the exponent. The amount of shift, and direction, is determined by subtracting the bias from the exponent. The result is a properly aligned term. This alignment step is necessary since different terms may have differing exponents and must therefore be aligned before performing addition. This is a lossy conversion because any fractional bits set between n+1 and 52 are lost, and the shifting operation may also introduce a loss in precision since there are only a fixed integral number of bits to represent the number. If properly controlled, this loss in precision does not severely affect convergence of EM-ML or image quality, as the experiments will show.

The enforcement of $0.0 \leq M_b < 2.0$ is applied during the initial conversion of floating-point values and during summation. During the summation of terms, overflow is detection is performed by comparing the current sum and the amount to be added. If the result is greater than 2.0, a maximum value of 1.9 is assigned.

The conversion from fixed-point back to double works by taking the fixed-point number, performing a re-normalization step, and then reconstructing a double precision number. To determine the magnitude of the exponent, the fixed-point

word is left-shifted until a 1-bit is in place in the implicit-one bit position. This shifting produces a shift-count which is then assigned, with bias, as the exponent of the re-constructed floating-point number (i.e., shift-count + bias). The fraction is then formed by the remaining bits in the fixed-point word.

4.3 Compression with Uniform Quantization

Scalar Quantization is the process of representing a large range of values by a smaller, possibly much smaller, set of values [23]. For application to global reduction EM-ML, the quantization process transforms a range of floating-point values into a range of integer codewords. The size of the integer may range from 1-byte to the largest supported word size of the computer architecture, depending on the number of levels used for quantization (see Equation 4.6). The quantization process consists of an encoder and decoder. The encoder divides the input range into intervals based on some boundary criteria. Each interval has a corresponding distinct codeword. Input values falling into a given interval will be assigned the codeword associated with that interval. The encoding step is lossy since there is a many-to-one relationship between an interval and a codeword. Thus, perfect reconstruction is not possible except for the most trivial of cases. The decoding phase converts codewords back into the input space. Since a codeword is only associated with an interval and not a specific value, a representative value must

be generated based on some criteria.

In order to quantize an input source at M intervals, $M+1$ endpoints must be specified. If we denote the decision boundaries (endpoints) of an interval by $\{b_i\}_{i=0}^{M}$, and the reconstruction (representative) values as $\{r_i\}_{i=1}^{M}$, and the quantization operation by $Q(\cdot)$, then

$$Q(s_i) = r_i \text{ iff } b_{i-1} \leq s_i < b_i. \tag{4.2}$$

Equation 4.2 indicates a bias to the lower boundary of an interval. This becomes important when reconstructing values from codewords and ensuring that the value of 0 is preserved, as EM-ML can be sensitive to noise.

For uniform quantization the interval (step) size is constant and is denoted by Δ. The uniform quantization codeword value, c, is obtained by

$$c = \lfloor (s_i - b_0)/\Delta \rfloor. \tag{4.3}$$

The codeword value is then also checked against the boundary condition $0 \leq c \leq M-1$. If $c < 0$, then $c = 0$. If $c > M-1$, then $c = M-1$.

The uniform quantization reconstruction value, r, is obtained by

$$r = \lfloor b_0 + c * \Delta \rfloor. \tag{4.4}$$

If values for M, b_0, and b_M are established, the interval size can be determined by

$$\Delta = (b_M - b_0)/(M - 1). \tag{4.5}$$

If values for b_0, b_M, and Δ are established, then the actual value of M can be determined by

$$M = (b_M - b_0)/\Delta + 1. \tag{4.6}$$

Uniform quantization achieves compression in EM-ML by using a codeword size which is less than the size of a double or single precision floating point number. Fixed-length codewords are used thus establishing a compression rate for the quantizer:

$$R = \lceil log_2 M \rceil. \tag{4.7}$$

Experiments have shown that R=4 (2-byte codewords) is a minimum to achieve an acceptable rate distortion measure. The value of R may need to become larger on parallel configurations with many nodes participating in a global reduction since each summation introduces quantization error and the number of summations in a global reduction increases proportionally with the addition of more nodes.

Although quantization does not provide perfect reconstruction, the quantization

step, $Q(\cdot)$, can be treated as a linear transform for sufficiently small Δ. Measurements of quantization error can be used to quantitatively determine how small Δ should be. Thus, once transformed into codewords, linear operators can be applied to the codewords (e.g., addition) with an expectation that once transformed back the reconstruction value will be reasonably close to the value that could have been achieved had the same linear operation been performed on the un-quantized source. From Equation 4.2 an upper-bound on the reconstruction error, E_r, can be determined by

$$E_r = P * \Delta. \tag{4.8}$$

where P is the number of operations performed on a quantized input. Thus, with quantization in global reduction the codewords are summed. The enforcement of $0.0 \leq M_b < 2.0$ is applied as it is with the fixed-point arithmetic method.

4.4 Distortion Measures and Image Quality

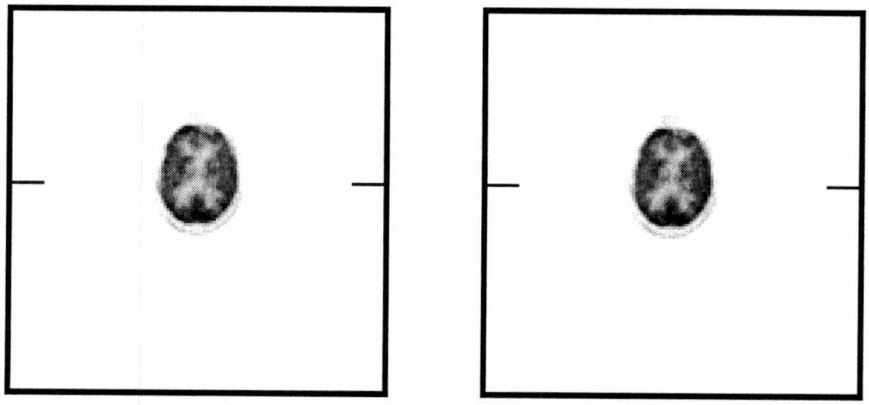
4.4.1 Brain Image Profiles

Brain image profiles provide a cross-sectional measure of emission activity for the different EM-ML methods. The profiles are generated from $\boldsymbol{\lambda}^{k=32}$ where each λ_b denotes the level of emission activity. Each λ_b value is scaled by the original observed emission levels, $\mathbf{n}^*$.

