INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Parallelization of the simulated annealing algorithm

Seacat, Russell Holland, III, Ph.D.
The University of Arizona, 1993
PARALLELIZATION OF THE SIMULATED ANNEALING ALGORITHM

by

Russell Holland Seacat III

A Dissertation Submitted to the Faculty of the
COMMITTEE ON OPTICAL SCIENCES (GRADUATE)

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

1993
As members of the Final Examination Committee, we certify that we have read the document prepared by Russell H. Seacat III entitled Parallelization of the Simulated Annealing Algorithm and recommend that it be accepted as fulfilling the requirements for the Degree of Doctor of Philosophy.

H.H. Barrett  12/21/93
Richard L. Shoemaker  12/21/93
Murray Sargent III  12/21/93

Final approval and acceptance of this document is contingent upon the candidate's submission of the final copy of the document to the Graduate College.

I hereby certify that I have read this document prepared under my direction and recommend that it be accepted as fulfilling the requirement.

H.H. Barrett  12/21/93
STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: [Signature]
ACKNOWLEDGEMENTS

I would like to acknowledge, first and foremost, my advisor Dr. Harrison Barrett (Harry). Harry's vast and varied technical expertise provided the foundation upon which this dissertation is built and his unwavering dedication and faith in my abilities proved to be the glue that has pulled it together.

I would also like to acknowledge the contribution to this dissertation of Dr. Richard Shoemaker (Rick). The designer and architect of TRIMM, Rick not only provided technical expertise, but also encouragement along the way.

Many graduate students come and go and I was fortunate enough to cross paths with several that contributed to the ideas that went into this research. Kathy Magee, while sharing the same laboratory, patiently put up with my fits of frustration. Tim Roney and Ted Gooley not only provided technical expertise and intellectual stimulation, but also friendships that often went above and beyond normal expectations. John Aarsvold, the answer man, was always willing to go above and beyond the call of duty, and often did, in helping me answer questions.

I would also like to thank Debbie Spargur for her patience, understanding, and help in coping with the administrative and bureaucratic mazes I often found myself lost in.

This work was funded by NCI grants PO1 CA23417 and ROI CA52643.
DEDICATION

For my parents, Russell and Wanda, without whose endless love and unwavering faith I would not have made it.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LIST OF ILLUSTRATIONS</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>LIST OF TABLES</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>ABSTRACT</td>
<td>14</td>
</tr>
<tr>
<td>1.</td>
<td>INTRODUCTION</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Scope of this Dissertation</td>
<td>18</td>
</tr>
<tr>
<td>2.</td>
<td>NUCLEAR MEDICINE IMAGING</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Coded-Aperture Imaging</td>
<td>24</td>
</tr>
<tr>
<td>3.</td>
<td>PARALLEL PROCESSING</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Evolution of the von Neumann Architecture</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Parallel Architectures</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>SIMD</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Pipeline Operation</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>CRAY-1</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>The Connection Machine</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>MIMD</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>Interconnection Networks</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>Fixed-Networks</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>Switched-Networks</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>Parallel Programming</td>
<td>46</td>
</tr>
<tr>
<td>4.</td>
<td>TRIMM</td>
<td>65</td>
</tr>
<tr>
<td>5.</td>
<td>IMAGE RECONSTRUCTION AND SIMULATED ANNEALING</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>Simulated Annealing</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>Previous Work on Parallel Simulated Annealing</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>Implementing the Algorithm</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>Estimate Constraints</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>Data-space and Object-space Implementations</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>Domain Decomposition and Space Dependence</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>Calculating $\Delta E$ in Data Space</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>Calculating $\Delta E$ in Object Space</td>
<td>88</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS -- Continued

## 6. PARALLEL IMPLEMENTATIONS ................................................................. 93

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Communications</td>
<td>95</td>
</tr>
<tr>
<td>Asynchronous Communications Scheme</td>
<td>95</td>
</tr>
<tr>
<td>Ring Implementation</td>
<td>95</td>
</tr>
<tr>
<td>TRIMM</td>
<td>96</td>
</tr>
<tr>
<td>Broadcast Mode Disabled</td>
<td>97</td>
</tr>
<tr>
<td>Broadcast Mode Enabled</td>
<td>97</td>
</tr>
<tr>
<td>Synchronous communications Scheme</td>
<td>98</td>
</tr>
<tr>
<td>Ring Implementation</td>
<td>98</td>
</tr>
<tr>
<td>TRIMM: Broadcast Mode Enabled</td>
<td>99</td>
</tr>
<tr>
<td>Hybrid Communications Scheme</td>
<td>99</td>
</tr>
</tbody>
</table>

| \( \Delta E(\hat{f}) \) v. \( \Delta E(\hat{g}) \)                    | 100  |
| Binary Myocardium                                                      | 101  |
| Speed                                                                  | 104  |
| Hamming Distance                                                       | 104  |

| Implementation: Reconstruction Space Decomposition and Global Communications | 106  |
| Imaging System                                                          | 106  |
| Theoretical Speedup                                                     | 106  |
| Domain Decomposition                                                    | 109  |
| Thyroid Phantom: Experimental Results                                   | 110  |
| Brain Phantom: Experimental Results                                     | 112  |
| Summary                                                                 | 114  |

## 7. SUMMARY AND SUGGESTIONS FOR FUTURE WORK ..................................... 166

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suggestions for Future Work</td>
<td>168</td>
</tr>
</tbody>
</table>

## 8. REFERENCES .................................................................................... 169
## LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>25</td>
</tr>
<tr>
<td>2.2(a)</td>
<td>26</td>
</tr>
<tr>
<td>2.2(b)</td>
<td>26</td>
</tr>
<tr>
<td>2.3</td>
<td>27</td>
</tr>
<tr>
<td>3.1(a)</td>
<td>52</td>
</tr>
<tr>
<td>3.1(b)</td>
<td>52</td>
</tr>
<tr>
<td>3.1(c)</td>
<td>52</td>
</tr>
<tr>
<td>3.2</td>
<td>53</td>
</tr>
<tr>
<td>3.3</td>
<td>54</td>
</tr>
<tr>
<td>3.4(a)</td>
<td>55</td>
</tr>
<tr>
<td>3.4(b)</td>
<td>55</td>
</tr>
<tr>
<td>3.4(c)</td>
<td>55</td>
</tr>
<tr>
<td>3.5</td>
<td>56</td>
</tr>
<tr>
<td>3.6</td>
<td>57</td>
</tr>
<tr>
<td>3.7</td>
<td>58</td>
</tr>
<tr>
<td>3.8</td>
<td>59</td>
</tr>
<tr>
<td>3.9(a)</td>
<td>60</td>
</tr>
<tr>
<td>3.9(b)</td>
<td>60</td>
</tr>
<tr>
<td>3.10</td>
<td>61</td>
</tr>
<tr>
<td>3.11</td>
<td>62</td>
</tr>
<tr>
<td>3.12</td>
<td>63</td>
</tr>
<tr>
<td>3.13</td>
<td>64</td>
</tr>
<tr>
<td>4.1</td>
<td>68</td>
</tr>
</tbody>
</table>
# LIST OF ILLUSTRATIONS -- *Continued*

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>Transputer architecture</td>
<td>69</td>
</tr>
<tr>
<td>4.3</td>
<td>TRIMM module architecture</td>
<td>70</td>
</tr>
<tr>
<td>4.4</td>
<td>Transputer serial-link connections</td>
<td>71</td>
</tr>
<tr>
<td>5.1</td>
<td>Estimate and transformation matrix for the data-space formulation of $\Delta E$</td>
<td>91</td>
</tr>
<tr>
<td>5.2</td>
<td>Estimate and transformation matrix for the object-space formulation of $\Delta E$</td>
<td>92</td>
</tr>
<tr>
<td>6.1</td>
<td>TRIMM ring configuration</td>
<td>115</td>
</tr>
<tr>
<td>6.2</td>
<td>Synchronous communication scheme flowchart</td>
<td>116</td>
</tr>
<tr>
<td>6.3</td>
<td>Asynchronous communication scheme flowchart</td>
<td>117</td>
</tr>
<tr>
<td>6.4</td>
<td>Binary myocardium</td>
<td>118</td>
</tr>
<tr>
<td>6.5</td>
<td>Binary myocardium imaging configuration</td>
<td>119</td>
</tr>
<tr>
<td>6.6(a)</td>
<td>C code for the inner loop of object-space $\Delta E$ formulation</td>
<td>120</td>
</tr>
<tr>
<td>6.6(b)</td>
<td>C code for the inner loop of data-space $\Delta E$ formulation</td>
<td>121</td>
</tr>
<tr>
<td>6.7</td>
<td>Reconstruction space decomposition</td>
<td>122</td>
</tr>
<tr>
<td>6.8</td>
<td>Reconstruction of the binary myocardium utilizing the data-space $\Delta E$ formulation</td>
<td>123</td>
</tr>
<tr>
<td>6.9</td>
<td>Reconstruction of the binary myocardium utilizing the object-space $\Delta E$ formulation</td>
<td>124</td>
</tr>
<tr>
<td>6.10</td>
<td>Digitized thyroid phantom</td>
<td>125</td>
</tr>
<tr>
<td>6.11</td>
<td>Digitized Hoffman brain phantom</td>
<td>126</td>
</tr>
<tr>
<td>6.12</td>
<td>Imaging configuration for the thyroid and brain phantoms</td>
<td>127</td>
</tr>
<tr>
<td>6.13(a)</td>
<td>Speedup for the ring architecture implementing the asynchronous communications scheme and the data-space $\Delta E$ formulation</td>
<td>128</td>
</tr>
</tbody>
</table>
### LIST OF ILLUSTRATIONS -- Continued

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.13(b)</td>
<td>Speedup for the ring architecture implementing the synchronous communications scheme and the data-space ΔE formulation</td>
<td>129</td>
</tr>
<tr>
<td>6.14(a)</td>
<td>Speedup for TRIMM: broadcast mode, asynchronous communications, and the data-space ΔE formulation</td>
<td>130</td>
</tr>
<tr>
<td>6.14(b)</td>
<td>Speedup for TRIMM broadcast mode, asynchronous communications and the object-space ΔE formulation</td>
<td>131</td>
</tr>
<tr>
<td>6.15(a)</td>
<td>Reconstruction speedup of the Hoffman brain phantom: ring architecture and both asynchronous and synchronous communications</td>
<td>132</td>
</tr>
<tr>
<td>6.15(b)</td>
<td>Reconstruction speedup of the Hoffman brain phantom: TRIMM with broadcast and both asynchronous and synchronous communications</td>
<td>133</td>
</tr>
<tr>
<td>6.16(a)</td>
<td>Theoretical speedup utilizing the proposed brain imager: ring architecture and both asynchronous and synchronous communications</td>
<td>134</td>
</tr>
<tr>
<td>6.16(b)</td>
<td>Theoretical speedup utilizing the proposed brain imager: TRIMM with broadcast and both asynchronous and synchronous communications</td>
<td>135</td>
</tr>
<tr>
<td>6.17(a)</td>
<td>Initial estimate and serial reconstruction of the thyroid phantom</td>
<td>136</td>
</tr>
<tr>
<td>6.17(b)</td>
<td>Thyroid phantom parallel reconstruction: ring architecture, 12 processors, and asynchronous communications</td>
<td>137</td>
</tr>
<tr>
<td>6.17(c)</td>
<td>Thyroid phantom parallel reconstruction: ring architecture, 24 processors, and hybrid communications</td>
<td>138</td>
</tr>
<tr>
<td>6.17(d)</td>
<td>Thyroid phantom parallel reconstruction: ring architecture, 36 processors, and synchronous communications</td>
<td>139</td>
</tr>
<tr>
<td>6.17(e)</td>
<td>Thyroid phantom parallel reconstruction: ring architecture, 48 processors, and synchronous communications</td>
<td>140</td>
</tr>
<tr>
<td>6.17(f)</td>
<td>Thyroid phantom parallel reconstruction: ring architecture, 60 processors, and synchronous communications</td>
<td>141</td>
</tr>
<tr>
<td>6.18(a)</td>
<td>Energy vs. time for serial reconstruction of the thyroid phantom</td>
<td>142</td>
</tr>
</tbody>
</table>
LIST OF ILLUSTRATIONS -- Continued

Figure                  Page
6.18(b)  Energy vs. time: thyroid phantom parallel reconstruction, ring architecture, 12 processors, and asynchronous communications .......................................................... 143
6.18(c)  Energy vs. time: thyroid phantom parallel reconstruction, ring architecture, 24 processors, and hybrid communications ............................................................................. 144
6.18(d)  Energy vs. time: thyroid phantom parallel reconstruction, ring architecture, 36 processors, and synchronous communications .......................................................... 145
6.18(e)  Energy vs. time: thyroid phantom parallel reconstruction, ring architecture, 48 processors, and synchronous communications .......................................................... 146
6.18(f)  Energy vs. time: thyroid phantom parallel reconstruction, ring architecture, 60 processors, and synchronous communications .......................................................... 147
6.19     Speedup for the parallel reconstructions of the thyroid phantom .................................................................................................................. 148
6.20(a)  Initial estimate and serial reconstruction of the Hoffman brain phantom .................................................................................................................. 149
6.20(b)  Hoffman brain phantom parallel reconstruction: ring architecture, 12 processors, and asynchronous communications .......................................................... 150
6.20(c)  Hoffman brain phantom parallel reconstruction: ring architecture, 24 processors, and asynchronous communications .......................................................... 151
6.20(d)  Hoffman brain phantom parallel reconstruction: ring architecture, 36 processors, and hybrid communications .......................................................... 152
6.20(e)  Hoffman brain phantom parallel reconstruction: ring architecture, 48 processors, and synchronous communications .......................................................... 153
6.20(f)  Hoffman brain phantom parallel reconstruction: ring architecture, 60 processors, and synchronous communications .......................................................... 154
## LIST OF ILLUSTRATIONS -- Continued

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.21(a)</td>
<td>Energy vs. time for serial reconstruction of the Hoffman brain phantom</td>
<td>155</td>
</tr>
<tr>
<td>6.21(b)</td>
<td>Energy vs. time: Hoffman brain phantom parallel reconstruction, ring architecture, 12 processors, and asynchronous communications</td>
<td>156</td>
</tr>
<tr>
<td>6.21(c)</td>
<td>Energy vs. time: Hoffman brain phantom parallel reconstruction, ring architecture, 24 processors, and asynchronous communications</td>
<td>157</td>
</tr>
<tr>
<td>6.21(d)</td>
<td>Energy vs. time: Hoffman brain phantom parallel reconstruction, ring architecture, 36 processors, and hybrid communications</td>
<td>158</td>
</tr>
<tr>
<td>6.21(e)</td>
<td>Energy vs. time: Hoffman brain phantom parallel reconstruction, ring architecture, 48 processors, and asynchronous communications</td>
<td>159</td>
</tr>
<tr>
<td>6.21(b)</td>
<td>Energy vs. time: Hoffman brain phantom parallel reconstruction, ring architecture, 60 processors, and asynchronous communications</td>
<td>160</td>
</tr>
<tr>
<td>6.22</td>
<td>Speedup for the parallel reconstructions of the Hoffman brain phantom</td>
<td>161</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>6.1</td>
<td>Configurations for Study</td>
<td>162</td>
</tr>
<tr>
<td>6.2</td>
<td>$\Delta E$ Formulations: Implementation Factors</td>
<td>163</td>
</tr>
<tr>
<td>6.3</td>
<td>$\Delta E$ Formulations: Experimental Results</td>
<td>163</td>
</tr>
<tr>
<td>6.4</td>
<td>Thyroid Phantom: Experimental Results</td>
<td>164</td>
</tr>
<tr>
<td>6.4</td>
<td>Hoffman Brain Phantom: Experimental Results</td>
<td>165</td>
</tr>
</tbody>
</table>
ABSTRACT

Nuclear medicine imaging involves the introduction of a radiopharmaceutical into the body and the subsequent detection of the radiation emanating from the organ at which the procedure was directed. The data set resulting from such a procedure is generally very underdetermined, due to the dimensions of the imaging apparatus, and underconstrained due to the noise in the imaging process. A means by which more information can be obtained is through a form of imaging utilizing code apertures. Although increasing the amount of information collected, coded-aperture imaging results in a multiplexing of the data. Demultiplexing the data requires a reconstruction process not required in conventional nuclear medicine imaging.