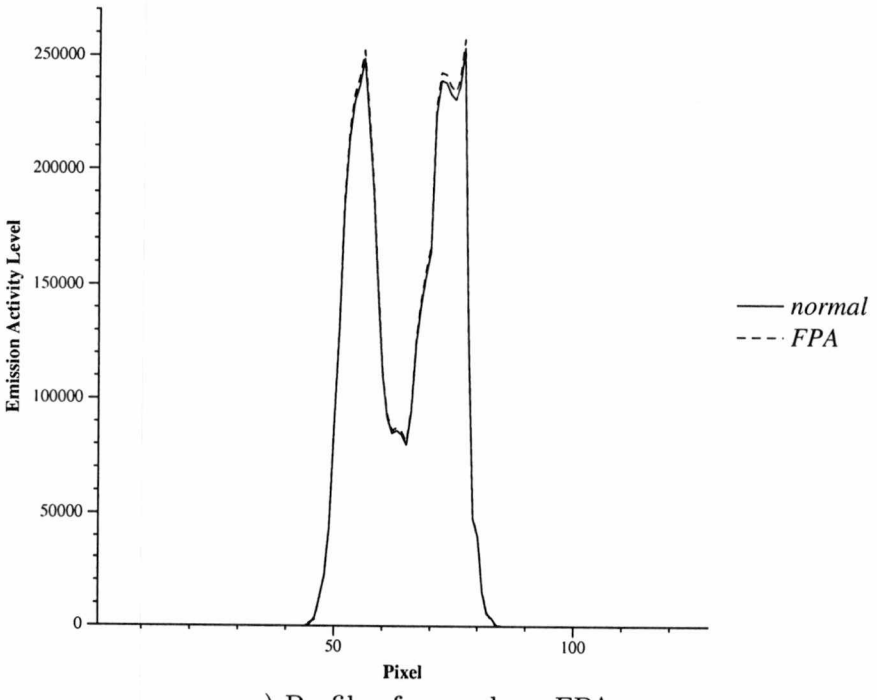
Figure 4.1 shows a comparison of a normal brain image and an EM-ML/FPA reconstructed brain image. The normal brain image, Figure 4.1-a, was reconstructed using EM-ML/FOA and normal resolution. The FPA brain image, 4.1-b, was reconstructed using 15-bits for the fraction and 1-bit for the digit to the left of the decimal point. Both images are the result of using 12 nodes in the image reconstruction process. As can be seen in Figure 4.1-c, the profiles for row 60 of each image are nearly identical in value. Slight deviations in the FPA-based image reconstruction may be detected at the peaks.

Figure 4.2 shows a comparison of a normal brain image and an EM-ML/FPA reconstructed brain image, both at three times normal resolution. The normal brain image, Figure 4.2-a, was reconstructed using EM-ML/FOA and normal resolution. The FPA brain image, 4.2-b, was reconstructed using 15-bits for the fraction and 1-bit for the digit to the left of the decimal point. A 12-node configuration was used for reconstruction of both images. As can be seen in Figure 4.2-c, slight deviations in the FPA-based image reconstruction may be detected at the peaks.

Figure 4.3 shows a comparison of a normal brain image and a quantized brain image. The normal brain image, Figure 4.3-a, was reconstructed using EM-ML/FOA and normal resolution. The quantized brain image, 4.3-b, was reconstructed using a global reduction quantizer with Δ=0.000030518044, b_0= 0.0, and b_M=2.0 M=65536(64K), and $\Delta = \frac{2.0}{65535}$. A 12-node configuration was used for reconstruc-

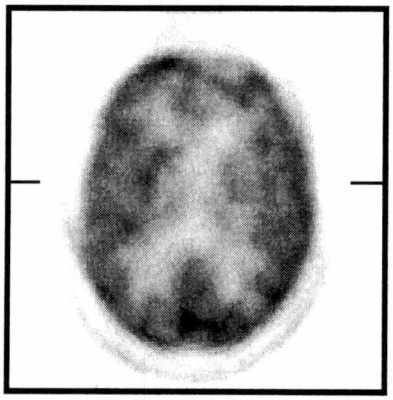


a) Normal. b) FPA.

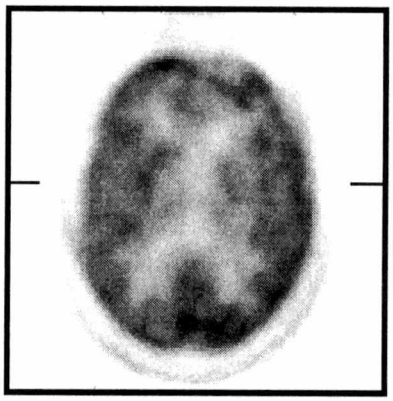


c) Profile of normal vs. FPA.

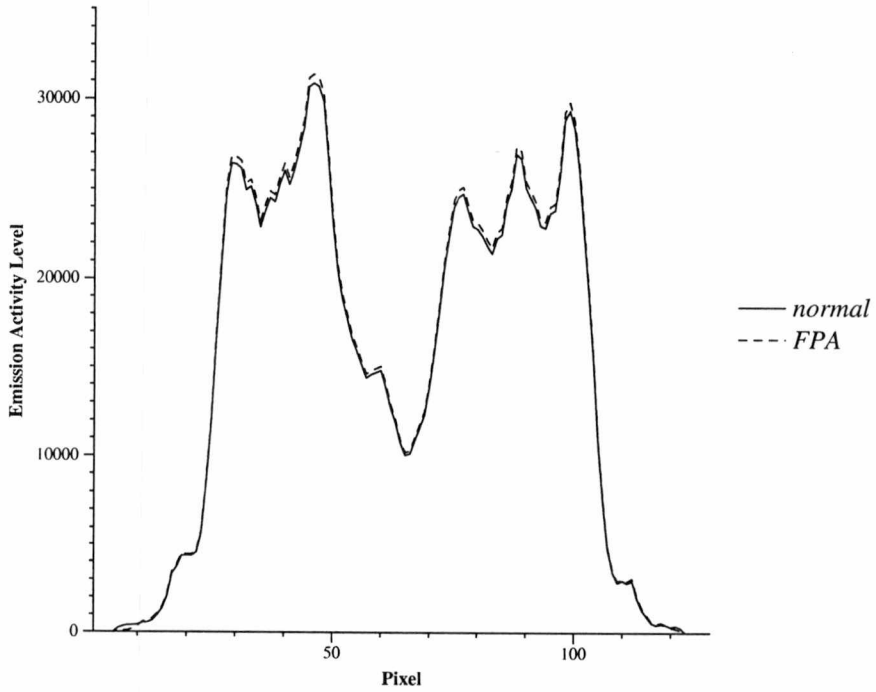
Figure 4.1: Normal and FPA Brain Images at Normal Resolution.



a) Normal.

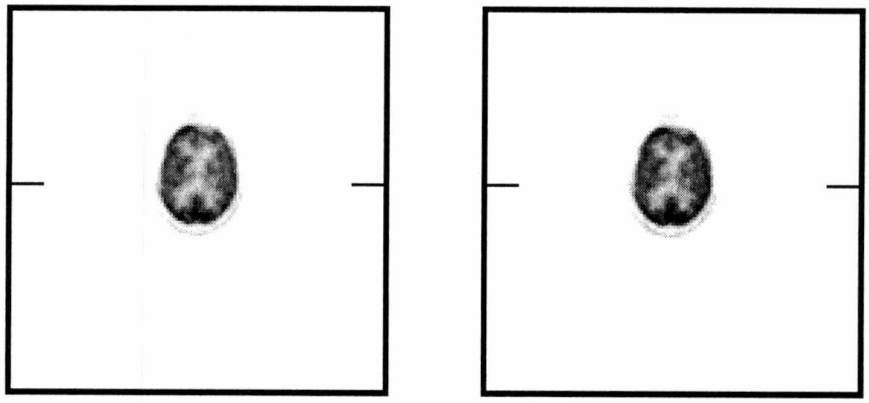


b) FPA.



c) Profile of normal vs. FPA.

Figure 4.2: Normal and FPA Brain Images at 3 x Normal Resolution.



a) Normal. b) Quantized.

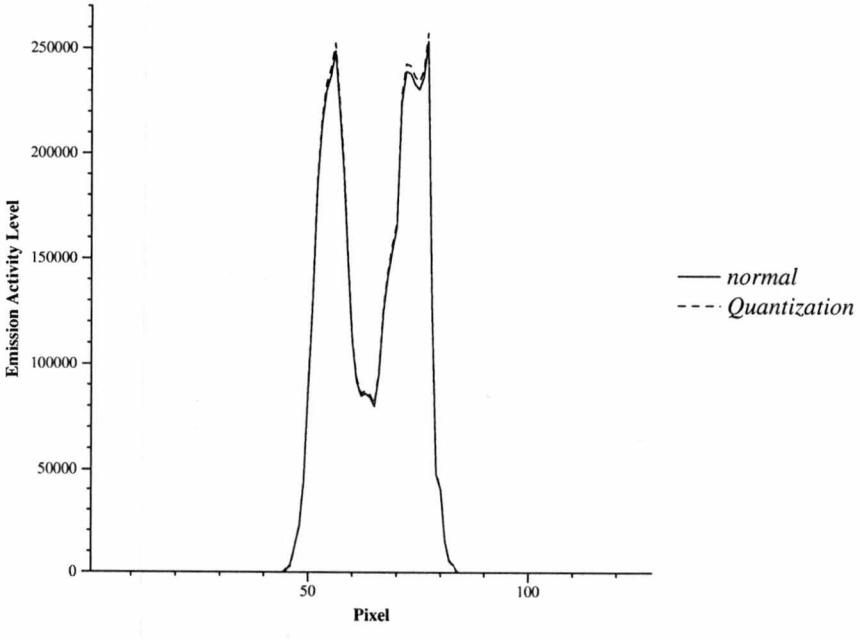


Figure 4.3: Normal and Quantized Brain Images at Normal Resolution.

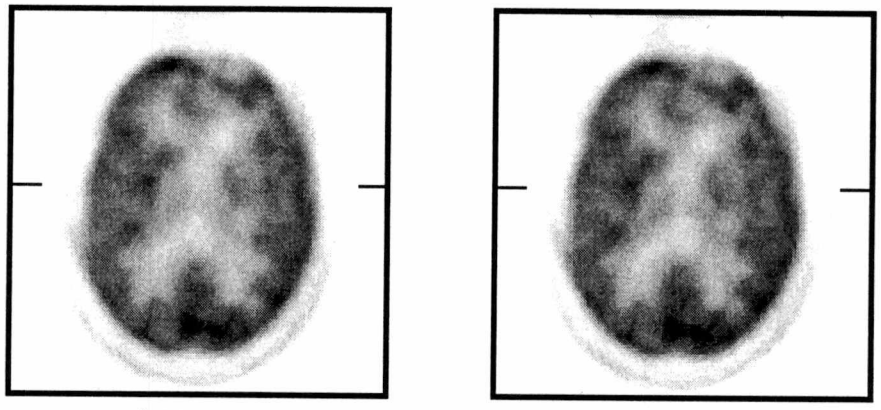
tion of both images. As can be seen in Figure 4.3-c, the profiles for row 60 of each image are nearly identical in gray-scale value. Just as in the FPA case, only near the peaks can the dashed line of the quantized reconstruction be detected.

Figure 4.4 shows a comparison of a normal brain image and a quantized brain image both reconstructed at three times normal resolution. The normal brain image, Figure 4.4-a, was reconstructed using EM-ML/FOA. The quantized brain image, 4.4-b, was reconstructed using a global reduction quantizer with b_0= 0.0, and b_M=2.0, M=65536(64K), and $\Delta = \frac{2.0}{65535}$. A 12-node configuration was used for reconstruction of both images. As can be seen in Figure 4.4-c, the profiles for row 60 of each image are nearly identical in gray-scale value. The quantized profile can be detected along the ridges of the top of the profile.

The difference in scale between normal resolution and three times normal resolution figures is due to the fact that at three times normal resolution 9 pixels corresponds to one pixel at normal resolution. Thus the emission activity is effectively spread across more pixels.

4.4.2 Region-of-Interest Measures

For quantitative assessment of the lossy methods, *region-of-interest* statistics (such as mean pixel value, standard deviation, and regional bias which is the relative difference between the observed and the true ROI mean pixel values) were gener-



a) Normal. b) Quantized.