The reconstruction process requires the optimization of an estimate to the object to be reconstructed. This optimization is done through the minimization of an energy functional. The minimization of such energy functionals requires the optimization of several parameters. Solution of this type problem is difficult because there are far too many degrees of freedom to permit an exhaustive search for an optimum, and in many cases no algorithms are known which will determine the exact optimum with significantly less work than exhaustive search. Instead, heuristic algorithms, such as the simulated annealing algorithm, have been employed and have proven effective in minimizing such energy functionals. Unfortunately, the simulated annealing algorithm, as characteristic of Monte Carlo algorithms, is very computer intensive; in fact, it is so intensive that insufficient computational power is often the chief hindrance to investigation of the algorithm. The simulated annealing algorithm, however, is amenable to parallel processing.

The goal of the research in this dissertation is to investigate the parameters involved in implementing the simulated annealing algorithm in parallel; however, the form of the simulated annealing algorithm implemented here requires no annealing because the energy functionals investigated are quadratic in form. The parameters related to the parallelization of the simulated annealing algorithm include the decomposition of the reconstruction space among the processors, the formulation of the problem at the estimate level with the smallest task being a single perturbation trial evaluated on a local basis, and the communications required to keep all the processors as current as possible with changes made simultaneously to the estimate. Three objects, varying in size, shape and detail, are reconstructed utilizing the TRIMM parallel processor.
CHAPTER 1
INTRODUCTION

In the age of information, processing more information in less time is a continuing quest. This desire can be satisfied in two different ways: processing a certain amount of information faster or processing more information in the same amount of time. Though the difference between the two concepts seems subtle, the efforts to achieve each have led in quite different directions. The former, on a fundamental level, has focused on developing faster processors while the latter involves incorporating many processors cooperating simultaneously, or in parallel, to solve a single problem.

Since its conception, the computer has undergone relatively few changes in its basic organizational structure as described by von Neumann in the 1940s. In this von Neumann architecture, a single stream of instructions is processed sequentially, input arguments are transferred from a single memory to an arithmetic unit, and results are returned to the same memory. One instruction is fetched from memory and interpreted for every arithmetic operation.

Through the advances made in solid-state technology, the performance of the components comprising the von Neumann architecture has improved greatly. This performance has resulted in more information being processed at a greater speed, which in turn has resulted in what is known as the von Neumann bottleneck. This "bottleneck" is caused by the steady stream of data exchanged by the processor and the memory each time an instruction is executed.

A fundamental constraint on computer architecture design, namely the cost of processing logic, has been removed through advances made in VLSI (Very Large Scale Integration), the result of which is the inevitable obsolescence of the von Neumann architecture and the realization of computer architectures that previously existed only in
theory. Problems that were once considered computationally intractable may now be attacked by implementing systems in which many processors work in parallel to solve the problem.

Parallel architecture designs vary from those incorporating a few very powerful processors to those incorporating thousands of relatively simple processors. The memory configurations incorporated in the myriad of architectures is as varied as the types, and number, of processors involved. Architectures vary from those providing substantial local memory (the memory associated with each processor) to those that provide one central memory shared by all the processors. Since most problems cannot be implemented in a fully parallel fashion, a means for communication among processing and/or memory elements must be provided. Communication among elements in a parallel architecture is achieved through an interconnection network providing the desired communication pathways. The existing designs of interconnection networks comprise as great a collection of choices as do the processor types and memory configurations. Interconnection networks range from a simple ring in which each processor is connected to its two nearest neighbors to a fully connected network in which every element, whether it be a processor or memory, is connected to every other element.

The wide variety of parallel architectures is a reflection of the varying degrees of parallelism exhibited in different classes of problems. Many parallel architectures that are in existence today have been designed for the implementation of a specific class of problems and the degree of parallelism associated with that class.

The primary advantage of a parallel processor is its greater "number-crunching" capabilities. For if one processor is capable of performing one arithmetic operation per unit time, then a parallel processor comprising N processors provides the potential to achieve N times as many operations during the same unit of time. This added power can be advantageous in many respects, one of which is the area of algorithmic research.
If an algorithm contains many variables and options and is extremely time-consuming to run to completion, then the investigation of these variables and options, and their effect on the output, is itself an even more time-consuming process. The computer-intensive nature of such an investigation is sometimes so overwhelming that it is the chief impediment to progress. One such algorithm that falls into this category is the simulated annealing algorithm.

The simulated annealing algorithm is a generalized optimization algorithm that has gained wide acceptance as an effective approach to solving large combinatorial optimization problems or finding global extrema of complicated functionals. In the context of image reconstruction, the simulated annealing algorithm is used to invert the imaging equation given by

\[ g = Hf + n \]

where \( g \) is the data set produced from the object \( f \) when imaged through a system described by \( H \), plus some observational error, \( n \), associated with the process.

Conventional nuclear medicine imaging procedures produce images that are inherently noisy due to the low number of photons available for imaging and the time required to complete the imaging process. In an effort to improve this situation by obtaining more photons, in a shorter period of time, an unconventional imaging process termed coded-aperture imaging is employed. Although providing substantial improvements in the number of photons acquired, and the acquisition time, a coded-aperture system presents a problem that is ill-posed and often seriously underconstrained. Such problems do not have unique solutions, so that any a priori knowledge about the object must be used to develop a unique reconstruction. The simulated annealing algorithm is an algorithm that is flexible enough to handle such a problem while simultaneously incorporating a priori information into the reconstruction process.
The inversion is performed through a Monte Carlo procedure. Monte Carlo procedures are in general quite time consuming due to the large number of calculations involved and the frequent need for random numbers; however, they are quite amenable to parallel processing.

As mentioned previously, varying degrees of parallelism are achievable in both architectures and implementations of algorithms. It is important to match the degree of parallelism in the algorithm to that of the architecture to avoid unnecessary overhead in either or both. Unfortunately, it is not always feasible to provide an adequate match from existing architectures. Even if a compatible architecture does exist, it is likely that it is not an affordable solution. The alternative, then, is to build a custom machine tailored to the needs of the problem at hand.

The TRIMM (Transputer Implemented Monte Carlo Methods) processor is a parallel processor designed specifically for the implementation of Monte Carlo algorithms. Based on a microprocessor chip called the transputer, TRIMM is a relatively inexpensive alternative to existing supercomputers with comparable performance for some applications.

Scope of This Dissertation

The simulated annealing algorithm involves many parameters and options. In an effort to explore thoroughly the full range of these parameters and optimize the algorithm, the need exists for a more rapid turn-around of the studies than is currently available. This dissertation addresses the effort to investigate various parameters involved in the parallelization of the simulated annealing algorithm on TRIMM. Because the energy functionals used for reconstruction simplify to a quadratic form, no local minimas exist, and therefore no annealing is required. The simulated annealing algorithm simplifies to a
Monte Carlo Search algorithm, which never the less serves as a meaningful test bed for studying some aspects of parallel simulated annealing.

The dissertation is organized as follows. Chapter 2 offers an introduction to the principles involved in conventional nuclear medicine imaging and coded-aperture imaging. Chapter 3 presents a brief history of computing, both hardware and software, and presents a survey of parallel architecture designs and software models. Chapter 4 discusses the architecture and attributes of TRIMM parallel processor. Chapter 5 discusses the basic principles of the simulated annealing algorithm, aspects of its parallel implementations on TRIMM, and its application to coded-aperture imaging. Chapter 6 contains the experimental results of various parallel implementations of the simulated annealing algorithm on ring architectures of different sizes. Also included are performance projections for analogous implementations on TRIMM utilizing the broadcast mode. Questions of domain decomposition of the reconstruction space among the processors, local and global update schedules for the estimate, and the application of a priori knowledge to the estimate are addressed. In chapter 7, the dissertation will be summarized and directions for further work will be suggested.
CHAPTER 2
NUCLEAR MEDICINE IMAGING

Nuclear medicine is a branch of diagnostic radiology that is concerned with imaging the functioning of an organ. This is in contrast, for example, to x-ray diagnostics where the determination of the structure itself is the primary concern. In nuclear medicine, a pharmaceutical is chosen to interact with the organ of interest (the target organ) and is combined, or tagged, with a radioactive isotope. The combination of pharmaceutical and radioactive isotope (termed a radiopharmaceutical) is then introduced into the body by injection or ingestion. The performance of the target organ can be measured, for example, by measuring its rate of uptake of the pharmaceutical. The rate of uptake can be determined by observing the spatial distribution of the pharmaceutical activity within the organ over a period of time.

The distribution of the pharmaceutical within the organ is determined by detecting the gamma-ray photons emitted isotropically by the radioactive decay of the isotope nuclei. Due to the high energy of these photons (on the order of 100,000 - 500,000 electron-volts), conventional methods of image formation are not applicable. Since there are no materials that can reflect or refract photons of this energy, alternatives must be employed. One such alternative is to selectively channel the photons from the source to the detector through a parallel-hole collimator.

Since a parallel-hole collimator is typically made out of lead, a material with a high absorption coefficient for photons of this energy, the function of the collimator can be thought of as a blocking action rather than a transmitting one. Placed in front of a detector sensitive to the photons being emitted, each collimator restricts the photons that can reach the detector to a small cone originating at the source of decay and terminating at the detector surface. For a single pinhole, for example, the fraction of photons emitted
from a volume element (voxel) in the organ and passing through the collimator opening is simply the solid angle subtended by the collimator opening at the voxel divided by the solid angle of the entire sphere. The fraction of emitted photons that is transmitted by the collimator is generally very small, on the order of $3 \times 10^{-4}$ (Barrett HH and Swindell W, 1981, p. 174). This small fraction of transmitted photons per object element, along with the desire to minimize the radiation dose given to the patient, results in relatively few photons being available to form the nuclear medicine image. This lack of photons manifests itself in what is the bane of nuclear medicine imaging, Poisson noise.

The type of detector used to detect these high-energy photons, termed a gamma-ray camera, generally consists of a scintillation crystal, sensitive to the photons of the energy being imaged, coupled to a two-dimensional array of photomultiplier tubes (PMTs). A camera of this configuration, called an Anger camera after its inventor (Anger HO, 1958), is depicted in figure 2.1. When a gamma-ray photon strikes the face of the crystal, a secondary shower of lower-energy photons is generated. This secondary shower of photons, generally in the visible range of the spectrum, is then detected by the PMTs. The spatial distribution of these secondary photons depends on where the gamma-ray photon strikes the face of the crystal. The PMTs detect the secondary photons, and the position of the original photon is estimated from the relative strengths of the PMT signals.

An Anger camera and collimator used in a stationary position provides a single two-dimensional view of the three-dimensional structures of the body. Used in this manner, the procedure to obtain an image is simple and straightforward, the image simply being the projection of the emitting structures through the parallel hole collimator onto the face of the detector. The resulting image, however, may consist of overlapping structures along a particular line-of-sight and without any delineating depth information about the respective structures it is often difficult to determine the distribution of the
pharmaceutical activity. In addition, an effect of the collimator is to blur the image of structures that lie farther from the plane of the collimator (the resolution of the collimator falls off with distance away from the plane of the collimator (Barrett HH and Swindell W, 1981, pp.127,171). Further complicating the situation, though not attributable to the detection system, are the effects of scattering and attenuation of the gamma-ray photons within the body. Therefore, the farther away from the collimator an emitting structure lies, the less likely it is that it contributes photons to the final image.

Single-photon emission computed tomography (SPECT), modeled after x-ray tomography, incorporates an Anger camera that rotates completely around the body to produce fully three-dimensional images of the pharmaceutical activity distribution within the body (fig. 2.2). The camera typically rotates 180 to 360 degrees around the body, stopping intermittently at the desired positions to record a projection. This process usually requires 30 to 45 minutes to complete and records on the order of fifty two-dimensional views of the body. Unlike the stationary system, however, a SPECT system requires a reconstruction step to produce the final (three-dimensional) image.

The rotating-camera system, though negating many of the detrimental features of the stationary system, is not without some of its own. Patient motion and half-life decay of the isotope are factors that must be contended with due to the time required for the data acquisition, and dynamic studies are often impractical. Just as with the stationary-camera system, the rotating-camera system suffers from body scatter and attenuation and poor collimator resolution. The possibility exists, however, to combine the positive features of both the stationary and rotating camera systems to overcome the negative features associated with each through the use of a fixed-camera, multiple-view, coded-aperture imaging system.
Coded-Aperture Imaging

A coded aperture is typically a lead plate with a series of openings through which gamma-ray photons may be transmitted. The pattern of openings defines the "code" of the aperture. Systems employing coded apertures consisting of pinholes (Paxman RG et al., 1984, Paxman RG, 1984, Bizais Y et al., 1983, Tipton MD et al., 1976, Chang LT et al., 1974, and Dicke RH, 1968) annuli (Simpson RG, 1978), and Fresnel zone plates (Barrett HH, 1972 and Rogers WL et al., 1972) have been implemented. Information about the object being imaged is encoded by its geometrical shadow cast by the aperture onto the face of the detector. For example, a point source casts a geometrical shadow of the code on the face of the detector. The size of the shadow encodes the distance of the object relative to the aperture plane while the position on the detector encodes the lateral position of the source (see figure 2.3).

The advantages of using a coded aperture can be seen through the comparison between a parallel-hole collimator and a code consisting of a pattern of pinholes. A single pinhole in the code produces a projection of the object onto the face of the detector. This projection consists of a series of line integrals through the object that converge at the pinhole (recall that the projection of the object through a parallel hole collimator is formed by approximate parallel line integrals through the object). More importantly, the resolution of the pinhole falls off more slowly with distance from the collimator plane than does the resolution of the parallel-hole collimator (Barrett HH and Swindell W, 1981, pp.127,171). Furthermore, the fraction of photons transmitted by a single pinhole is comparable to, though somewhat smaller than, that transmitted by a collimator (Barrett HH and Swindell W, 1981). Therefore, if there are N pinholes in the aperture, then there is the potential for an N-fold increase in the number of photons transmitted. To gain an equivalent increase in the transmission efficiency of a parallel-hole collimator, the bore
size of the holes would have to be increased, but a larger bore hole would result in a loss of resolution. Unfortunately, an N-fold increase in the number of photons collected does not mean that a coded-aperture image will be N times better than an image obtained by more conventional means with fewer photons collected. This is because, although many more photons are collected in a coded-aperture system, many more are needed to overcome what is perhaps the greatest drawback of a coded-aperture system, multiplexing of the image data on the detector.

Multiplexing of the image data is caused by photons transmitted by several openings and detected by a single detector (many line integrals through the object converging on a single detector). As the number of pinholes is increased, the degree of multiplexing also increases (a single detector detects photons transmitted by even more apertures); therefore, more information, in the form of additional photons, is needed to decode the data. Along with the multiplexing of the image data, there are several other problems associated with coded-aperture systems. If the number of object voxels is greater than the number of detector elements, then there are many solutions that could result from the data set collected. The result is an inverse problem that is seriously underdetermined. Since there is not a unique object solution corresponding to the data collected, information known a priori about the object must be used in the reconstruction algorithm to develop a unique reconstruction. Therefore, in contrast to the filtered back-projection reconstruction algorithm commonly used in conjunction with the rotating-camera systems, a more general type of reconstruction algorithm is needed for use with a coded-aperture system. The reconstruction algorithm must be capable of decoding the multiplexed data and flexible enough to incorporate information known a priori about the object. One such algorithm that has proved particularly well-suited for this is simulated annealing.
Fig. 2.1. Anger camera -- top view and cross-section.
Fig. 2.2. (a) Conventional SPECT imaging utilizing (a) a rotating Anger camera (b) A parallel hole collimator assembly.
Fig. 2.3. Magnification and position of the image as encoded by the aperture.
CHAPTER 3
PARALLEL PROCESSING

During the early development of any new system, it is usual to find a wide range of innovative thought and design followed by a period dominated by a heavy investment in one particular type of design. After this period, further innovation tends to be very difficult due to the size of this initial investment. This pattern is particularly evident in the development of the computer. Many novel architectural principles for computer design were described in the 1950's, although until 1980, only systems of the von Neumann architecture had met with any commercial success. However, with the advances made in VLSI technology, and the advent of inexpensive microprocessors, architectures that once existed only in theory are becoming practical realities.

Evolution of the von Neumann Architecture

The first generation of computers (1950's) used vacuum tubes as their logic elements. The gate delay of such vacuum tubes, defined as the time taken for a signal to travel from the input of one logic element to the next, was approximately 1 microsecond. The UNIVAC, the first commercially available computer (1951) is a prime example of this technology. The second generation of computers (early 1960's) used discrete transistors of both germanium and silicon. This technology was exemplified by the IBM 7090 which used discrete germanium transistors with a gate delay of approximately 0.3 microseconds. The third generation of computers (1964-1973) was the first to utilize integrated-circuit (IC) technology. Incorporating silicon, bipolar, IC's at the level of small-scale integration (SSI), with a few gates per chip and a gate delay of approximately 10 nanoseconds, the Control Data Corporation (CDC) 6600 was introduced (the chief designer at Control Data at this time was Seymour R. Cray, the founder of Cray)
The fourth generation of computers (1970's) was marked by the advent of large-scale integration (LSI). ICs on silicon with gate-delay times on the order of one microsecond and near one hundred circuits per chip were utilized in computers such as the AHMDAHL 470 V/6 (1975).