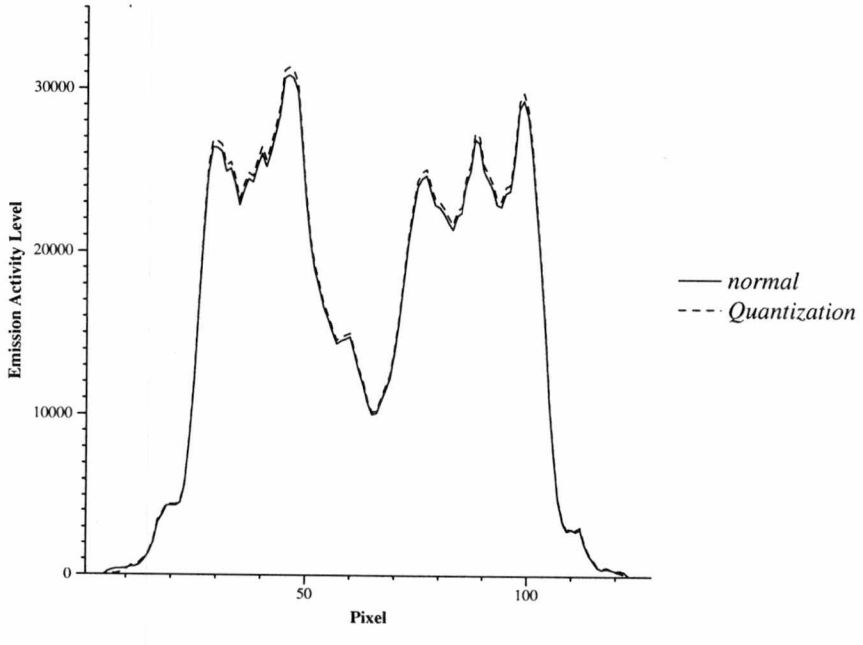


Figure 4.4: Normal and Quantized Brain Images at 3 x Normal Resolution.

ated. Figure 4.5 shows the simulated phantom which provides a noise-free sinogram obtained by projecting a version of the Shepp-Vardi phantom [24] via the **P**-matrix. The four ROI's are denoted by emission counts and assigned a numeric identifier. The four ROI's cover uniform pixel counts of 14208, 18870, 9472, and 2886.

Figures 4.6 and 4.7 plot the observed mean pixel value $\pm$ 1 standard deviation for each iteration between 2 and 256 iterations for the EM-ML reconstructions obtained with fixed-point arithmetic and uniform quantization, respectively. In each figure, the lossy reconstructions are plotted by $\triangle$. The true mean value for each ROI is denoted by the dashed horizontal line. The measured statistics for EM-ML/FOA reconstruction is plotted by $\bigcirc$. The standard deviations are practically indistinguishable for the different reconstructions, ROI's, and iterations. During the early iterations, the emission activity levels tend to deviate from the true values for all ROI's. However, given enough iterations, the measured mean values become very close to the true values for all ROI's. Based on these results, both the fixed-point arithmetic and uniform quantization approaches do not result in any quantifiable degradation for the ideal data.

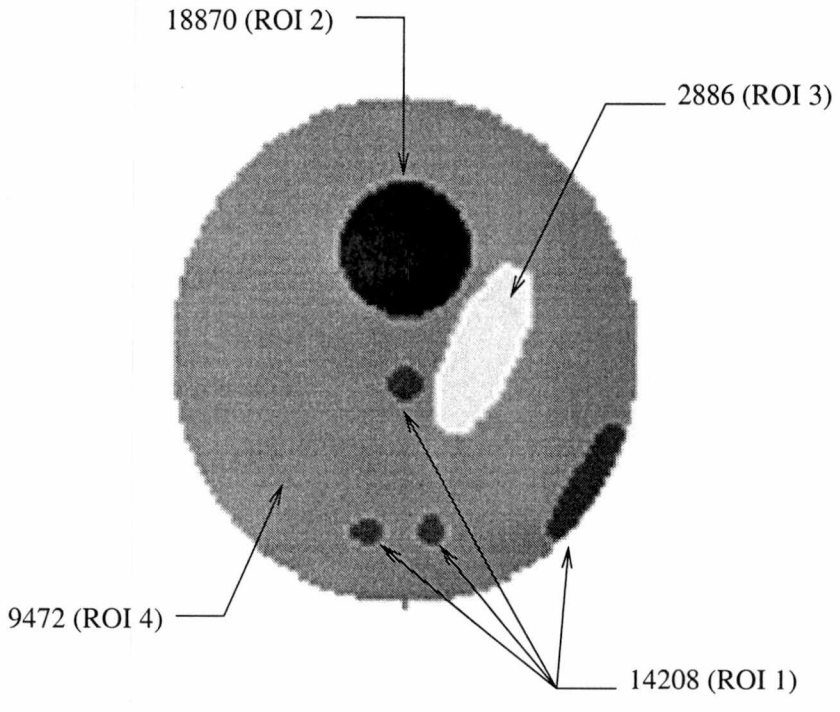


Figure 4.5: Simulated Phantom Data Set. Each emission level is noted by a numeric ROI identifier.

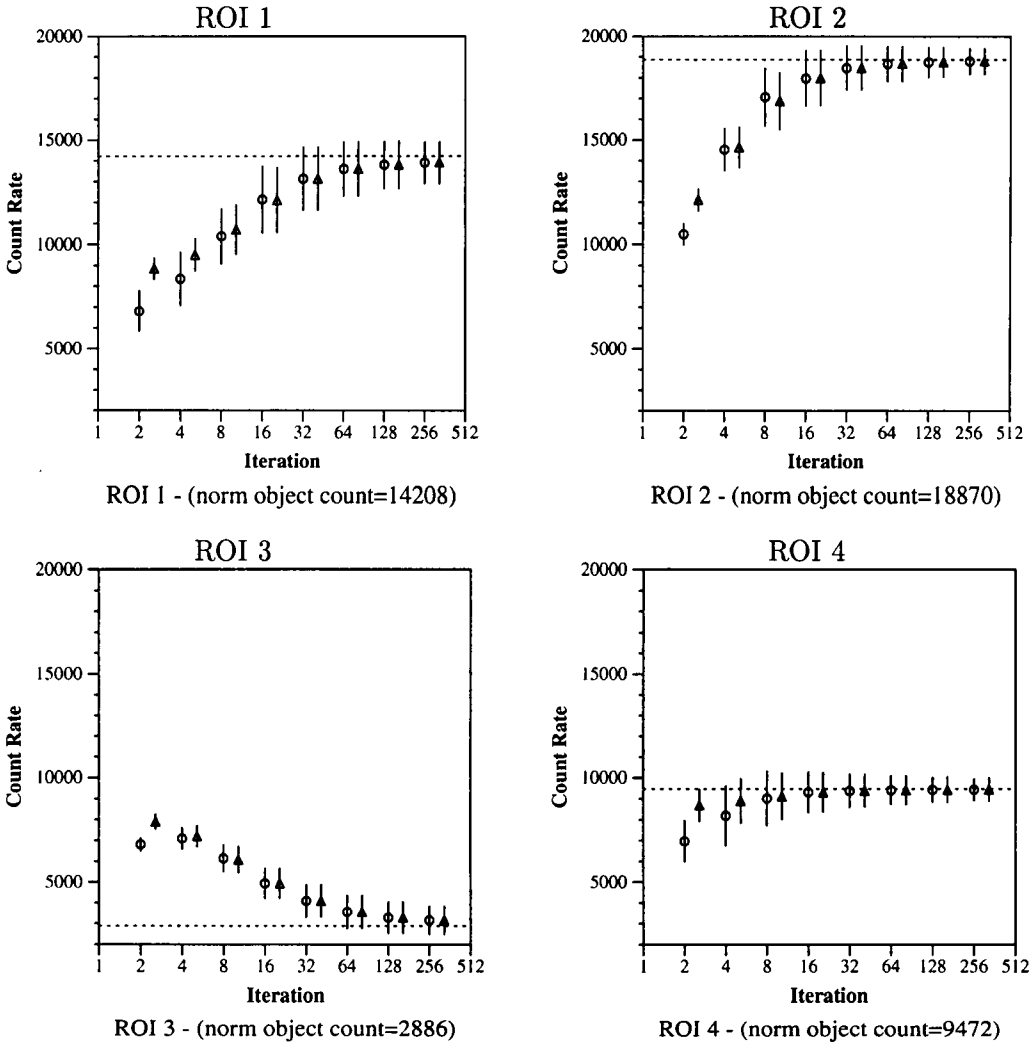


Figure 4.6: ROI Measures - Fixed-Point Arithmetic. Mean and standard deviation results of measuring the activity in each of the four ROI's defined for the simulated phantom. ○ and △ markers denote the results obtained with EM-ML/FOA and EM-ML/FOA with FPA, respectively.