By the 1980's, microprocessors with an operating speed and storage capacity equal to the first-generation, vacuum-tube computers were available on a single chip of silicon a few millimeters square. The trend of miniaturization continued with improvements in lithography process technology and fabrication techniques. Present-day lithography and process technology can achieve a resolution of 0.5 to 1.0 micrometers. Optical lithography can probably yield devices as small as 0.3 micrometers, and x-ray lithography can yield resolutions better than 0.1 micrometers.

During the period of development between 1950 and 1985, the speed of the components (as measured by the inverse of the gate-delay time) increased a thousand-fold; but the performance of the computer, as a function of number of operations performed per second, increased by a greater factor. The reason for this additional speed has been improvements in computer architecture, principally the introduction of parallelism.

Parallelism has been introduced into the von Neumann architecture in forms that are generally transparent to the user. For instance, some architectures utilize multiple memory banks that can be accessed independently. The two most common manifestations of parallelism and the use of co-processors and pipelining. The central processing unit (cpu) and a floating-point unit can function somewhat independently. Hardware is also available that allows program flow to overlap. For example, the prefetching of subsequent instructions as the preceding instruction is being executed -- this is termed pipelining. Though the introduction of these parallel aspects into computer architectures has increased the performance of the basic von Neumann architecture, the hardware
technology being implemented for a single processor is reaching its fundamental limits. However with the cost-of-logic constraint removed from architecture design, it is possible to incorporate many processors working in parallel to solve a problem.

**Parallel Architectures**

The most widely known classification of parallel architectures is given by Flynn's taxonomy (Flynn MJ, 1972). Flynn bases his classification on how the machine relates its instructions to the data being processed. Four broad classifications are defined based on how the instruction and data stream (or streams) are manipulated by a processor. (An instruction stream is defined as a sequence of instructions executed by a processor, and a data stream is defined as a sequence of data operated on by a processor.)

1. **SISD** -- single-instruction/single-data. This classification describes the conventional von Neumann architecture in which there is a single stream of instructions and therefore, in practice, only one processing unit. Each arithmetic instruction initiates one arithmetic operation, leading to a single stream of data and results. Computers falling under this classification have also been called scalar processors because each instruction is capable of manipulating only data items comprised of single numbers.

2. **SIMD** -- single-instruction/multiple-data. The SIMD architecture, the first real departure from the von Neumann architecture, is typified by the implementation of vector instructions. A vector instruction is defined as an instruction that initiates many arithmetic operations. In a SIMD computer, a single instruction fetch and interpretation produces many arithmetic operations, one for each element of the vector being operated on. The manner in which the vector instructions are implemented characterizes two distinct implementations of the SIMD architecture. In the first implementation, each
operation is performed on all data simultaneously. This architecture is used in computers such as the Connection Machine (Hill WD, 1985) and the ICL Distributed Array Processor (DAP) (Bowler KC and Pawley G, 1984 and Hockney RW and Jesshope C, 1981). The second implementation uses a pipeline to implement its instructions. The operation of a pipeline is similar in action to an assembly line -- the data move from station to station, a different operation being performed at each. The Cray-1 (Hockney RW and Jesshope C, 1981 and Kogge PM, 1981) and CDC Cyber 205 (Levine RD, 1982) are based on this implementation. It should be noted that due to the very broad nature of Flynn's categories, some architectures, such as the pipelined processors, do not fit neatly in any category. SIMD, vector, and array processors are, in some circles, regarded as synonymous. This school of thought arises due to the vector nature of the data operated on by the processor -- vectors are one-dimensional arrays or arrays are multidimensional vectors. A broad definition of a vector processor is that it contains special hardware that allows a sequence of identical operations on data arranged in a regular array to be performed faster that a sequence of nonidentical operations. A SIMD architecture means that a single instruction causes the execution of identical operations on multiple data. So SIMD, vector, and array processors are somewhat synonymous when looked at this way (Almasi GE and Gottlieb A, 1989, p. 303).

(3) MIMD -- multiple-instruction/multiple-data. The MIMD architecture generally consists of several processors, each with its own instruction and data stream, connected via a communications network. The number and type of processors used, the memory distribution among the processors, and the network by which the processors and/or memory elements are connected provide a myriad of design possibilities. Computers such as the Texas Reconfigurable Array Computer (TRAC) (Lipovsky GJ and Malek M, 1987) and TRIMM are examples of the MIMD architecture.
(4) MISD -- multiple-instruction/single-data. The MISD architecture is the least investigated parallel architecture. As the name implies, such a machine would implement several instructions operating simultaneously on a single data item. Although few machines have been designed to specifically operate in this fashion, a MIMD machine could be used in this capacity by simply allowing each processor to perform the same operations independently on the same data.

SIMD
Pipeline Operation

A broad definition of a vector processor is that it contains special hardware that allows a sequence of identical operations on data arranged in a regular array to be performed faster than a sequence of nonidentical operations; however, with the advent of supercomputers like the CRAY series, a vector processor has come to mean specifically a pipelined machine designed to do well on floating-point operations. Some of the key aspects of vector performance can be understood through a comparison of several operational cases -- the scalar case, pipeline case, and the parallel case -- in a simple analytical model presented by Hockney (Hockney RW and Jesshope C, 1981).

In the "ideal overlap" scalar case (depicted in fig. 3.1a), where instruction prefetching and pipelining have made it possible to execute one instruction per cycle, the time needed to perform a vector operation is

$$T_{\text{serial}} = (s + nq)\tau,$$

where $s$ is the setup time, $q$ is the number of suboperations performed on each pair of vector elements, $\tau$ is the clock time, and $n$ is the vector length. The value of $q$ depends on the operation being performed. The average execution time per element for large vectors approaches $q\tau$, that is,
The time required for pipelined execution of a vector (as shown in fig. 3.1b) is

\[ T_{\text{pipe}} = (s + q)\tau + (n - 1)\tau, \]

where the first term is the time to start and fill the pipe and produce the first result, and the second term is the time to produce the remaining \((n-1)\) results. For long vectors \((n \to \infty)\), the average execution time per pair of vector elements approaches \(\tau\), that is,

\[ \bar{t}_{\text{pipe}} = \frac{T_{\text{pipe}}}{n} \to \tau \text{ as } n \to \infty. \]

The speedup over the scalar case, namely \(\bar{t}_{\text{scalar}}/\bar{t}_{\text{pipe}}\), approaches \(q\). If a pipeline stage exists for each suboperation to be performed, then the speedup is equal to the number of pipeline stages.

In the parallel case (fig. 3.1c), the time for one or \(n\) vector elements is the same, as long as \(n\) does not exceed the number of processing elements. The vector execution time is

\[ T_{\text{parallel}} = (s + q)\tau, \]

and the average execution time per pair of vector elements is

\[ \bar{t}_{\text{parallel}} = \frac{1}{n} (s + q)\tau. \]

The processing rate is just the inverse of \(\bar{t}\) in each case. An empirical measure often associated with pipelines is the asymptotic processing rate \(r^*\), the peak processing rate that can theoretically be achieved for very long vectors.

**CRAVY-1**

Overall, the CRAY-1 architecture comprises a main memory feeding data to and from a set of scalar and vector registers. The arithmetic and logic operations are
performed by 12 independent functional units working to and from the registers. All functional units are pipelined and may operate concurrently, with pipeline stages of lengths varying from 2 to 14. For the CRAY-1, \( r_- \) is 80 MFLOPS/pipeline. It should be noted, however, that this is not the peak rate for the entire machine. A special feature of the CRAY-1 architecture is the ability to chain together a series of vector operations so that they operate together as one continuous pipeline. This "chaining" of separate pipelines is implemented by allowing the same register to serve as the output for one vector operation and the input for another. If two vector operators use two separate functional units, then the second operation can start as soon as the first result from the first operation leaves its functional unit. This optimization is equivalent to constructing a new pipeline to compute the combined functions of each pipeline without paying a second startup time (only the compute time of the second operation is added). Thus, due to chaining two pipelines together, the entire-machine peak rate is 160 MFLOPS. Peak MFLOPS is, however, a purely empirical measurement and is almost never met due to factors such as operations being of a memory-to-memory type, rather than of the register-to-register variety, vector lengths being greater than the number of processing elements, and inefficiencies due to compilers.

The CRAY-1 memory-register data path may be visualized as an 11-stage pipeline, 64 bits wide, transferring data in only one direction at a time. The resulting bandwidth of 80 Mwords/s for data transfer between the registers and main memory is low for a computer with the arithmetic speed of the CRAY-1. An arithmetic operation such as floating-point multiplication has two arguments and one result. If all data are to be read from and returned to main memory, the memory bandwidth should be three times the computing rate. The maximum computing rate on the CRAY-1 is 160 MFLOPS/s, hence the bandwidth required on the above basis is 480 Mwords/s. Although the memory bandwidth of 320 Mwords/s almost meets this requirement, the limited width of the data
bus to the registers allows only 80 Mwords/s to be used for computational purposes. Thus, at best, the CRAY-1, uses only one-sixth of the bandwidth required to support its computational capability. Of course the machine is designed on the basis that many arithmetic operations may be performed on data that are resident in the registers before the results are returned to main memory. Careful assembler coding can sometimes achieve this result, but the performance obtained from FORTRAN code is often less than hoped for because of the bottleneck caused by the relatively low register-to-main-memory bandwidth.

The effects of the architectural structure on the peak MFLOPS performance is evidenced by considering a simple vector multiplication. The value of r\textsuperscript{DO} is measured at 22 MFLOPS, degraded to 28% of its peak value of 160 MFLOPS. This degradation is due to two factors: the finite register length and the fact that only one read or write per cycle is allowed (Hockney RW and Jesshope C, 1981).

The Connection Machine

Although the Connection Machine belongs to the same classification (SIMD) as the CRAY series, its architecture and design philosophy are in stark contrast to that of the "vectorized" machines. One view of the Connection Machine is that it attempts to solve the von Neumann bottleneck by replacing one large processor communicating with one large memory with many small processors distributed throughout the memory (in fact, it was originally called the Connection Memory (Hillis WD and Memo AI, 1981).

The Connection Machine incorporates 65,536 1-bit processors into its architecture, and was not designed to be a "number cruncher" like the CRAY-1, but instead to be a "symbol cruncher", i.e., it was designed to answer questions more along the lines of "Is Tucson hot?" than "What proton mass does QCD predict?". Thus it is best suited for problems that can benefit from parallel pattern-matching search algorithms.
rather than algorithms that are heavy in floating-point computations. This approach represents a highly parallel computational model termed connectionism. Connectionism suggests that instead of representing pieces of information as passive patterns that are operated on by a program, information should be represented as simple, active, computing elements that interact in parallel by exchanging simple messages.

For example, consider the retrieval of documents from a large collection of texts. Each processor can be chosen to represent a particular document in the data base. A "search sample", chosen for its particular relevance, is broadcast to all the processors in the network, and each processor compares the document it represents with the search sample. These comparisons are all done simultaneously, and once completed, the processors can exchange information and rank the documents according to how well they match the search sample.

The architecture of the Connection Machine divides the 65,536 processors into 4096 groups of 16 processors. Each group of processors, along with a single message router and instruction decoder that serve the entire group, occupies a single 1-cm square chip. The processors within a specific group are connected in a daisy-chain pattern, and the entire collection of processors is connected in a 256 x 256 grid. Communications between processors on different chips are processed through the routers. A message containing the address of the destination processor is passed along the grid in the direction of its destination. Along the path toward the destination processor, each router encountered reads the message and passes the message on toward its destination address. (Although it has not been used extensively to date, it is also possible to reconfigure the interconnection network in the form of a 12-dimensional hypercube. An n-dimensional hypercube is a cube in an n-dimensional space and will be discussed in detail in the succeeding section on MIMD interconnection networks).
Although not specifically designed to be a number cruncher, the Connection Machine, for specific types of problems, has the potential to be just that due to the sheer number of processors of which it is comprised. Consider the case of processing an image consisting of 65,536 pixels. A conventional computer operates on only one pixel at a time, so that even a simple image-processing operation includes 65,536 steps. The Connection Machine, on the other hand, assigns a single processor to each point of the image. If the operation being performed on the image is a simple one that requires no communication among the processors, then the time required to process the image is the same as that required to process a single pixel of the image (albeit with a simple single-bit processor). For example, to find all the points in the image that are brighter than a certain minimum, a sequential machine must check the 65,536 pixels in succession, comparing each one with the threshold value. In the Connection Machine, that comparison is made simultaneously by the 65,536 processors - each one operating on a single pixel of the image. Most interesting computations, however, require that the processors exchange information as the operation proceeds, therefore requiring some form of an interconnection network. A simple convolution operation, for example, blurs a two-dimensional image by averaging each point with its four nearest neighbors; this operation is particularly suited to the interconnections provided by the two-dimensional grid.

The early demonstrations of the Connection Machine have not been on numerical problems, but rather on applications such as document retrieval. One performance example cited by Hillis (Hillis WD, 1985) is the retrieval of a news story matching some key words from among 50,000 news stories and the subsequent retrieval of all related stories in times as short as a few milliseconds. These demonstrations, although utilizing a single processor per data item, have relied primarily on the grid interconnections for communications among processors. Many of the results are therefore applicable to other
massively parallel SIMD machines made up of large arrays of small processors such as the ICL DAP (Bowler KC and Pawley G, 1984) (a 64x64 array of 1-bit processors each with 4 Kbits of memory and a 200 nanosecond clock cycle) and the Goodyear Massively Parallel Processor (MPP) (Batcher K, 1980) (a 128 x 128 grid of 1-bit processors each with 1024 bits of memory and a 100 nanosecond clock cycle).

**MIMD**

By definition, MIMD machines are comprised of multiple processing elements operating on multiple data streams. Unlike the processors of the SIMD machines, each processor in a MIMD machine is able to execute its own program independently of the other processors. For the majority of the MIMD machines in use, it is assumed that the task at hand can be divided into several subtasks that can be distributed among the processors for execution. This "divide-and-conquer" approach to problem-solving creates some new concerns that were not apparent in a more conventional sequential approach. First, and perhaps most importantly, the algorithm must lend itself to the particular type of parallelism provided by the architecture. If, for example, the number of subtasks does not correspond to the number of processors available, or the execution times of the subtasks are not all the same, then the subtasks must be scheduled on the processors in a way that efficiently balances the load on the processors. Once executing on the processors, the subtasks will, except in the most ideal case, need access to various system resources as well as to each other; therefore, a communications network is essential.

There is a growing awareness that data communications is the key to successful exploitation of parallelism. Making the right data available to the right processor at the right time is essential to keeping all the processors busy, and this is impossible without the appropriate communications paths.
Interconnection Networks

The vast number of MIMD machines discussed in the literature form a confusing menagerie of computer designs, and it is often the communication network that serves as the primary distinguishing characteristic among machines. The communications network communications can generally be divided into two broad categories -- those with a separate and identifiable switch (switched-network MIMD) and those in which the processing elements are connected by a recognizable, fixed network (fixed-network MIMD). In the former, all connections between the processors are made via the switch, which is usually quite complex and a major part of the design. In the latter, individual processors can communicate directly only with their neighbors in the network, and long-range communication across the network may require the routing of information via a large number of intermediate processors. This division is shown in the chart in figure 3.2.

Switched networks can be further subdivided into those in which all the memory is distributed among the processors as local memory, with the processors communicating via the switch (distributed memory); and those in which the memory is a shared resource that is accessed by all the processors through the switch (shared memory). It is not uncommon, however, for a system to have both a shared, common memory and distributed, local memory. A further subdivision, applicable to both distributed-memory and shared-memory systems, is made according to the nature of the switch. Examples will be given of, and comparisons made between, such dynamic networks as the conventional bus, crossbar network, and several of the multistage switching networks (Omega, Benes, and Banyan).

Most of the fixed-network MIMD systems are of the distributed-memory variety. The networks, however, can be subdivided according to the topology implemented by each. Fixed networks range from simple linear chain and star configurations to
multidimensional meshes including the hypercube. Examples also exist of hierarchical networks based on trees, pyramids, and bus-connected clusters.