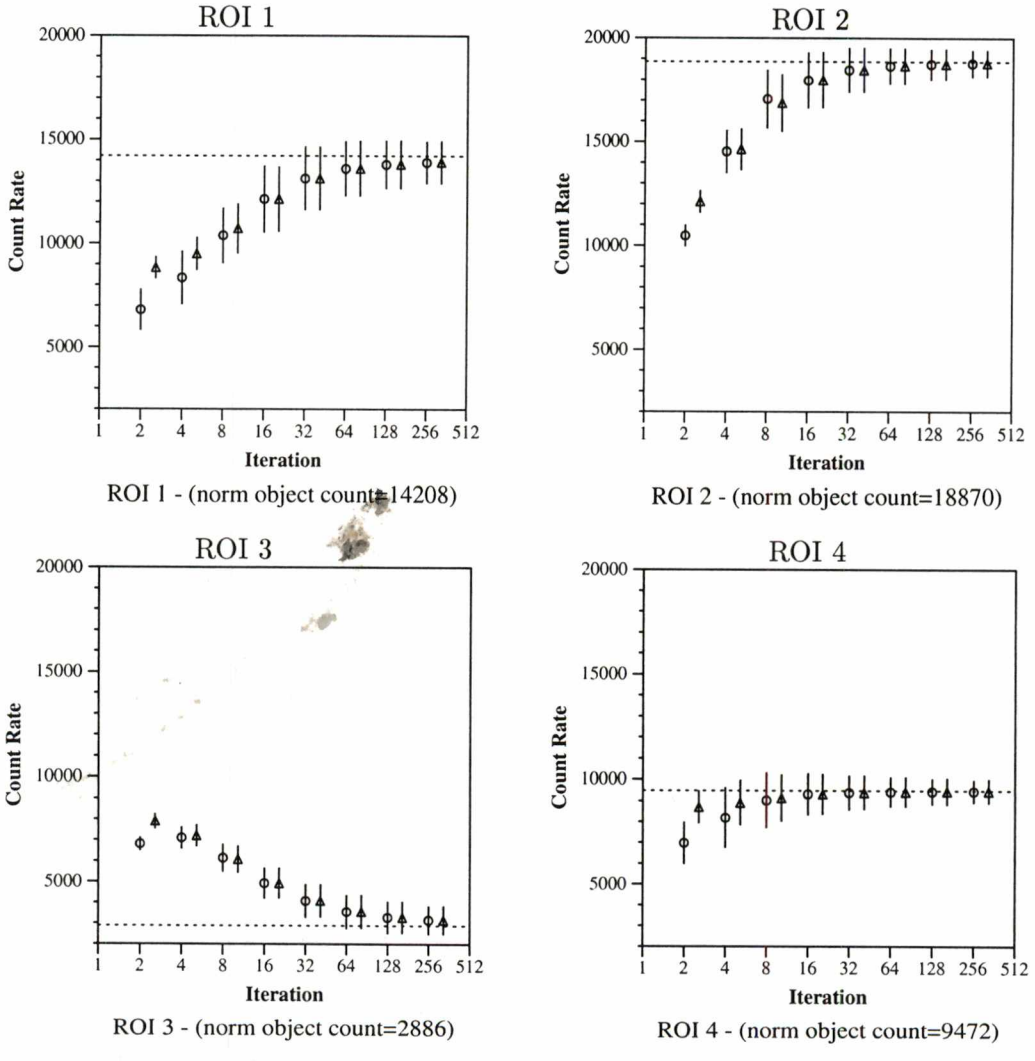


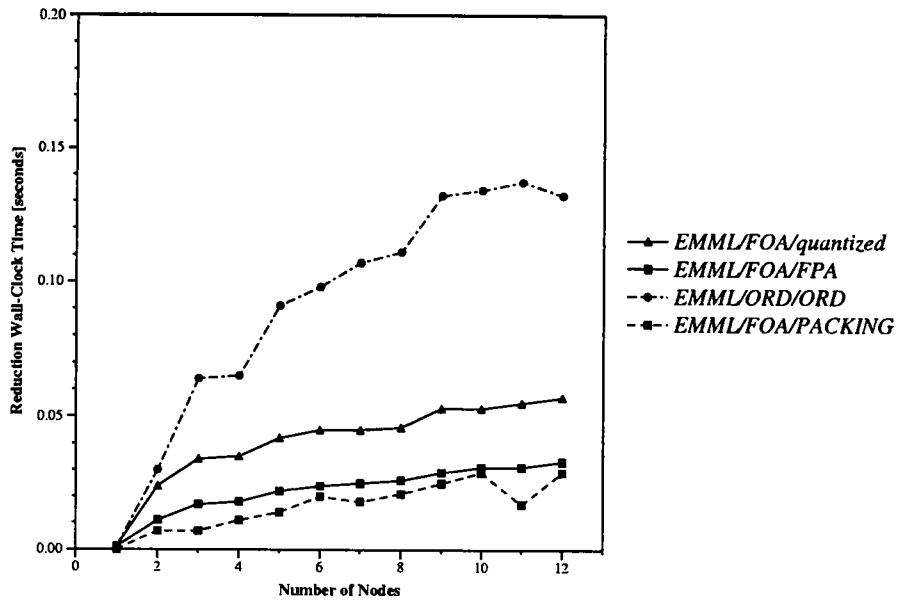
Figure 4.7: ROI Measures - Uniform Quantization. Mean and standard deviation results of measuring the activity in each of the four ROI's defined for the simulated phantom. ◯ and △ markers denote the results obtained with EM-ML/FOA and EM-ML/FOA with quantization, respectively.

4.5 Lossy Methods in Global Reduction

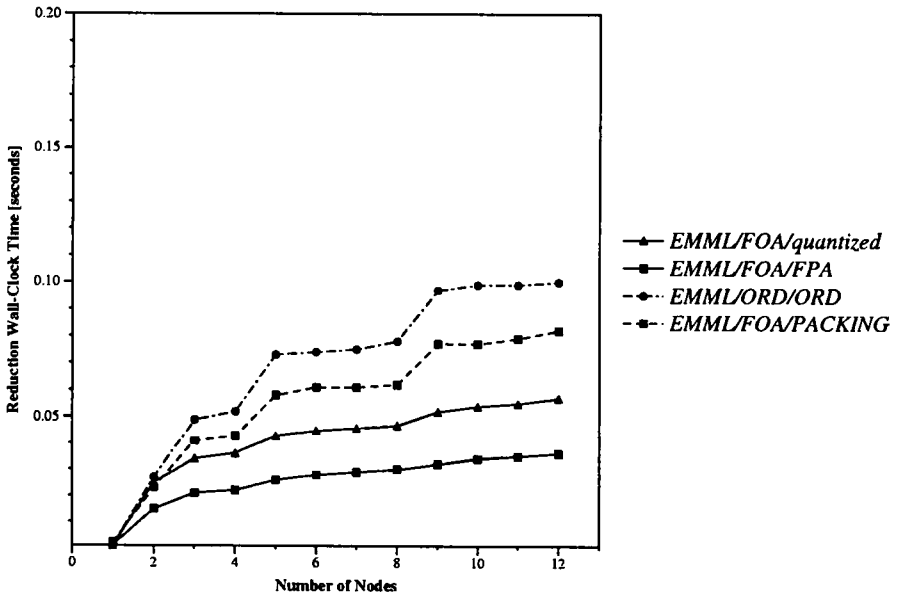
Both the fixed-point arithmetic and uniform quantization methods work in EM-ML global reduction by transforming $\mathbf{M^k}$ once locally on each node, performing the reduction and broadcast operations, and then transforming $\mathbf{M^k}$ back. Thus for a global reduction operation, the total number of transform operations is $2 \times N$ (see Sect. 2.3.2).

Figures 4.8 and 4.9 show a comparative global reduction performance of the FPA and quantization approaches to the ordinary and focus-of-attention EM-ML algorithms. At normal resolutions, on both the ATM (figure 4.8-a) and Ethernet (figure 4.9-a), there is not a performance savings over focus-of-attention with packing. At this resolution, the focus-of-attention with packing method is achieving roughly 90% compression whereas both the FPA and quantization methods are achieving a 75% compression ratio (4:1). Due to its more simplistic and straightforward nature, the packing method may also have a slight CPU performance advantage.

In figures 4.8-b and 4.9-b, both FPA and quantization outperform focus-of-attention with packing during the global reduction at three times normal resolution. The FPA method also tends to have a slight performance advantage over quantization due to the fact FPA is operating at the bit-level rather than using floating-point operations as in quantization.

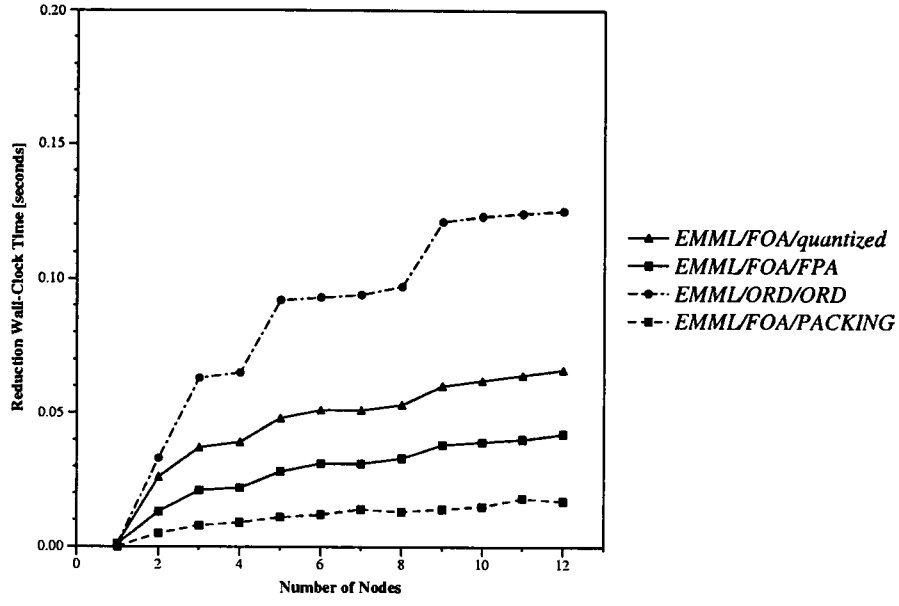


a) Compression Performance at normal resolution - ATM.