The most suitable computer network for a given application will certainly depend on the nature of the problem to be solved, but the basic measure of a communication network is how quickly it can deliver what is needed to the right place reliably. The following terms, according to Feng (Feng T, 1981), can be used to define performance criteria:

Latency: the transit time for a single transmission.

Bandwidth: how much transmission traffic the network can handle.

Connectivity: how many immediate neighbors each node has (the node's degree) and how often each neighbor can be reached.

Hardware cost: what fraction of the total hardware cost the network represents.

Functionality: additional functions performed by the network such as arbitration and combining of messages.

The first three performance criteria above would be optimized by a fully connected network. A fully connected network, as pictured in figure 3.3, is a network in which every node is connected to every other node. The fully connected network provides a uniformly fast access time to any other node (the path length between every node is
identical). If the network consists of N nodes, each node is of degree N-1 and is provided with N(N-1)/2 connections. Unfortunately, the cost of this network grows as N², rendering it impractical for more than a few nodes. The fully connected network does, however, provide a benchmark by which other interconnection networks can be measured.

**Fixed Networks**

The simplest fixed network is the one-dimensional mesh, or linear chain, as pictured in figure 3.4a. Interior nodes have a degree of two, and the boundary nodes have a degree of one. If the two boundary nodes are connected, the result is a ring with all nodes of degree of two (fig. 3.4b). Higher-dimensional meshes are configured analogously to the one dimensional mesh: with k dimensions, interior nodes have a degree 2k (if the outer nodes are connected as in the ring configuration, then all nodes are of degree 2k).

The star topology (fig. 3.4c) consists of a single central node connected to all the other nodes. Since the central node has degree N-1 for an N-node star, large configurations are impractical.

In the basic binary tree (fig. 3.5), interior nodes have degree 3 with two children (connected nodes downward), and one parent (connected node upward); uppermost nodes in a branch (leaves) have degree 2, and the bottom-most node (root) has degree 1.

Routing data from one node to another for each of the topologies described so far is a simple matter. For the mesh and ring topologies, the routing is simply performed one dimension at a time, the message being relayed from node to node until the destination node is reached.

If either of the nodes communicating in the star topology is the central node, then the routing is identical to that of the fully connected topology; if not, then the path
consists of two path-lengths, the first connecting the node of origin to the central node
and the second connecting the central node to the destination node.

Routing in the tree network consists of detecting the lowest ancestor of the
destination node while traveling upward from the node of origin, and then determining
whether to turn left or right while traveling downward toward the destination node.

A network of great current interest is the binary hypercube -- a multidimensional
cube. If each of the nodes is viewed as the corners of a cube in k-dimensional space, the
connections are the edges of the cube. In a k-dimensional hypercube, there are \( N = 2^k \)
nodes, each of degree k (the \( k = 3 \) case is illustrated in fig. 3.6). If the processors are
numbered from 0 to \( 2^k - 1 \), as in figure 3.6, nodes whose binary numbering differs in
exactly one position are connected. With this numbering scheme, routing among the
various nodes is a relatively simple procedure. Given two nodes, one being the
originating node and the other the destination node, the exclusive or of their node
numbers contains a 1 in precisely those dimensions in which the numbers differ, and thus
indicates the dimensions along which the data must travel to complete a communication.
The routing algorithm, therefore, contains k steps: during the \( i^{th} \) step \( (i \leq k) \), the data are
sent to the adjacent node in dimension \( i \) if the \( i^{th} \) bit of the exclusive or is 1; otherwise
the data remain where they are. Since \( N = 2^k \), the routing takes on the order of at most
\( \log_2 N \) steps.

A slight variation on the hypercube topology is the cube-connected cycle. It has
been observed that many of the connectivity and routing advantages of a hypercube can
be maintained while preventing the node degree from growing as fast as the dimension of
the cube (Preparata F and Vuillemin J, 1981). Each node of the original \( k \)-dimensional
cube is replaced by a cycle of \( k \) nodes, resulting in a total of \( k2^k \) nodes. If the cycles are
numbered from 0 to \( 2^k - 1 \) (as the individual nodes were numbered in the hypercube) and
the nodes on a cycle from 0 to \( k-1 \), then cycles whose numbers differ by \( 2^i \) connect node \( i \)
on these cycles. Thus the k connections to each node on the original hypercube have been
distributed to the k nodes on the cycle replacing this node. The k = 3 cube-connected
cycle configuration is depicted in fig. 3.7.

Routing for the cube-connected cycles is slightly more involved than for the basic
hypercube configuration. Preparata and Vuillemin (Preparata F and Vuillemin J, 1981)
show that the routing can be accomplished within 2.5 log2 N steps. Recall that a k-
dimensional cube-connected cycle is a k-dimensional hypercube in which each vertex
contains N = k2^k nodes. A transmission is possible from any cycle to any other cycle
within 2k steps by alternating movement along the cycles with movement across the
hypercube if necessary. Once at the desired cycle, a path to the destination node can be
found with length not exceeding 0.5k and thus the entire path from source to destination
has length no more than 2.5k < 2.5 logN.

The perfect shuffle (originally called the faro shuffle) was used for many years by
magicians when performing card tricks. Imagine shuffling a deck of N = 2^d cards
numbered from 0 to N-1 in binary form. This action interleaves cards in the top half of
the deck (i.e., cards whose numbers begin with 0) with cards in the bottom half of the
deck (cards whose binary numbers begin with 1). With this numbering scheme, the
shuffle can be accomplished by rotating the card numbers one bit to the left and
reordering the cards based on their new numbers. A k-way shuffle network contains N =
k^d nodes each connected to 2^k. Thus nodes x and y in the 2-way shuffle are connected if
and only if one is the shuffle of the other.

Switched Networks

The difficulty of realizing a multipurpose parallel computer has given rise to
much research on dynamic interconnection networks. Just as with the fixed networks,
there is a wide range of performance and cost for each approach. The architectures
available range from the standard bus architecture to what is perhaps the best approximation to a fully connected network, the crossbar switch. Several multistage switching networks have characteristics in between these extremes. Like the static topologies, the limiting factors of the architectures at each end of the spectrum make them impractical for implementations involving large numbers of nodes.

The bus, which has been the predominant interconnection network for the past quarter century, is simple in concept, well understood, and readily assembled from existing technology. The bus works well as long as it services only a small number of processing elements. For a large number of processing elements, the performance of the bus is limited by its bandwidth (the bandwidth of a bus is simply the product of the bus-cycle frequency and the width of the data path). The bandwidth available to each processing element on the bus is inversely proportional to the total number of processing elements on the bus.

Bus widths over 72 bits are rare, and the only other way to increase the bandwidth is by increasing the clock frequency. But the same technological advances that make higher bus clock rates possible will also make faster processors possible, so the ratio between processor power and bus bandwidth is likely to remain roughly the same. Thus the number of processors that can be supported on a single bus will remain limited.

A crossbar switch (fig. 3.8), in contrast to the bus, allows any processing element to contact any other nonbusy processing element at any time. The crossbar switch is the best approximation, in terms of connectivity and latency, to the fully connected network. However, since a crossbar switch is basically a grid of logic gates, as more processing elements are added the capacitive loading is increased, resulting in a greater switch delay. The switch cost increases as well, scaling as $N^2$.

Multistage switching networks (Benes VE, 1965, Patel JA, 1981, Feng T, 1981, and Wu C and Feng T, 1980) have characteristics that fall between those of the bus and
the crossbar network. Networks such as the Omega, Benes, and Banyan are all collections of 2 x 2, or slightly larger, crossbar switch elements, arranged in an array whose dimensions are close to Nlog₂ N. Each is an attempt to approximate the connectivity and throughput of a crossbar switch while reducing its cost scaling factor from N² to Nlog₂ N. This comes at the price of an increase in the network latency on the order of log₂ N. That is, the processors are still equidistant from each other in communication terms (unlike static topologies), but that distance is roughly log₂ N times greater than in a crossbar network. Each of these networks is capable of connecting a single input terminal to any output terminal. Their designs, however, differ in connection topology, operation mode, control strategy, and the nature of the switch boxes.

The most basic of the multistage switching networks is the omega network (Lawrie HL, 1975). Its operation is simple, and it is the basis upon which several of the other networks are built. The omega network provides exactly one connection from every input to every output. It consists of 2x2 crossbar switches that can provide the connections shown in figure 3.9a. The interconnection topology, depicted in figure 3.9b, consists of log₂ N columns, each containing N/2 switch boxes interconnected in perfect-shuffle wiring patterns.

The switching method used in the omega network is termed packet switching. A packet, consisting of an address and data, is sent through the system. The address is read at each node and is then forwarded to the next node according to some control scheme until it arrives at its destination.

The omega network's sparseness and simplicity make a simple routing algorithm possible and keep the network small, cheap, and fast, but the price paid is a high probability of "blocking" among potential message paths. Blocking refers to certain combinations of input-output connections that the network cannot achieve because one
path blocks another (a crossbar switch, for example, is a nonblocking network because it can connect any input to any output at any time).

The Benes network (Benes VE, 1965) belongs to a category with characteristics that fall between those of the crossbar and omega networks, referred to as rearrangeable nonblocking networks. Unlike the omega, these networks have multiple ways to "get from here to there". The Benes network, as pictured in figure 3.10, is essentially an omega network with the added hardware necessary to establish many possible paths from each input to each output. The potential exists to achieve all the permutations of a true nonblocking network like the crossbar, but only if all the desired connections are specified before the routing computation begins. That is, with a crossbar switch, one can make the connections in an arbitrary time sequence. If this is attempted with the Benes network, the desired connection can be made, but some of the connections made previously may have to be rearranged to do it. There is a price for this reduced blocking and increased bandwidth however: the additional stages increase the size, cost, and latency of the network. In addition, the connection pattern takes time to calculate.

A Banyan network (Feng T, 1981) (fig. 3.11) is essentially a tree with each input node given its own tree connection, sideways, to all the output nodes; however, the trees are interlinked and not independent. Like the omega network, a unique path from each input to each output exists in the Banyan network, but the number of the outputs of the switch boxes is not equal to the number of the inputs. A comparison of the latency, bandwidth, connectivity, and cost of the various networks, both fixed and switched, is given in the chart shown in figure 3.12 and figure 3.13, respectively.

**Parallel Programming**

The need for the an architecture that corresponds efficiently to the algorithm is an accepted principle for architectural design, but just how that algorithm is expressed for
The various parallel architectures require not only new ways of performing the computations and new ways to execute algorithms, but in many cases new ways of thinking about the algorithms. The basic challenge that confronts the parallel programmer is how to transform the structures of the job at hand so that it matches the structure of the parallel computer that will run the job. Each method of programming is a different way of expressing a parallel solution to a problem. The ultimate criteria of the programming means and how efficiently the program communicates the problem to the parallel architecture used and how easy the means can be manipulated by the programmer.

To map a problem onto a parallel architecture, the programmer must first divide the problem into segments that will execute in parallel, and then, in most cases, determine how the processing elements will communicate and synchronize with one another. This same procedure is followed no matter what the problem is and regardless of the parallel machine that will execute the problem.

Programs exhibit many levels of parallelism and in most cases the programmer will have several choices of how to distribute the work among the processing elements. The idea is to determine the level of parallelism that closely matches the architecture being used. A commonly used measure of parallelism, in both hardware and software, is termed granularity. Parallel architectures, in terms of the processing elements used, can be described as coarse-grained or fine-grained machines. A coarse-grained machine is composed of large, powerful processors, generally with a substantial amount of local memory, whereas a fine-grained machine utilizes much smaller, simpler processors with little local memory. In terms of a parallel program, the granularity is an indicator of how much computing each processor can do independently, in relation to the time it must spend exchanging information with the other processors (Seitz CL, 1985). A coarse-
grained application can be divided into large independent operations, with little synchronization or communication. A fine-grained application, on the other hand, is comprised of a smaller sequence of steps that can be executed independently before communicating with other processors. In other words, a coarse-grained application has a much higher ratio of computation to communication than a fine-grained application. The granularity of an application is sometimes determined by the degree of interaction within a problem and sometimes by the architecture used.

Once the granularity of a problem has been determined, the next step is to find the best form of communication between processors. As with the granularity of the problem, the means of communication may be dictated by the architecture. The communications network and the memory configuration of the machine used generally define the communications options available to the programmer. In a shared memory system, the processing element generating a piece of data simply writes it to a shared location and any processing element that needs the data can read it. In a distributed-memory system, the communication is conducted on a point-to-point basis. Data are generated by a source processor and delivered to a destination element. If there is no direct communication path between source and destination, data are routed via intermediate processors.

The method used to synchronize processors depends on the approach taken to communication. In shared-memory communication, synchronization between a writing and reading processor must occur only once; once data has been written, it can be read at any time. Programming techniques such as locks (a variable that indicates whether or not another process is using the data) and semaphores keep writers and readers in step (Darema F et al., 1987). For a distributed memory system, synchronization is implicit. A processor that is to transmit some information must suspend operation until the processor that is to receive the data is ready. While implicit synchronization makes the parallel programming easier, synchronization with each transmission may impose excessive
overhead, lowering the program efficiency. Once these factors have been determined --
the level of parallelism to be used, the hardware configuration, and the amount of
communications and synchronization required by the implementation -- the problem can
be transcribed in such a manner as to constitute a parallel program.

A parallel program differs from a sequential one as radically as a parallel
computer differs from its sequential counterpart. A conventional program does one thing
at a time, but a parallel program must create and manage many simultaneous threads of
activity. It must divide a task into many parts, and it must coordinate them by establishing
an orderly flow of instructions, data, and results.

Conventional languages lack the vocabulary and syntax for specifying
parallelism. They enable a programmer to write, "Do A, then B and then C" but not "Do
A, B, and C at the same time". A programming language is the expression of a particular
software model -- a particular approach to constructing programs. In an effort to develop
a means for programming in parallel two distinct schools of thought have arisen: the
exploitation of the inherent parallelism in existing software and the development of
parallel software models.

The first approach, driven by the "dusty deck" (the "dusty deck" is a reference to
an old deck of punched cards) problem -- the large investment of time and money
represented by existing programs -- avoids the question of developing a parallel software
model altogether. Instead it focuses on transforming sequential programs into parallel
programs through the use of a parallelizing compiler (Padua DA and Wolfe MJ, 1986). A
parallelizing compiler finds the parallelism hidden in a sequential program by searching
its text for operations that can be done simultaneously. The compiler then generates code
that reflects the implicit parallelism it has found. This approach is fairly robust in that a
cross-compiler may allow a specific program to run on several different machines, but the
parallel implementation produced is not necessarily the most efficient parallel implementation of the algorithm.

Dataflow computers offer another way to take advantage of implicit parallelism. These machines are usually not aimed at the dusty-deck problem, since they tend to be conceived together with a new style of programming. But they address the same basic question: how to avoid explicitly parallel software models. A dataflow-machine program does not specify an execution sequence. The only constraints on its order of execution are those that are forced by the program's logical structure. Any two instructions can be executed by separate processors simultaneously, unless one instruction needs a value the other is directly responsible for computing. The data flow from instruction to instruction as the program executes. Thus (in a sense) the computer itself, and not the program, sets up the parallel threads of activity. The languages that are usually associated with dataflow machines, "functional languages," make it easier for the machine to decompose the program into parallel activities during execution, than if the program were written in a more structured language. These languages simplify the data relations among statements by allowing a programmer to set the value of a variable only once. After a variable's value has been set, other parts of the program can use the variable freely and simultaneously without worrying about whether and when its value will change (Gerlernter D, 1986 and Gerlernter D et al., 1987).

The most well known of these languages is called Linda (Gerlernter D, 1986 and Gerlernter D et al., 1987) and is based on a construct called tuple space. A tuple is simply an ordered set of elements. Tuples can be either active (a process) or passive (data). Active tuples do the work of computation and communicate by generating, reading, and consuming passive tuples while passive tuples hold the data structures. Passive tuples appear and disappear as the program executes. Active tuples that need data can either read passive tuples or "consume" them so that particular tuple no longer exists. As active
tuples develop results, they either generate new passive tuples or they themselves become
a passive tuple by holding their result. In Linda, then, "a parallel program is a swarm of
simultaneously computing active tuples surrounded by and ultimately blending into a
haze of passive tuples" (Gelernter D, 1986). A Linda program for matrix multiplication
might store the rows of one matrix and the columns of the other one in passive tuples, one
row or column per tuple. Active tuples repeatedly read a row and a column, compute their
inner product, and dump the result in a new passive tuple.