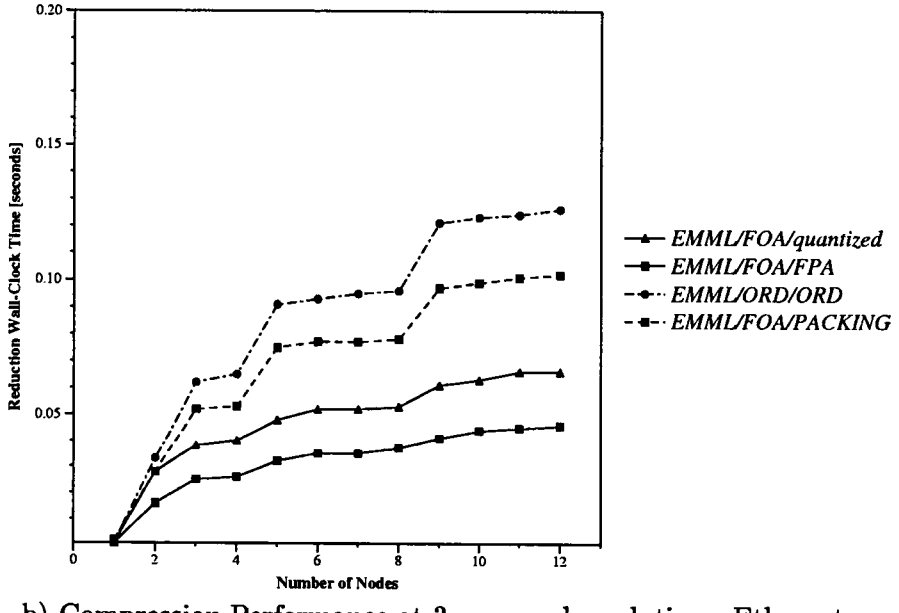


b) Compression Performance at 3 x normal resolution - ATM.

Figure 4.8: Performance of EM-ML with lossy compression on the ATM Network. The brain image was reconstructed at normal resolution in (a) and at 3 x normal resolution in (b).



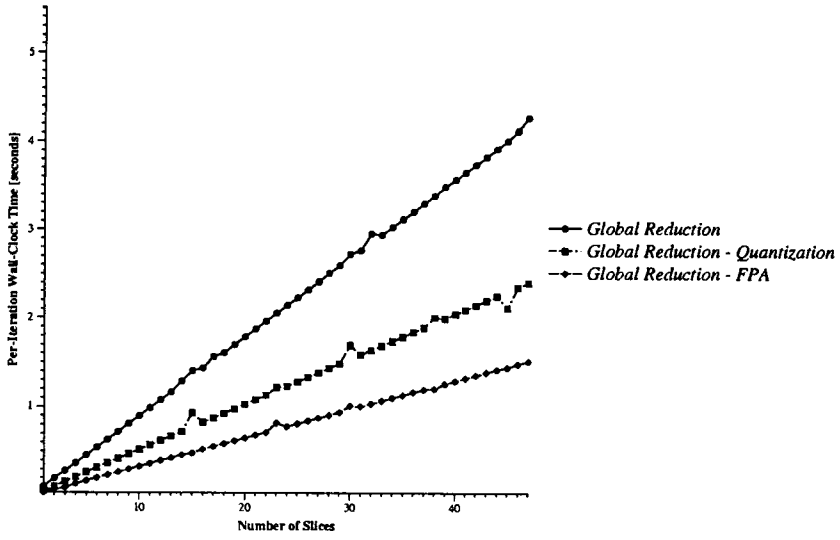
a) Compression Performance at normal resolution - Ethernet.



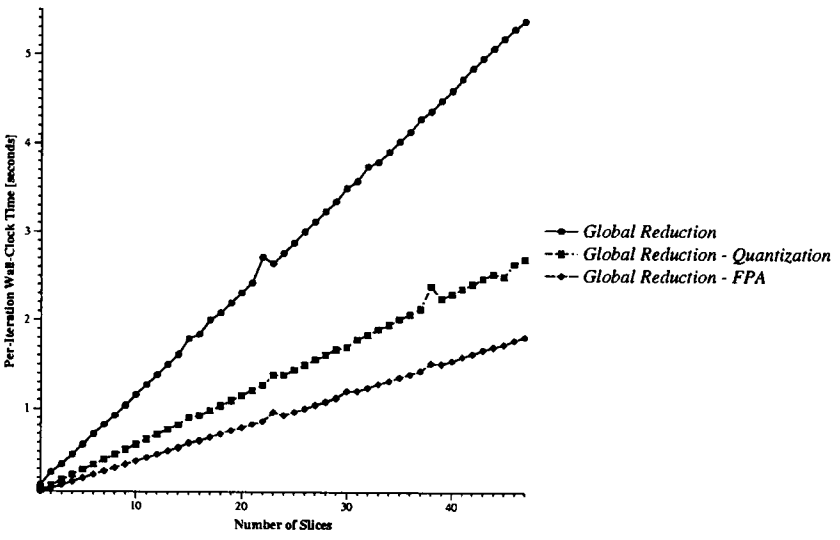
b) Compression Performance at 3 x normal resolution - Ethernet.

Figure 4.9: Performance of EM-ML with lossy compression on the Ethernet Network. The brain image was reconstructed at normal resolution in (a) and at 3 x normal resolution in (b).

Figure 4.10 provides a comparison of the synthetic volume global reduction experiment with the FPA, quantization, and ordinary EM-ML approaches. Both FPA and quantization maintain the linear performance as the amount of volume data grows from 128 KB (1 slice) to 5 MB (47 slices). The performance savings of both the FPA and quantization algorithms over ordinary EM-ML increase with significance as the volume grows. It should also be noted, however, that no performance optimizations were made to the FPA and quantization coders. Such optimizations may include incremental transmission of transformed memory blocks to take advantage of pipelining effects when dealing with this amount of data.



a) Synthetic Volume Performance - ATM.



b) Synthetic Volume Performance - Ethernet.

Figure 4.10: Performance of global reduction with large data sets and lossy methods. All runs were conducted on a 12-node configuration.

Chapter 5

CONCLUSIONS

The results of the research show that some lossy compression methods can provide modest performance improvements for EM-ML distributed computations even on the high-speed 155 Mbs ATM and 100 Mbs Ethernet networks. More significant performance improvements may be seen on slower speed networks, such as 10 Mbs non-switched Ethernet. The lossy methods developed required the use of fixed compression ratios and heuristics. Using the lossy methods, EM-ML can see performance improvement when scaling to more nodes and more data. Quantitative assessment of the lossy methods indicates no significant degradation of image quality. However, the fixed compression ratios could prove to be problematic when higher fidelity is needed to meet image quality criteria since the fixed codeword size would increase, thus having a negative impact on overall compression. This may be seen in distributed computing configurations with a large number of nodes

whereby more precision is needed than can be provided for a given codeword size. Thus, the codeword size would need to be increased. More research is needed to study the effects of global reduction with lossy methods with a large number of nodes.