A tuple space is created more easily on a shared-memory system. The tuples are
kept in the memory shared by all the processors and active tuples are assigned as the
processors become free. Each active tuple has access to all the passive tuples, which are
stored in the shared memory as well. Creating a tuple space on a distributed-memory
machine is a harder problem, but a new distributed-memory machine is being built
specifically to run the Linda language.

The second school of thought is the development of software models for parallel
programming. One type of model for parallel programming contains the same constructs
as its sequential counterpart, but has added constructs to support the needs of parallel
computations. There are now parallel Fortran and C compilers available as well as
languages written specifically for parallel programming, such as Inmos Occam. The
extended Fortran and C languages have the same syntax as their serial versions, but
provide the programmer with the additional constructs needed to establish
communications between processes. This method, although requiring more "thought"
overhead, give the programmer explicit control, within the bounds allowed by the
program, over the degree of parallelization introduced into the program.
Fig. 3.1. Scalar, pipelined, and vector processing. (a) Four suboperations needed for floating-point addition. The time required for the addition of a 12-element floating-point vector, $T(\text{scalar})$, on a high-performance scalar machine is depicted in (b). The time required to perform the same addition, $T(\text{pipeline})$, on a pipeline vector processor is shown in (c); pipelined execution is achieved by overlapping the suboperations of the addition. The time required to perform the addition in a truly parallel fashion, $T(\text{parallel})$, is shown in (d).
Fig. 3.2. A structural taxonomy of MIMD computer systems.
Fig. 3.3. Fully-connected network. \( \square \) denotes the individual processing elements.
Fig. 3.4. (a) Linear chain, (b) ring, and (c) star fixed-network topologies. ☐ denotes the individual processing elements.
Fig. 3.5. Binary-tree fixed-network topology. \( \mathcal{P} \) denotes the individual processing elements.
Fig. 3.6. Hypercube fixed-network topology. $P$ denotes the individual processing elements.
Fig. 3.7. The cube-connected cycles, a variant of the hypercube. P denotes the individual processing elements.
Fig. 3.8. Crosspoint switch connecting any processor ♂ to any memory module ♦.
Fig. 3.9. Omega network. (a) The four switch functions for an omega network switchbox. (b) The Omega network topology.
Fig. 3.10. Benes network. The dotted line indicates a connection that was not possible in the "blocking" Omega network of figure 3.9b.
Fig. 3.11. Banyan network.
<table>
<thead>
<tr>
<th>Network</th>
<th>Maximum Latency</th>
<th>Maximum Bandwidth per PE</th>
<th>Connectivity (neighbors)</th>
<th>Wire Cost</th>
<th>Switch Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>crossbar</td>
<td>constant</td>
<td>constant</td>
<td>any to any non-busy</td>
<td>N</td>
<td>(N^2)</td>
</tr>
<tr>
<td>bus</td>
<td>constant</td>
<td>1/N</td>
<td>any to any if bus free</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>multistage</td>
<td>log N</td>
<td>constant</td>
<td>any to any non-busy if not blocked in network</td>
<td>N log N</td>
<td>N log N</td>
</tr>
</tbody>
</table>

Fig. 3.12. Tradeoffs among switching interconnection networks.
<table>
<thead>
<tr>
<th>Network</th>
<th>Maximum Latency</th>
<th>Maximum Bandwidth per PE</th>
<th>Connectivity (neighbors)</th>
<th>Wire Cost</th>
<th>Switch Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>all-to-all</td>
<td>constant</td>
<td>constant</td>
<td>any-to-any</td>
<td>N^2</td>
<td>---</td>
</tr>
<tr>
<td>hypercube</td>
<td>log N</td>
<td>constant</td>
<td>any-to-any</td>
<td>N</td>
<td>---</td>
</tr>
<tr>
<td>cube connected cycles</td>
<td>log N</td>
<td>1/N</td>
<td>any to any if bus free</td>
<td>N</td>
<td>---</td>
</tr>
<tr>
<td>tree</td>
<td>log N</td>
<td>constant</td>
<td>any to any non-busy if not blocked in network</td>
<td>N log N</td>
<td>---</td>
</tr>
<tr>
<td>mesh</td>
<td>N^{1/2}</td>
<td>constant</td>
<td>4-8 nearest neighbors</td>
<td>N</td>
<td>---</td>
</tr>
</tbody>
</table>

Fig. 3.13. Tradeoffs among fixed interconnection networks.
CHAPTER 4
TRIMM

TRIMM is a computing array comprised of 60 Inmos T800 transputers and a Motorola 68020 processor acting as the system host. A block diagram of the overall system architecture is pictured in figure 4.1. TRIMM is a hybrid shared-memory/distributed-memory machine with each transputer having 1 Mbyte of local DRAM (Dynamic Random Access Memory) that is also accessible by the host via a VME bus interface. Interprocessor communications among the transputers is made possible through the reconfigurable link connections (discussed below). Due to the high demand for random numbers in Monte Carlo calculations, each transputer has high-speed access to random numbers from a bit stream generated by a hardware random-number generator.

The transputer is a 20 Mhz, true 32-bit processor that utilizes a RISC-like architecture and is designed specifically to be used in parallel architectures. Figure 4.2 shows a block diagram of the internal architecture of the transputer. Unique features of the transputer include an on-chip floating-point processor and 4 high-speed serial links. A link consists of two unidirectional, serial pathways - one for input and one for output - capable of transmitting data at 20 Mbits/sec. Each link is also equipped with its own on-chip DMA controller to facilitate asynchronous communications. A single transputer can be connected to as many as four other transputers, providing a total communications bandwidth of up to 160 Mbits/sec-transputer.

The host processor in TRIMM can act as the global manager of the array by taking responsibility for data collection and distribution. The host can access each local memory by simply reading from, or writing to, the desired memory location. Although only a single memory location can be read at a time, memory can not only be written to
on an individual location basis, but the same locations in the memories of every transputer can be written to simultaneously using a broadcast addressing mode. The host can also act as the user interface to the system.

Six transputer modules, each consisting of a transputer and a megabyte of memory (see fig. 4.3), are placed on a single printed-circuit card and are hardwired together in a 3x2 array (this configuration is depicted in fig. 4.4). Four of the ten unconnected ten links can be used to connect the boards together via front panel jumpers, thus forming a 2x30 array. The remaining 6 links, one per transputer, are connected to the link interconnection bus which, through the selection of fixed hardware jumpers, allows any link to be connected to any other link. The transputers can therefore be connected in several configurations such as a 6x10 mesh array or various pipeline, ring, and binary tree arrangements. The hardware random number generator resides on a separate printed-circuit card and supplies random numbers for the entire system via the VME bus. A wideband noise source provides a source of Gaussian random noise that is digitized by high-speed 8-bit flash A/D converters. Each transputer receives a stream of random bits in a 16-bit shift, register and when the processor requires a random number for calculation, it simply reads the contents of the shift register to obtain a 16-bit uniformly-distributed random number.

The communications network of TRIMM depends on the configuration of the transputer modules and the communications pathways used. Hence the performance criteria for a communications network, as defined by Feng (discussed in chapter 3), will vary from configuration to configuration and be a function of the communications methods used. For example, if the broadcast mode is used, then the latency of a communication to all processors from the host is essentially nonexistent. If only the transputer array is used, then some latency will indeed be evident, but if the algorithm being executed is not highly dependent on communications (asynchronous operation) for
performance, then the latency is again negligible. Likewise, the bandwidth is dependent on the configuration used, but theoretically a bandwidth of 8600 Mbits/sec (60 processors x 4 link/processor x 20 Mbits/sec-link x 2 directions) could be achieved if each transputer is fully connected. The connectivity can be as high as 4 if all the links are utilized and the cost of the network scales as the number of processors and associated local memory (there is some overhead for the VME interface logic, but only one interface is needed to serve 6 modules).

The primary consideration in implementing a problem in parallel is how best to map the problem onto the parallel architecture being used. TRIMM, through its ability to reconfigure the transputer array and the communications alternatives, allows the operator several degrees of freedom in finding the best architecture and communications scheme for the problem to be solved.
Fig. 4.1. Overall system architecture for the TRIMM parallel processor.
Fig. 4.2. Block diagram of the internal architecture of the Inmos T800 transputer.
Fig. 4.3. Block diagram of a TRIMM transputer module.
Figure 4.4. Serial-link connections between transputers.
CHAPTER 5
IMAGE RECONSTRUCTION AND SIMULATED ANNEALING

The nature of tomographic imaging renders a data set that is often underdetermined and underconstrained. The image-forming process is described by the matrix equation

\[ g = Hf + n, \]  

(5.1)

where \( g \) represents a vector of data values (lexicographically ordered), \( f \) is a vector representing the object (e.g., the voxel values of an object, again lexicographically ordered), and the operator \( H \) is the matrix representing the imaging system. The noise \( n \) associated with the imaging process is then also a vector, to be dimensionally correct.

The goal is to invert equation 5.1. The inversion is performed by optimization of an estimate \( \hat{f} \) of the unknown object. The measure (of an estimate) that is to be optimized is called the energy of the estimate. The energy is a scalar, and its form is designed to use the data as well as information known \textit{a priori} about the object. The energy values can be thought of as defining a structure in the object space that has its lowest point (minimum value) at the desired solution. To find the desired solution, the algorithm must find the solution that corresponds to this minimum.

The most prominent contribution to the energy is typically, though not necessarily, a least-squares fit of the transformed estimate to the data:

\[ E = || g - \hat{Hf} ||^2, \]  

(5.2)

where \( ||x||^2 \) represents the squared norm of the vector \( x \). A least-squares solution is any estimate \( \hat{f} \) that satisfies
However, because of the noise in the data, and because $H$ may be insensitive to some of the structure of $f$, a least-squares fit is often not sufficient to recover $f$. There may be many solutions giving the lowest energy (the problem is ill-posed), and the energy differences between estimates near the global minimum may be indistinguishable due to the noise in the data (the problem is ill-conditioned). Therefore, additional constraints, often representing *a priori* knowledge of the object, must be placed on the estimate to produce a more reliable solution. These additional constraints, most often employed to control noise in the estimate or force a unique solution, will generally be linear additions to the least-squares energy term; therefore, the complete energy functional, in most cases, will consist of two terms.

The minimization of such energy functionals requires the simultaneous optimization of many parameters. Finding solutions for this type of problem is difficult because there are far too many degrees of freedom to permit an exhaustive search for an optimum. Furthermore, in many cases (so-called NP-complete problems), no algorithms are known which will determine the exact optimum with significantly less work than exhaustive search. Instead, heuristic algorithms have been employed, and have proven effective in minimizing such energy functionals.

If the energy functional to be minimized is simply the best least-squares fit to the data (eq. 5.2), then a Monte Carlo search is one heuristic method that can be used to minimize this functional. In one particular version of this algorithm, the search is implemented as a series of perturbation trials. The initial estimate for a particular trial is altered by the addition or subtraction of a perturbation to one of its elements, thus forming a trial estimate. The energy of the trial estimate is calculated and compared to the
energy of the initial estimate. If the energy of the trial estimate is less than the energy of the initial estimate, the trial estimate becomes the initial estimate for the next trial; otherwise, the initial estimate remains the same for the next trial. This procedure is repeated many times for all elements in the estimate.

The logical flow of the Monte Carlo search algorithm can be outlined as shown in the following:

1) Propose a perturbation $\Delta \hat{f}$ to the initial estimate $\hat{f}$.
2) Calculate the change in energy $\Delta E(\Delta \hat{f})$ that would result from this perturbation.
3) Decide to either accept or reject the perturbation:
   a) if $\Delta E \leq 0$, then accept the perturbation unconditionally
   b) if $\Delta E > 0$, then reject the perturbation unconditionally
4) Update the initial estimate for the next trial to reflect the decision made in step 3.

These four steps are required for each perturbation trial and constitute the inner loop of the Monte Carlo search algorithm.

The procedure is an effective means for finding the absolute minimum of the simple least-squares energy functional because it is quadratic in nature. Thus the absolute minimum is easily found because the structure mapped out by the energy values is a multidimensional parabola, and any perturbation that reduces the energy moves the estimate down a side of this parabola toward the bottom and hence the desired solution. There are, of course, more efficient means (such as conjugate gradient or steepest descent routines) for finding the minimum of such a parabolic structure. However, if the structure defined by the values of the energy functional is not purely parabolic, but instead has
multiple minima, then both the gradient search and the Monte Carlo search may be ineffective in finding the absolute, or global, minimum. A method that has proven effective in finding the global minimum of a structure containing several minima is the simulated annealing algorithm.

Simulated annealing has been applied to varied problems exhibiting both local and global minima. These problems range from cell-placement in integrated-circuit design (Darema F et al., 1987) to that of solving the travelling salesman problem (Aarts EHL et al., 1986).

**Simulated Annealing**

The simulated annealing algorithm was originally introduced by Metropolis et al. (Metropolis N et al., 1953) for applications in statistical mechanics. 1983 saw the simulated annealing algorithm applied to the areas of integrated circuit design (Kirkpatrick S et al., 1983) and the reconstruction of nuclear medicine images (Smith WE et al., 1983) Since that time, it has come to be widely recognized as an effective approach to solving large combinatorial optimization problems or to finding global extrema of complicated multidimensional structures.

As in the Monte Carlo algorithm described above, simulated annealing starts with an initial estimate for each perturbation trial and generates a new estimate by randomly perturbing the initial estimate in some manner. The energy of the trial estimate is compared to the energy of the initial estimate, and if the energy of the trial estimate is less than energy of the initial estimate, the trial estimate is accepted as the initial estimate for the next trial, just as in the Monte Carlo algorithm. If the energy of the trial estimate is greater than the energy of the initial estimate, however, the trial estimate is not necessarily rejected in simulated annealing. Rather, it is accepted as the current estimate with a probability given by $e^{-\frac{\Delta E}{kT}}$, where $\Delta E$ is the difference between the energies of
the initial and trial estimates, $T$ is a control parameter referred to as the temperature, and $k$ is a constant.

The algorithm begins with an initial temperature high enough so that the probability of accepting perturbations resulting in an increase in energy is very close to 1. The temperature is then slowly decreased toward zero in accordance with some cooling schedule, hence, the analogy with the physical process of annealing.

With certain restrictions on the way in which the initial estimate is perturbed to generate a trial estimate, the simulated-annealing algorithm can be shown to converge asymptotically to the global minimum (Mitra D, 1985). Unfortunately, any real implementation of the algorithm is only an approximation, because the asymptotic behavior of the parameters that govern the convergence of the algorithm can only be approximated. If the annealing is slow enough, then only the estimates with a non-zero probability of occurring are the ones with minimum energy.

The logical flow of the simulated annealing algorithm is nearly identical to that of the Monte Carlo search (the Monte Carlo search is actually a special case of the simulated annealing algorithm with the temperature set to zero), differing only in the action taken when $\Delta E > 0$. The logical flow of the simulated annealing algorithm is shown below. The **boldface** type indicates the course of action defining the simulated annealing algorithm.

1) Propose a perturbation $\Delta \hat{r}$ to the initial estimate $\hat{r}$.
2) Calculate the change in energy $\Delta E(\Delta \hat{r})$ that would result from this perturbation.
3) Decide to either accept or reject the perturbation:
   a) if $\Delta E \leq 0$, then accept the perturbation unconditionally
   b) if $\Delta E > 0$, then accept the perturbation with a **Boltzmann probability**, $e^{(\Delta E/kT)}$. 
4) Update the initial estimate for the next trial to reflect the decision made in step 3

These four steps constitute the inner loop of the simulated-annealing algorithm.

The general problem of determining an optimal cooling schedule for the simulated-annealing algorithm, still a topic of considerable research, is not treated here. In this dissertation we explore variations from the standard form of simulated annealing (i.e., a serial implementation) that allow the algorithm to be implemented in parallel on TRIMM. Convergence (to the best estimate of the object) of simulated annealing can be proved in the serial implementation, but the variations considered here will invalidate the usual proofs. Thus, the visual quality of the reconstruction, and the efficiency of the calculations resulting from a parallel implementation of the simulated annealing algorithm must both be considered as measures of success.

**Previous Work on Parallel Simulated Annealing**

Parallelization has been introduced into the simulated annealing algorithm at three distinct levels: the decision level, the estimate level, and the Markov Chain level. The decision level is the lowest level of parallelization and is known as functional decomposition of a perturbation. The first three steps of the inner loop (as described above) must be executed sequentially because it is not possible to evaluate the cost (energy change) of a perturbation without having chosen it, nor is it possible to apply the acceptance criterion without knowing the cost of the perturbation. An attempt can be made to implement this functional decomposition of a perturbation on an even finer scale by breaking each step into smaller tasks, some of which can be carried out in parallel (Kravitz SA and Rutenbar A, 1986). The scheduling of the tasks must be synchronized carefully, and hence the parallelism suffers. Experimental results show that only a little
speedup can be achieved with this technique. The reported speedup factor is less than 2 using three processors, and was projected to increase only slightly with the addition of more processors.