Experiments also revealed that lossless compression methods were not effective in reducing the performance costs of global reduction operations. This is due in part to the fact that the lossless statistical and dictionary-based methods require a compress and decompress operation every time a pair of nodes exchange data to be summed. This creates overhead which exceeds the communication cost of just sending the data as-is on the high-speed ATM and Ethernet networks. Additionally, the constraints of entropy serve to prevent the lossless methods from compressing $\mathbf{M^k}$ as the uniformity of the multipliers increases from either zooming operations or background noise. However, more experimentation is needed on slower networks (for example 10 Mbs Ethernet) before completely dismissing the lossless methods. In addition, incremental transmission approaches may be studied to determine how pipelining affects global reduction performance. But, the problem of entropy-coding using simple models, such as order-0, will continue to prevent the realization of reasonable compression ratios to even warrant the use of lossless methods. Therefore, future research should focus first on the problem of entropy-coding EM-ML data before examining performance speed-up with

lossless methods.

Network congestion on the high-speed ATM and Ethernet networks does not impact overall performance for small 2D image reconstruction. In this case the overhead of the the global reduction operation itself induces the greatest performance penalty. However, moving to 3D image reconstruction, a performance penalty can be expected from both high network utilization, potential network saturation, and global reduction overhead. The realization of network saturation is dependent upon the bandwidth and access method of the underlying physical network along with the number of nodes participating in the global reduction. More research is needed to determine how process group size impacts global reduction performance on different network types.

Bibliography

Bibliography

[1] R. B. Ash. *Information Theory.* John Wiley & Sons, Inc., first edition, 1965.

[2] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression.* Prentice-Hall, Englewood Cliffs, N.J., first edition, 1990.

[3] T.C. Bell. Better OPM/L text compression. *IEEE Trans. Communications,* COM-34(12):1176–1182, December 1986.

[4] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Soc., Ser. B,* 39(1):1–38, 1977.

[5] R. B.K. Dewar and M. Smosna. *Microprocessors - A Programmer's View.* McGraw-Hill, first edition, 1990.

[6] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard.* Technical Report CS-94-230, Computer Science Dept., University of Tennessee, Knoxville, TN, 1994.

[7] R.G. Gallagher. Variations on a theme by Huffman. *IEEE Trans. Information Theory*, IT-24(6):668–674, November 1978.

[8] G. A. Gibson and Y-C Liu. *Microcomputers for Engineers and Scientists.* Prentice-Hall, Englewood Cliffs, N.J., second edition, 1987.

[9] J. Gregor and D. A. Huff. A computational study of the focus-of-attention EM-ML algorithm for PET reconstrution. *To appear in special issue of Parallel Computing*, 1998.

[10] J. Gregor and D. A. Huff. A focus-of-attention preprocessing scheme for EM-ML PET reconstruction. *IEEE Trans. Medical Imaging*, 15(2), April 1998.

[11] R. W. Hamming. *Coding and Information Theory.* Prentice-Hall, Englewood Cliffs, N.J., first edition, 1980.

[12] G.T. Herman, A. Lent, and S.W. Rowland. ART: Mathematics and application. *J Theor. Biology*, 42:1–32, 1973.

[13] D.A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, September 1952.

[14] N.S. Jayant and P. Noll. *Digital Coding of Waveforms - Principles and Applications to Speech and Video.* Prentice-Hall, Englewood Cliffs, N.J., first edition, 1984.

[15] D. W. Jones. Application of splay trees to data compression. *Communications of the ACM*, 31(8):996–1007, August 1988.

[16] L. Kaufman. Maximum likelihood, least squares, and penalized least squares for PET. *IEEE Trans. Medical Imaging*, 12:200–214, 1993.

[17] K. Lange and R. Carson. EM reconstruction algorithms for emission and transmission tomography. *J. Computer Assisted Tomography*, 8:306–316, 1984.

[18] A. Lempel and J. Ziv. A universal algorithm for sequential data compression. *IEEE Trans. Information Theory*, 23(3):337–343, May 1977.

[19] A. Lempel and J. Ziv. Compression of individual sequences via variable-rate coding. *IEEE Trans. Information Theory*, 24(5):530–536, September 1978.

[20] M. Nelson and J-L Gailly. *The Data Compression Book.* M&T Books, New York, New York, second edition, 1996.

[21] S.P. Olesen, J. Gregor, M.G. Thomason, and G. Smith. EM-ML PET reconstruction on multiple processors with reduced communications. *Intl. J. Imaging Syst. & Techn.*, (7):215–223, 1996.

[22] T-S Pan and A. E. Yagle. Numerical study of multigrid implementations of some iterative image reconstruction algorithms. *IEEE Trans. Medical Imaging*, 10:572–588, 1991.

[23] K. Sayood. *Introduction to Data Compression.* Morgan Kaufman Publishers, Inc., San Francisco, CA, first edition, 1996.

[24] L. A. Shepp and Y. Vardi. Maximum likelihood reconstruction for emission tomography. *IEEE Trans. Medical Imaging*, 1:113–122, 1982.

[25] D.D. Sleator and R.E. Tarjan. Self-adjusting binary search trees. In *Proceedings of the ACM SIGACT Symposium on Theory of Computing*, pages 235–245, ACM, New York, New York, 1983.

[26] R.E. Tarjan and D.D. Sleator. Self-adjusting binary search tree. *J. ACM*, 32(3):652–686, July 1985.

[27] T. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, June 1984.

[28] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, June 1987.

VITA

John McNeely was born in Upper Sandusky, Ohio on October 2, 1968. He attended public school in Bristol, Tennessee and Bluff City, Tennessee, the Sullivan County Public School System, where he graduated from Sullivan East High School in June, 1986. He entered the University of Tennessee - Chattanooga on a track and cross-country scholarship during August of 1986, where in May, 1990 he received the Bachelor of Science degree in Computer Science, cum laude. Upon graduation, Mr. McNeely began work with Southwest Research Institute in San Antonio, Texas as a Research Analyst. In March of 1992, he returned to Tennessee to work for Martin Marietta Energy Systems as a Computing Analyst. In the spring of 1993, he began graduate studies in computer science part-time at the University of Tennessee - Knoxville while still working full-time. The Master of Science degree was received May, 1998.

Mr. McNeely is currently Vice President of Sword and Shield Computer Services, Inc. and maintains residence in the Knoxville area.