The highest level of parallelization is the Markov Chain level (a Markov Chain is a sequence of events, or links, in which each event only depends on the immediately preceding link or event in the chain). The starting point for the algorithm is the theoretical model for the serial implementation of the simulated annealing algorithm, which can be described as a sequence of stationary Markov Chains, one for each temperature step. A decomposition of this serial implementation can be done at the Markov Chain level (Aarts EHL et al., 1986). At each temperature, the Markov Chain is divided into subchains, each of which can be generated by a different processor. Thus, the algorithm achieves parallelism by having each processor work on its own copy of the estimate, carefully selecting the initial conditions for each stage of the cooling schedule. At a specified time in the calculation, the results of the subchains are compared and the one with the lowest energy is accepted as the starting point for the next subchain. Experimental results report computation times six times smaller than for the sequential algorithm using eight processors. However, the efficiency in processor utilization drops as the number of processors increases, and extrapolation of the results obtained by the authors (Aarts EHL et al., 1986) shows that no gain can be obtained beyond about 30 processors.

An intermediate level of parallelization can be performed at the estimate level and is the area that has received the most attention (Darema F et al., 1987 and Cassotto AF and Sangiovanni-Vincentelli, 1987). This approach is distinguished by the characteristic that only one copy of the estimate is shared among all the processors. The algorithm variations discussed in this dissertation belong to this class.
The primary consideration at this level of parallelization is how to get multiple processors to act cooperatively. This cooperation manifests itself in the means used to keep each processor as current as possible with the ever-changing global estimate. Two factors that must be given consideration here are how the reconstruction space is divided (domain decomposition) among the processors, and how the communications, or update, scheme for the current estimate is implemented. For, as the current estimate upon which the acceptance/rejection criteria is based becomes less accurate, more erroneous decisions are made, and the estimate takes on a chaotic nature.

The reconstruction space can be divided either dynamically or statically. As is generally the case, the choice is dictated somewhat by the hardware configuration. Dynamic allocation (allocating elements of the object estimate) is much more amenable to shared-memory systems, and static allocation is more appropriate for distributed-memory systems.

In a shared-memory system, the estimate elements can be dynamically allocated as the processors become free. As a means to provide each processor with the best current global estimate, locks are used to restrict access to an estimate element that is involved in the current perturbation process (a lock is a flag that, when set, restricts access to a specific element). This method can be carried a step further by locking not only the element involved in the perturbation process, but also any additional elements involved in the application of an additional constraint on the estimate. For example, suppose there is an additional constraint in the energy functional that involves elements of the estimate that are contiguous (nearest neighbors) to the element being perturbed. The nearest-neighbor elements may then also be locked to ensure that they are not altered before a decision is made on the element in question. Although resulting in more locking overhead, the more elements that are can be locked, less chaos is introduced into the estimate (Darema F et al., 1987).
Static allocation on a distributed-memory system has been implemented by Cassotto et. al (Cassotto AF and Sangiovanni-Vincentelli, 1987). Their implementation involves distributing the estimate elements, in a predetermined fashion, among the processors. The updating of the global estimate by each processor, as well as any other information that is shared between processors, is communicated via the interprocessor communications network. Variations on this scheme involve changing the frequency of the updates to each processor, or changing what information is shared. While the former can be experimented with, the latter is generally dictated by the formulation of the problem. This implementation allows for more chaos to be introduced into the system because the price of dynamic distribution and an associated "locking" mechanism analogous to that used in the shared memory system is too high.

One particular implementation, allows essentially no chaos to be introduced into the estimate independently of the allocation scheme used (Kravitz SA and Rutenbar RA, 1986). It uses the concept of "serializable subsets of perturbations". A subset $P$ of $k$ perturbations $P = \{p_1, p_2, \ldots, p_k\}$ is serializable if none of the perturbations in $P$ interact with each other. This means that the decision of accepting or rejecting a particular perturbation $p_i$ does not depend on the order in which the perturbations in $P$ are processed; only serializable perturbations are allowed. Because the determination of large subsets of serializable perturbations is extremely difficult, the algorithm restricts its attention to the "simplest" serializable subsets, i.e., subsets in which all the perturbations are rejected, with the possible exception of one. Such serializable subsets of perturbations are obtained by attempting many perturbations in parallel, and then accepting into the global estimate only the perturbation that is accepted first, aborting all the others. A disadvantage with this scheme is that it seems to be biased in favor of those simple perturbations whose cost can be evaluated faster than the others. This approach shows a
near linear speedup at very low temperatures, when very few perturbations are accepted, but its behavior is poor at high temperatures when many perturbations are accepted.

**Implementing the Algorithm**

The first task is the determination of the change in energy, $\Delta E$, between successive estimates resulting from a perturbation $\Delta \hat{f}$. Recall the functional form of the least-squares data-agreement expression (eq. 5.1),

$$E = \| g - H\hat{f} \|^2.$$  

Equation 5.4 can be expanded to give

$$E = \| g - H\hat{f} \|^2 = \| g \|^2 - 2(g, H\hat{f}) + \| H\hat{f} \|^2.$$  

where $(a,b)$ denotes the inner product between the vectors $a$ and $b$. The change in energy resulting from a finite perturbation $\Delta \hat{f}$ is then

$$\Delta E(\Delta \hat{f}) = 2(H\hat{f}, H \Delta \hat{f}) - 2(g, H \Delta \hat{f}) + (H\Delta \hat{f}, H\Delta \hat{f}).$$  

Equation 5.6 can be rewritten in several forms through the use of two simple rules of linear algebra:

$$(a,b) = (b,a)$$  

and

$$(a,Hb) = (H^\dagger a,b)$$
where $H^\dagger$ is the Hermitian adjoint of $H$. Because the elements of $H$ are real, the adjoint of $H$ is equivalent to its transpose, $H^\dagger = H^T$. One such form of equation 5.6 is

$$\Delta E(\Delta \hat{f}) = 2(H^T H \hat{f}, \Delta \hat{f}) - 2(H^T H \hat{f}, \Delta \hat{f}) + (H^T H \Delta \hat{f}, \Delta \hat{f}).$$  

(5.9)

Both $\Delta E$ formulations (eqs. 5.6 and 5.9) consist of three terms: the first term represents the inner product between a representation of the current estimate $\hat{f}$ and the current perturbation vector $\Delta \hat{f}$; the second term represents an inner product between a representation of the data vector $g$ and the current perturbation vector; and the third term represents the inner product of the current perturbation vector with itself.

**Estimate Constraints**

As mentioned previously, due to the underdetermined nature of the problem, it is often necessary to place additional constraints, representing the a priori knowledge of the object, on the estimate in order to produce a more reliable solution. Constraints often used in reconstructions of nuclear medicine images are positivity and smoothing.

If it is known that the elements of the object to be reconstructed are never negative, then a positivity constraint should be enforced. A positivity constraint would reject a perturbation to an object element that would cause that element to be negative.

If it is known that an object is smooth, then a "smoothing" term can be incorporated into the energy functional. This term is designed to bias the reconstruction in favor of agreement between nearest neighbor elements of the estimate. The smoothing term can be expressed as

$$E_{\text{smoothing}} = \alpha \sum_{i=1}^{N} (f_i - \bar{f})^2,$$

(5.10)
where $\bar{f}$ is the average of the $x$ nearest neighbors of the $i^{th}$ element of the estimate and $\alpha$ is a constant weighting factor. This constraint is a linear addition to the least-squares energy term, and the entire energy functional is then

$$E = \| g - Hf \|^2 + \alpha \sum_{i=1}^{N} (f_i - \bar{f})^2.$$  \hspace{1cm} (5.11)

If, in addition, the size of the perturbation is restricted to the assumed level of activity in the bolus, the result is a strict binary constraint on the estimate.

The smoothing term in equation 5.10 may be expanded to give

$$\Delta E_{\text{smoothing}} = \sum_{i=1}^{N} (2(f_i - \bar{f}) + \Delta f_i) \Delta f_i$$ \hspace{1cm} (5.12)

If only a single element of the estimate is perturbed, equation 5.12 becomes

$$\Delta E_{\text{smoothing}} = (2(f_1 - \bar{f}) + \Delta f_1) \Delta f_1$$ \hspace{1cm} (5.13)

Regardless of the $\Delta E$ formulation used, the change in energy resulting from the smoothing term is simply added to the change in energy due to the data-agreement term, so that

$$\Delta E = \Delta E_{\text{data-agreement}} + \Delta E_{\text{smoothing}}.$$  

It should be noted that an energy functional consisting of the sum of equations 5.4 and 5.10 produces an energy surface that is purely quadratic in nature and therefore has no local minima. Since no local minima exist, there is no need for annealing and the simulated annealing algorithm reduces to a Monte Carlo search. There are, however, other energy functionals that do produce local minima and therefore require an algorithm such as simulated annealing for solution. For example, a Markov random field (MRF) regularization term will produce local minima. Not within the scope of this dissertation, this particular regularization term is discussed at length by Gooley (Gooley TA, 1990) and Jennison and Jubb (Jennison C and Jubb C, 1987).
Data-space and Object-space Implementations

Although mathematically equivalent, equations 5.6 and 5.9 differ significantly in their effects on the implementation of the algorithm. The two formulations may be characterized by the way in which the effect of the perturbation appears in the calculation.

The representation of the perturbation vector determines the "space" in which the $\Delta E$ calculation is performed, the space being either the object or data space. Equation 5.6 represents a calculation done in the data space and can be rewritten as

$$\Delta E(\Delta \hat{g}) = 2(g, \Delta \hat{g}) - 2(g, \Delta \hat{g}) + (\Delta \hat{g}, \Delta \hat{g}), \quad (5.14)$$

through the transformation

$$\Delta \hat{g} = H\Delta \hat{f} \quad (5.15)$$

and

$$\hat{g} = H\hat{f}. \quad (5.16)$$

Equations 5.15 and 5.16 are merely representations of the perturbation vector and object-estimate vector, respectively, in data space.

In contrast, equation 5.9 represents a $\Delta E$ calculation being performed in object space where the perturbation vector is represented simply as $\Delta \hat{f}$. In the first term of equation 5.9, the current estimate is projected from object space through the system into data space ($H\hat{f} = \hat{g}$) and then backprojected into the object space. Backprojection is the operation that projects the data from data space into object space. The backprojection operator can be formed from the projection operator $H$ by simply transposing the elements of $H$. The data vector, $g$, in the second term, backprojected into object space,
takes the form of $H^T \hat{\theta} (= H^T H \hat{f})$. Likewise for the perturbation vector in the third term, $\Delta \hat{f}$ is projected into the data space, rendering $H \Delta \hat{f}$, and then backprojected into the object space, resulting in $H^T H \Delta \hat{f}$.

**Domain Decomposition and Space Dependence**

Although the space in which the $\Delta E$ calculation does not influence the way in which the reconstruction space is decomposed into subdomains among the processors, it does dictate what information is requisite for each processor to conduct its perturbation-trial process for its respective subdomain of the estimate. For purposes of discussion, each "subdomain" of the estimate, for which a specific processor is responsible for reconstructing, will henceforth be referred to as that portion of the estimate "local" to that processor. The term local, where appropriate, will also be used to refer to the information used by a specific processor to conduct its perturbation-trial process on its local estimate. In contrast, the term "global" refers to the entire domain (estimate, data vector, transformation matrix, etc.).

The differences in implementing the data-space and object-space formulations in parallel are greatly influenced by the dimensions of the vectors involved in the problem. The dimensions of the problem determine how much memory is required to store the information required for the $\Delta E$ calculation, the number of calculations involved in determining $\Delta E$, and the information that must be shared among the processors to maintain the most current global estimate possible.

**Calculating $\Delta E$ in data space**

As mentioned previously, the reconstruction problem is generally underdetermined. The $H$ matrix is a nonsquare matrix of dimensions $M \times N$, with $M$
typically 1/4 to 1/8 of N. Lexicographically ordering the vectors results in a \( \mathbf{g} \) vector of dimension \( M \times 1 \) and an \( \hat{\mathbf{f}} \) vector of dimension \( N \times 1 \). The greater the number of elements in \( \hat{\mathbf{f}} \), the greater the potential to observe fine detail in the reconstructed object. If, for example, the reconstruction space is a 55 x 55 grid (a fairly coarse resolution), then the dimensions of the \( \hat{\mathbf{f}} \) vector are 3025 x 1. If \( M \) (the number of detector elements) is approximately 1/4 of \( N \), then the dimensions of the \( \mathbf{g} \) vector are approximately 756x1. The \( \mathbf{H} \) matrix therefore consists of more than two million elements (\( M \times N \)). If these elements are stored as single-precision floating-point numbers or long integers (both require 32bits/element), then over 9 megabytes are required to store the entire \( \mathbf{H} \) matrix for even this small problem. Fortunately, the \( \mathbf{H} \) matrix is generally sparse -- most of the elements are zero. The sparseness of the \( \mathbf{H} \) matrix is a result of using a pinhole system and is due to the fact that not every object voxel will be provided with a direct line of sight to every detector element. In order to take advantage of the sparseness of \( \mathbf{H} \), however, sparse- matrix methods must be used for computations involving the elements of \( \mathbf{H} \). Sparse matrix methods require storage of only the nonzero elements of the matrix, but they also require storage of the location of each nonzero element's position (some form of the row number and column number) within the matrix. Also, the time required to access elements of the matrix stored in sparse form is greater than if the matrix were stored in conventional fashion, because several steps are required to either retrieve an element or determine whether it is nonzero. Nevertheless, because of the great amount of memory required to store the entire \( \mathbf{H} \) matrix, sparse-matrix methods must be utilized.

To calculate \( \Delta \mathbf{E} \) in data space, the processor must be provided with the columns of \( \mathbf{H} \) that correspond to the elements of the object estimate which it is responsible for reconstructing. For example, if a processor's reconstruction space corresponds to the first 100 elements of the estimate, then the first 100 columns of \( \mathbf{H} \) must be stored locally to this processor. This division of \( \mathbf{H} \) and \( \hat{\mathbf{f}}_{\text{global}} \) is depicted in figure 5.1. In addition, the
processor must be provided with a copy of the entire \( \mathbf{g} \) vector. The first step in the calculation involves calculating \( \Delta \hat{\mathbf{g}} (= \mathbf{H} \Delta \hat{\mathbf{f}}) \). Next, if only the effect of a single perturbation is determined at a time (only one element of \( \Delta \hat{\mathbf{f}} \) is nonzero), for a perturbation \( \epsilon \) applied to the \( i \)th element,

\[
\Delta \hat{\mathbf{g}}_k = \epsilon \mathbf{H}_{ki}.
\]

(5.17)

This operation consists of multiplying the column of \( \mathbf{H} \) that corresponds to the element of the estimate being perturbed by the magnitude of the perturbation, \( \epsilon \); for example, if the \( i \)th element of the estimate is being perturbed, then the values of the resultant data vector are simply the values of the \( i \)th column \( \mathbf{H} \) scaled by the magnitude of the perturbation \( \epsilon \).

Once \( \hat{\mathbf{g}} \) and \( \Delta \hat{\mathbf{g}} \) have been determined, \( \Delta E(\Delta \hat{\mathbf{g}}_i) \) can be calculated (Note that the subscript \( i \) refers to the \( i \)th \( \Delta \hat{\mathbf{g}} \) vector -- the perturbation vector corresponding to the \( i \)th element of the estimate -- and not the \( i \)th element of the \( \Delta \hat{\mathbf{g}} \) vector itself.) The first two terms of equation 5.14 involve an inner product of \( M \) elements: the first term, between the current \( \hat{\mathbf{g}} \) and \( \Delta \hat{\mathbf{g}} \), and the second term, between \( \mathbf{g} \) and \( \Delta \hat{\mathbf{g}} \). The third term reduces to \( || \Delta \hat{\mathbf{g}} ||^2 \). Thus, determining the change in energy of the least-squares energy functional due to a single perturbation requires \( (3M) \) multiplications and additions.

If \( \Delta E(\Delta \hat{\mathbf{g}}_i) \) is less than zero, then the perturbation is accepted, and the current object estimate must be updated to reflect this. In addition, \( \Delta \hat{\mathbf{g}} \) must be added to the current \( \hat{\mathbf{g}} \) (requiring \( M \) additions), in order to remain as current as possible. To summarize, calculating the change in energy in data space requires a minimum of \( (3M) \) multiplications and \( (4M) \) additions, all of which may be floating point operations.

The global \( \hat{\mathbf{g}} \) vector consists of a sum of the local \( \hat{\mathbf{g}} \) vectors:
\[ \hat{\mathbf{g}}_{\text{global}} = \sum_{i=1}^{P} (\hat{\mathbf{g}}_{\text{local}})_i, \]

where \( P \) equals the number of processors. Therefore each local portion of \( \hat{\mathbf{g}} \) must be shared among the processors in order for the perturbation-trial decisions on proposed perturbations to be made as accurately as possible. Depending on the form of the noise-control or regularization term used in the energy functional, the processors may also need to be kept as current as possible with respect to the global estimate of the object as well.

**Calculating \( \Delta E \) in object space**

In contrast to the \( \Delta E \) formulation in data space where only \( H \) needs to be stored, the formulation in object space requires that each processor store the necessary elements of the \( H^T H \) matrix. Recall that \( H \) is an \( M \times N \) matrix; therefore, \( H^T H \) has dimensions \( N \times N \). For the example given above of a \( 55 \times 55 \) object space, \( H^T H \) has more than 9 million elements. Requiring four bytes per element, this matrix would require in excess of 36 megabytes to store the whole matrix. This storage demand is not necessarily alleviated by the fact that \( H \) is sparse, because that does not guarantee \( H^T H \) to be sparse. In fact, for the objects investigated in this dissertation, \( H^T H \) is nearly full.

To calculate \( \Delta E(\hat{f}) \), each processor must be provided with the rows of \( H^T H \) corresponding to the elements of \( \hat{f}_{\text{global}} \) that the particular processor is responsible for reconstructing. For example, if a processor's reconstruction space corresponds to the first 100 elements of the estimate, then the first 100 rows of \( H^T H \) must be stored locally to this processor. This division of \( H^T H \) and \( \hat{f}_{\text{global}} \) is depicted in figure 5.2. In addition, each processor must be provided with the elements of \( H^T g \) that correspond to its respective elements of the object estimate.
If the effect of only a single perturbation is evaluated at a time, the $\Delta E(\Delta \hat{f})$ calculation is straightforward. Only one element of the perturbation vector $\Delta \hat{f}$ is nonzero, and a perturbation of size $\varepsilon$ is applied to the $k^{th}$ element gives

$$\begin{align*}
(\Delta \hat{f})_k &= \varepsilon \delta_{kk}.
\end{align*}$$

For each object element that is perturbed, the corresponding element of $H^T H \hat{f}$ must be calculated. This calculation could result in as many as $N^2$ multiplications and additions, which may or may not be floating-point operations (depending on how $H^T H$ is stored). Once $(H^T H \hat{f})_k$ has been calculated, the first term of the formulation is simply a multiplication of $(H^T H \hat{f})_k$ by $\varepsilon$. Likewise, the second term is a single multiplication of $(H^T g)_k$ by $\varepsilon$, and the third term is the product of $(H^T H)_{kk}$ (diagonal element of the $k^{th}$ row of $H^T H$) and $\varepsilon^2$. Therefore, the $\Delta E$ calculation essentially requires the number of multiplications and additions dictated by the number of elements in the row of $H^T H$ that corresponds to the particular object element being perturbed. If the perturbation is accepted into the object estimate ($\Delta E(\Delta \hat{f}_i) < 0$), the estimate can be updated by simply adding the perturbation to the $i^{th}$ element of the object estimate. Unlike the data-space formulation, only the current local estimates ($\hat{f}_{\text{local}})_i$ of the object must be shared among the processors in order to keep the acceptance criterion as current as possible.

To summarize, the most significant differences between the two formulations are the amount of memory required to store the transformation matrix $H$ in the data-space formulation and $H^T H$ in the object-space formulation, and the number of calculations required to determine $\Delta E$. Slightly more information must be transferred in the data-space formulation, but if the communication network is sufficiently fast compared to the time
required to complete the inner loop, this additional communications time will be inconsequential.

Investigation of these two formulations of the data-agreement functionals, of various global update schemes, and of the implementation of additional constraints in the energy functional are discussed in the next chapter on experimental results.
Fig. 5.1. Estimate and transformation matrix for the data-space formulation of $\Delta E$. Each processor 1 through $N$ is provided with the columns of $H$ that correspond to the elements of $\hat{f}$ ($\hat{f}_{\text{local}}$) for which is responsible for reconstructing. The brackets within the matrix bars designate the respective rows of $H$ and the elements of $\hat{f}_{\text{local}}$ local to each processor. In addition each processor is provided with the entire $\hat{g}$ vector.
Fig. 5.2. Estimate and transformation matrix for the object-space formulation of $\Delta E$. Each processor 1 through N is provided with the rows of $H^T H$ that correspond to the elements of $\hat{f}$ ($\hat{f}_{local}$) for which is responsible for reconstructing. The brackets within the matrix bars designate the respective rows of $H^T H$ and the elements of $\hat{f}_{local}$ local to each processor. In addition each processor is provided with the entire $\hat{g}$ vector.
Recall that because of the quadratic nature of the energy functional used for reconstruction, the simulated annealing algorithm simplifies to a Monte Carlo search. The parallel implementations of this reconstruction algorithm are parallelized at the estimate level, with the smallest task being a perturbation trial (inner loop). Therefore, it is at this level that the dynamics of the reconstruction process must be analyzed. As discussed in chapter 5, the inner loop of the simulated annealing algorithm can be formulated in either object space or data space. The two $\Delta E$ formulations differ in terms of the number of calculations involved, the information that must be stored local to each processor, and the global information that must be shared among all the processors. The ideal situation, of course, would allow every element of the estimate to be reconstructed independently of all the other elements. However, due to the basic nature of the algorithm, this is not possible. Therefore, the emphasis of these investigations, as is often the case in parallel applications, is placed on the communications between the processors. It is the application of the $\Delta E$ formulations, the communications schemes used to share data among the processors, and the interplay between these two, that constitute each parallel implementation and provide the impetus for the research done in this dissertation.

Since each perturbation-trial decision made on a local level is based on the global estimate, the more current the global estimate that each processor is provided with, the more reliable the decision will be. Unfortunately, to keep each processor more up-to-date globally requires a greater degree of communications, and system performance suffers. The more independently the processors are allowed to function -- allowing more perturbations to be accepted simultaneously based on a less current global estimate -- the greater the degree of parallelism achieved. However, the more independent the operation,
the greater the risk of introducing chaos into the estimate. Therefore, in an effort to determine the most efficient means of communications, three schemes, each varying in degree, are investigated: the asynchronous case, in which the processors are allowed to make simultaneous acceptances throughout a reconstruction; the synchronous case, in which the processors are allowed to make simultaneous acceptances for a short period before being temporarily halted and synchronized to the same global estimate; and a hybrid case, in which the processors are allowed to perform the decision-making process simultaneously, as with the asynchronous case, but being synchronized to the same global estimate only when deemed necessary according to a synchronization criterion.

TRIMM (as discussed in chapter 4) was designed to provide various communication pathways between processors. Each transputer module's memory is accessible by the host processor (the memory local to each transputer is mapped into the memory of the 68020 controller), and the host can communicate with each transputer individually by simply reading from or writing to a specific memory location. Although only one local memory can be read at a time, all the local memories can be written to simultaneously through the use of the broadcast mode. These capabilities, particularly the broadcast mode, allow for quick and efficient distribution of information that must be shared between the processors. In addition, the ability to connect the transputers in various configurations through the use of their links provides an extremely fast serial pathway between immediate connections; in fact, if the amount of data to be transferred is relatively small, efficient transmission between nonconnected processors, via intermediate processors, can be realized as well.

The architecture used for investigating parallel implementations of the Monte Carlo search algorithm is a ring configuration as shown in figure 6.1. The ring architecture was chosen because the algorithm can be adapted to simulate the various modes of operation that will be provided when TRIMM is fully operational. As a
comparison between the actual performance of the ring and the anticipated performance of TRIMM utilizing the broadcast mode, timing predictions based on the communications scheme implemented and the number of processors involved are included.

**Global Communication Schemes**

The degree of synchronicity, or lack thereof, dictated by a communication scheme directly reflects how independently each processor is allowed to function in reconstructing its fraction of the object. Each update scheme is characterized by how a global update cycle is performed. A global update cycle consists of two phases and is complete when each processor has received the most current global estimate. In the first phase, each local portion of the estimate is collected, thus forming the current global estimate, and in the second phase this global estimate is distributed to all the processors. Just how this is update done, and what form it takes, depends on the architecture being utilized, the energy functional being implemented, and of course, the degree of synchronization required.

**Asynchronous Communication Scheme**

**Ring Implementation**

The asynchronous communication scheme allows the processors to operate the most independently by eliminating a strict synchronization step. Each processor receives the current global estimate, updates it with its current local estimate, and passes the updated global estimate to its succeeding processor. The processor then continues its perturbation-trial process utilizing the newly updated global estimate. This process continues with each successive processor circulating the current global estimate around the ring.
As discussed in chapter 5, different information is required for each formulation of the $\Delta E$ calculation. The object-space formulation requires only the global estimate of the object $\hat{f}_{\text{global}}$ while the data-space formulation requires $\hat{g}_{\text{global}}$. In addition, if there exists a term in the energy functional that cannot be expressed in data space, then $\hat{f}_{\text{global}}$ is also required by the data-space formulation.

Implementing the asynchronous scheme (see flowchart in fig. 6.2) in the object-space formulation on a ring architecture is straightforward. Each transputer is connected to two others, one serving as the input and the other as the output for that particular transputer. Once communications are initiated, each processor simply inputs the current global estimate, incorporates its local estimate, $\hat{f}_{\text{local}}$, into the global estimate, and passes this updated version of $\hat{f}_{\text{global}}$ to its neighboring (output) processor. Thus the global estimate continuously circulates around the ring, while being updated by each processor. Implementing the asynchronous scheme in this way results in minimal communications overhead within each processing element. Specifically, before inputting the global estimate, each processor must first save the changes made locally since the last communication, so that they are not overwritten and lost.

If the data-space $\Delta E$ formulation is used, slightly more overhead is involved, since the updating of $\hat{g}_{\text{global}}$ involves the sum in equation 5.18. Not only must each processor add $\hat{g}_{\text{local}}$ to the circulating global estimate, but each element of $\hat{g}_{\text{local}}$ must then be set to zero after passing on the updated $\hat{g}_{\text{global}}$. This operation is necessary to prevent the same changes being continually added to the global estimate. Also, as was done in the object-space formulation, $\hat{f}_{\text{global}}$, must also be circulated.

TRIMM

Two options are available for implementing this asynchronous scheme on TRIMM if the host processor is utilized. In both variations, the host simply acts as a
system manager, collecting the local portions of the global estimate and distributing them among all the processors. Just how this distribution is accomplished delineates one method from the other.

**Broadcast Mode Disabled**

The first implementation, although incorporating the host processor, involves a communication scheme that, in principle, is identical to that of the ring architecture just discussed. The host reads the most recent changes made to a respective processor's portion of the estimate, updates the global estimate with these changes, and writes the updated global estimate back into the processor's memory. This process is repeated for each processor in the array. In the object-space formulation, this method eliminates the communications overhead of storing $\hat{f}_{\text{local}}$ before inputting the global estimate.

The data-space formulation is more efficient as well, because the $\hat{g}_{\text{global}}$ sum is performed by the host (after collecting $\hat{g}_{\text{local}}$ from each transputer) and not locally by each transputer.

**Broadcast Mode Enabled**

The second implementation using the host processor utilizes the broadcast capabilities of TRIMM. The host collects the changes made locally to the estimate from each processor and updates the entire global estimate before distributing the new global estimate simultaneously to all the processors. For the object-space formulation, as with the stand-alone ring architecture, each processor must save the changes made locally since the last read of its estimate by the host, and then restore those changes into its copy of $\hat{f}_{\text{global}}$ before resuming its decision-making process.

The operation for the data-space formulation is similar to its implementation on the stand-alone ring architecture except that the addition of $\hat{g}_{\text{local}}$ to $\hat{g}_{\text{global}}$ is performed
by the host instead of locally. $\hat{g}_{\text{local}}$ must be zeroed after receiving the updated global estimate to avoid duplicating the effect of previously made changes.

**Synchronous Communication Scheme**

The synchronous communications scheme (see flowchart in fig. 6.3) was designed to provide a more stable path of convergence, at the sacrifice of speed, than may result from the asynchronous scheme. In fact, it was this concern that provided the impetus to implement the broadcast addressing mode. This scheme, in contrast to the asynchronous implementation, consists of two distinct phases, the collection phase and the distribution phase. Thus, the distribution of the global estimate is not done on the fly, but instead distribution occurs only after the entire global estimate has been collected and updated. The processors are halted from simultaneous operation and synchronized to the same global estimate before resuming the perturbation-trial process.

**Ring Implementation**

This scheme is a bit more cumbersome to implement than was the asynchronous scheme. Similar in operation to the asynchronous case, a global estimate is continually circulated around the ring. Once the collection phase is initiated, each processor inputs the circulating global estimate, updates the global estimate by incorporating its local contribution, and passes the updated global estimate to the next processor in the ring. However, instead of immediately incorporating the global estimate into its perturbation-trial process and continuing (as in the asynchronous case), the processor halts operation until a new, complete global estimate is formed and ready for distribution. After every processor has been read, and the global estimate formed, the global estimate is passed
around the ring with each processor taking its own copy. The perturbation-trial process then begins again and continues until the next collection phase.

In the object-space formulation, only $\hat{f}_{\text{global}}$ must be circulated. During the collection phase, each processor inputs $\hat{f}_{\text{global}}$ and substitutes its current $\hat{f}_{\text{local}}$ into the global estimate. The processor then passes the updated $\hat{f}_{\text{global}}$ on to the next processor and halts operation. Once $\hat{f}_{\text{global}}$ has been completely updated, it is passed around the ring so that each processor can obtain a copy and resume its perturbation-trial process with the updated $\hat{f}_{\text{global}}$ as the initial estimate.

Similarly, in the data-space formulation, $\hat{g}_{\text{global}}$ is circulated around the ring. The global estimate is formed by adding each processor's $\hat{g}_{\text{local}}$ to $\hat{g}_{\text{global}}$. Once a processor adds its $\hat{g}_{\text{local}}$ to the circulating $\hat{g}_{\text{global}}$, it passes the updated global estimate to the next processor and halts operation until the entire global estimate is formed. Once $\hat{g}_{\text{global}}$ is complete, it is passed around the ring with each processor taking a copy and resuming its perturbation-trial process using the updated $\hat{g}_{\text{global}}$ as the initial estimate.

**TRIMM: Broadcast Mode Enabled**

As mentioned previously, the TRIMM broadcast addressing mode was designed with the synchronous communications scheme in mind. To implement this scheme using broadcast addressing, the host processor simply reads each processor's local estimate (after which the processor halts), forms the entire global estimate, and then broadcasts this current global estimate to all the processors.

**Hybrid Communication Scheme**

As might be inferred, the hybrid communication scheme is a hybrid of the asynchronous and synchronous communication schemes. This communication scheme allows the processors to function asynchronously, being synchronized only when
necessary, in accordance with a synchronization criterion. By operating in this fashion, the algorithm proceeds with the speed of the asynchronous scheme, but limits the chaotic behavior in the estimate by intermittently synchronizing the processors to the same global estimate. The synchronization criterion used is simply one in which the processors are synchronized after a fixed number of passes through the reconstruction space. Each form of the \( \Delta E \) calculation is implemented as described in the sections on the asynchronous and synchronous communications schemes in each stage of the hybrid scheme.

To summarize, these three communications schemes offer varying degrees of synchronous operation. The asynchronous scheme, at the risk of not converging, provides no distinct synchronization mechanism, relying instead on the absolute speed of the links to maintain adequate "currency" of the estimate. The synchronous scheme, at the sacrifice of speed, provides a strict synchronization at the perturbation-trial level. Finally, the hybrid scheme provides an intermediate level of synchronization, by synchronizing the processors in accordance with the chosen synchronization criteria.

These three communication schemes will be implemented in reconstructions of objects of several types and sizes and compared with each other using the speedup and convergence of the reconstruction as figures of merit.

\[ \Delta E(\Delta \hat{f}) \text{ vs. } \Delta E(\Delta \hat{g}) \]

Recall the data-space \( \Delta E \) formulation,

\[ \Delta E(\Delta \hat{f}) = 2(H \hat{f}, H \Delta \hat{f}) - 2(g, H \Delta \hat{f}) + (H \Delta \hat{f}, H \Delta \hat{f}), \quad (6.1) \]

and the object-space \( \Delta E \) formulation,

\[ \Delta E(\Delta \hat{f}) = 2(H^T \hat{f}, \Delta \hat{f}) - 2(H^T \hat{f}, \Delta \hat{f}) + (H^T \hat{f}, \Delta \hat{f}). \quad (6.2) \]

Although mathematically equivalent, these two formulations differ in their parallel performance due to the dynamics introduced as a result of the parallelization of the
algorithm. Specifically, the length of time required to complete a perturbation trial, the communications scheme implemented, and the resulting pattern of simultaneous acceptances into each processor's local estimate all affect performance. For example, the time a specific processor spends in calculation is the time required for the remaining processors to update and communicate the circulating global estimate. Since the framework common to each communication scheme is such that each processor checks for an input communication only after a complete perturbation trial (see the flowcharts in figs. 6.2 and 6.3), regardless of the outcome, the communication time between adjacent processors is quantized to the time to complete the inner loop once. The calculation time provided a processor therefore fluctuates as a function of the number of processors and the time to process the inner loop. As a result, some processors may attempt more perturbations on its local space than others, thus biasing the final reconstruction. It is therefore important to decompose the reconstruction space among the processors as evenly as possible so not to aid in this bias. Because of these dynamics, even given the identical architectures, data, and communications scheme, it is probable that the perturbations accepted in one $\Delta E$ formulation will not be the same as those perturbations accepted in the other. In an effort to investigate the performance of the $\Delta E$ formulations, each formulation was implemented on the same architecture and used to reconstruct the same object. Each reconstruction begins with the same initial estimate and utilizes the same communications scheme. Table 6.1 contains tabulated information detailing specific architectures, communication schemes, and objects that are discussed throughout the remainder of this dissertation.

**Binary Myocardium**

The object to be reconstructed is a binary cross section of the myocardium (fig. 6.4). The term binary means that there are only two levels of quantization within the
object. This characteristic reflects two distinct levels of radioactivity physically present in the myocardium. The configuration of the imaging system used to generate the data is shown in figure 6.5. The system is comprised of $M = 128$ detectors configured to provide two orthogonal views (64 detectors per side). The aperture placed in front of each detector array consists of 9 pinholes, each 1mm in diameter, placed random distances apart. The object was digitized on a 39x39 grid ($N = 1521$). The resultant problem, with $N = 1521$ and $M = 128$, is severely underdetermined by a ratio of nearly 12 to 1.

One method of finding a solution to a problem that is so severely underdetermined is to incorporate a term into the energy functional that utilizes information known \textit{a priori} about the object. For this particular problem, it is known that the object is binary; therefore, a term must be added to the energy functional to reflect this. Recall, the smoothing term of eq. 5.10.

$$E_{\text{smoothing}} = \alpha \sum_{i=1}^{N} (f_i - \bar{f})^2$$

(6.3)

If, in addition to the smoothing term, the size of the perturbation is restricted to the known level of activity in the object and the value of any element of the estimate is not allowed to exceed that level, the result is a strict binary constraint.

A strict binary constraint can result in a scenario in which local minima do exist. Because of the restriction on the perturbation size, only limited moves down the energy surface are allowed. A contour of lower energy may never be reached because that move is not allowed due to the restriction placed on the perturbation size. This instance can be envisioned by considering a particular position on the energy surface shaped like a square with the current estimate residing at one corner, a lower energy level at the opposite corner, and the only allowed moves, because of the binary constraint, being along either
side of the square. Therefore, the lower energy level (the opposite corner of the square) cannot be reached and the estimate is trapped in a local minima. However, this scenario may not arise in a parallel implementation because of the allowed simultaneous acceptances. The allowed simultaneous acceptances actually act as variable temperature thus kicking the estimate out of the local minima in which it was entrapped.

The $H$ matrix used for the binary myocardium reconstruction is approximately 83% sparse, so that sparse-matrix methods are applicable. On the other hand, $H^T H$ is approximately 98% full (98% of the elements of $H^T H$ are nonzero) and is more efficiently stored as the full matrix. The computer code (written in the C programming language) for the inner loop of the object-space and data-space $\Delta E$ formulations is shown in figures 6.6a and 6.6b, respectively. The inner loop of the object-space formulation of $\Delta E$ requires $N$ multiplications and additions to calculate the element of the $H^T H \hat{f}$ vector that corresponds to the element of the estimate being perturbed. However, it is not absolutely necessary that each element of the $H^T H \hat{f}$ vector be calculated on a per-element basis and performed within the inner loop. Each processor could calculate its entire local portion of the $H^T H \hat{f}$ vector outside of the inner loop and use each element only as needed. This approach, however, would be inefficient due the communications schemes used and the desire to keep the estimate as current as possible. Including the $H^T H \hat{f}$ calculation inside the inner loop amounts to updating the estimate on a local basis after every perturbation acceptance.

The inner loop of the data-space $\Delta E$ formulation is more involved to calculate, but is more efficient due to the dimensions of the problem. Since the calculations are done in data space, the necessary vectors are of dimension $M$ instead of dimension $N$ as in the object-space formulation. The perturbation $\Delta \hat{f}$ appears in the vector $H \Delta \hat{f}$ which is unique for each perturbation trial. As discussed in chapter 5, $H \Delta \hat{f}$ is formed by simply multiplying each element of the column of $H$ corresponding to the perturbed element of $\hat{f}$
by the perturbation $\varepsilon$. This requires $M$ multiplications. Once $H\tilde{\Delta}f$ is determined, the calculation of each term in the $\Delta E$ formulation consists of $M$ multiplications and additions (see Table 6.2).

**Speed**

In the present problem, where $M = 128$ and $N = 1521$, the object-space formulation requires approximately 66% more calculations (see Table 6.2) than the data-space formulation. The resulting time to perform the inner loop (measured using the time() subroutine provided in the C compiler) for the object-space formulation (fig. 6.6a) and data-space formulation (fig. 6.b), is 79µsec and 200µsec, respectively. The data-space formulation is more efficient in terms of both calculation time and memory storage; however, because of the dynamics associated with each implementation, the question of convergence must be addressed.

The results of each implementation of the $\Delta E$ formulation are only equivalent to each other in the serial case where the initial estimate for each successive perturbation trial is equivalent. However, in the parallel case (excluding the case of serializable subsets), due to the allowance of simultaneous acceptances in the processors, the results of each $\Delta E$ formulation are not equivalent. Since simultaneous acceptances are allowed, the "current" estimate upon which each processor bases its perturbation-trial decision is different for each $\Delta E$ formulation and continually changing.

**Hamming Distance**

The convergence of the estimate can be estimated by calculating the energy of the estimate as it evolves throughout the reconstruction. However, due to the binary nature of the object, and the fact that the data contain no noise, a more precise measure of
reconstruction accuracy can be used. A figure of merit analogous to the Hamming distance in information theory can also be implemented. The value of each element in a binary estimate can be thought of as either being high or low in analogy with the bits in a digital signal transmission. The number of bits in the received signal that differ from the transmitted signal define the Hamming distance. This concept can be applied to the binary reconstruction problem, where the reconstructed object can be interpreted as the received signal and the original object as the transmitted signal.

The initial estimate in each case was a uniform level equal to zero, and the asynchronous update scheme was used for global communications. The reconstruction was performed on a ring architecture consisting of six transputer modules. The corresponding reconstruction-space decomposition among the processors is depicted in figure 6.7, with each gray level representing the elements local to each processor. The reconstructions of the binary myocardium cross-section corresponding to the data-space and object-space formulation can be seen in figures 6.8 and 6.9, respectively. The resulting Hamming distance for each reconstruction indicates that there is not a large deviation between the two formulations. Therefore, considering the significantly smaller amount of memory required for storage of the $H$ matrix versus the $H^T H$ matrix, the faster calculation time for the inner-loop, and the comparable reconstruction results, the data-space formulation was deemed the more efficient of the two (see Table 6.4). The data-space $\Delta E$ formulation was therefore used to investigate the various aspects of the communications schemes and reconstruction-space decomposition discussed throughout the remainder of this dissertation.
Implementation: Reconstruction Space Decomposition and Global Communications

Two objects of varying complexity and size were reconstructed in order to study the effects of the reconstruction-space decomposition and global communications scheme: a thyroid phantom (fig. 6.10) and the Hoffman brain phantom (fig. 6.11). Unlike the noise-free data used to reconstruct the myocardium cross-section, the data for these reconstructions is actual measured data, collected from a system that physically exists.

Imaging System

The system used to image these objects is comprised of 16 modular cameras arranged in an octagon as shown in figure 6.12. An array of pinholes was chosen as the coded aperture. As Roney demonstrated (1989), because of the symmetry of the system, only 1/4 of the imaging hardware actually had to be constructed. Four cameras, representing two sides of the octagon, were physically used to collect the image data for each object. This data was in turn used to emulate the characteristics of the complete 8-sided configuration (since only 1/4 of the detectors were physically used, the remaining 3/4 of the $H$ matrix was generated in software utilizing the collected data and symmetry constraints (Roney TJ, 1989). The reconstruction space for the thyroid phantom was digitized on a 55x55 grid while the brain phantom was digitized on a 71x71 grid. The number of detector elements used was 1024 and 2048 for the thyroid phantom and the brain phantom, respectively.

Theoretical Speedup

The primary goal of implementing the simulated annealing algorithm in parallel is to speed up the reconstruction process. If the convergence of the estimate is the same in both the parallel and serial implementations, then the speedup is simply the ratio of the
time required to process all the elements of the estimate the same number of times. The
time to process all the elements of the estimate in parallel, as a function of global-update
scheme and the number of processors, is given by

\[ T_{\text{par}} = \left( \frac{n}{N} \right) t_{\text{il}} + t_{\text{comm\_ovhd}} + t_{\text{sync}} \]

where

- \( n \) = number of elements in estimate
- \( N \) = number of processors
- \( t_{\text{il}} \) = time to process inner loop
- \( t_{\text{comm\_ovhd}} \) = communications overhead time
- \( t_{\text{sync}} \) = processor idle time during synchronization.

The term \( t_{\text{comm\_ovhd}} \) reflects the communications overhead associated
with the global-update scheme used. This factor includes the time required for
each processor to input the global estimate, \( t_{\text{comm\_l/p}} \), to update it, \( t_{\text{update}} \), and to
transmit it, \( t_{\text{comm\_o/p}} \); therefore,

\[ t_{\text{comm\_ovhd}} = t_{\text{comm\_l/p}} + t_{\text{update}} + t_{\text{comm\_o/p}}. \]

The term \( t_{\text{sync}} \) reflects the time that the processor is idle during
synchronization. For the asynchronous case, \( t_{\text{sync}} \) is zero and the only time the
processor spends idle is \( t_{\text{comm\_ovhd}} \). However, for the synchronous case, \( t_{\text{sync}} \) is
the time required for each processor to update the circulating global estimate
with its most current local estimate \( t_{\text{comm\_ovhd}} \) and, upon the global estimate
being fully updated, the time (again \( t_{\text{comm\_ovhd}} \)) required to circulate the new,
current global estimate. Consequently, for the synchronous case,

\[ t_{\text{sync}} = 2N t_{\text{comm\_l/p}} + t_{\text{update}}. \]
The calculated speedup for the data-space $\Delta E$ formulation implemented on the ring architecture utilizing the asynchronous and synchronous update schemes is pictured in the graphs in figures 6.13a and 6.13b, respectively. The same calculations for TRIMM utilizing the broadcast mode are shown in the graphs in figures 6.14a and 6.14b, respectively. For both architectures, the synchronous scheme is approximately half as fast as the asynchronous scheme since the time required to complete each of the two phases of the synchronous cycle is approximately equal to the time to complete one full asynchronous cycle. A testament to the speed of the transputer links is that the speedup for the ring, in both cases, is comparable to that of TRIMM in broadcast mode. However, as the calculation load decreases and the communication load increases, TRIMM, with the broadcast mode enabled, should out perform the ring due to saturation of the transputer's communications bandwidth.

In the case of the brain phantom, the calculation load is approximately the same as that in the case of the thyroid phantom (the number of nonzero elements of the $H$ matrix in each case is nearly equal) while the communication load increases by approximately 60% (due to the larger $\hat{f}$). The speedup for the ring configuration versus TRIMM in broadcast mode is shown in figure 6.15(a) for the synchronous update scheme. In this case, it can be seen that TRIMM with broadcast appears to exhibit better performance than the ring. This hypothesis can be carried a step further by considering the coded-aperture brain imager proposed by Barrett and Rowe. The system is comprised of 24,000 detectors and the reconstruction space digitized to 80,000 voxels, resulting in an $H$ matrix that is approximately 2% full. The relative speedups for this problem are pictured in the graph of figure 6.16(b). As can be seen, for a system of these dimensions, TRIMM with the broadcast mode enabled outperforms the ring configuration by an even greater margin.
Unfortunately, due to the allowance of simultaneous acceptances, the accuracy of the estimate is not the same in both parallel and serial cases. The speedup cannot be calculated purely as a function of the time required to process the elements of the estimate, but instead is a function of the dynamics associated with the global communications scheme and the decomposition of the reconstruction space.

**Domain Decomposition**

The first question to be addressed is that of decomposition of the reconstruction space among the processors. The way in which the reconstruction space is decomposed presents two concerns: the first being the location of the boundaries between local domains, and the second being just how finely the object can be divided among the processors and still converge to an acceptable estimate.

The boundaries between local domains are a consideration only if there is a term in the energy functional that requires the entire estimate for calculation. For example, because the estimate values for the nearest neighbors are required for calculation of the smoothing term, the accept/reject decision for perturbations applied to boundary elements are likely to be based on less current information than that used for decisions on nonboundary elements. The reason for this is that some of the elements involved in this calculation are contained in domains other than that local to the specific processor in question. This effect results in a desire to have as many nearest neighbors as possible in the same local domain. For the reconstructions of the thyroid and brain phantom, the boundaries are not a concern because no additional smoothing is used.

The question of how finely a reconstruction space can be divided among the processors and still converge is the heart of the problem. The more finely the estimate is divided, the more parallel operation will be realized. Although a finer reconstruction-space division results in greater speed in terms of perturbations/second attempted, the
resulting reconstruction is not guaranteed to be of good visual quality. Since each processor is responsible for reconstructing a smaller fraction of the total reconstruction space, more of the information on which a perturbation trial decision is based is not contained locally, but instead in the local domains of other processors. Therefore, the information used to determine whether or not a perturbation is accepted into the local estimate cannot help but be less current the more finely the reconstruction space is divided. As a result, a greater degree of synchronization is required than would be needed to achieve the same level of convergence as that achieved in a more coarsely divided case.

**Thyroid Phantom -- Experimental Results**

The thyroid phantom (fig. 6.10) was digitized on a 55x55 grid (N = 3025) and imaged utilizing 1024 (= M) detector elements. The energy functional is the standard data-agreement term (eq. 5.2) with the only additional constraint being that of positivity. The initial estimate for the reconstruction is the collected data back-projected into object space. The initial estimate and the serial reconstruction are shown in figure 6.17a.

The parallel architecture used to reconstruct the thyroid phantom is the ring architecture. Ring sizes of 12, 24, 36, 48, and 60 processors were used for investigation. For each ring size, all three communications schemes were implemented, and the one that produced an estimate with energy value closest to that achieved in the serial reconstruction, in the shortest period of time, was considered the most appropriate for that specific configuration.

The reconstructions and the associated reconstruction-space decompositions for each architecture are shown in figures 6.17b through 6.17f. The corresponding graphs tracking the energy of the estimate as a function of time for each reconstruction are shown in figures 6.18a through 6.18b.
As expected, the more finely decomposed the reconstruction space is among the processors, the greater the degree of synchronization required to reach the energy level achieved in the serial case. For example, consider the graphs in figures 6.18b and 6.18c. The graph in figure 6.18b tracks the energy of the estimate for the 12 processor system. With this system size, each processor is responsible for reconstructing approximately 8.3% of the object. The asynchronous communication scheme, requiring 90 passes through the reconstruction space, produced an estimate with the desired energy value (the energy value of the serially reconstructed estimate) in the shortest period of time. The graph in figure 6.18c tracks the energy of the estimate for a 24-processor system in which each processor is responsible for approximately 4.1% of the reconstruction space. Due to the finer division of the estimate among the processors, a reconstruction utilizing the asynchronous communication scheme failed to produce an estimate with an energy value comparable to that of the serial estimate. Therefore, a communications scheme providing a greater degree of synchronization had to be implemented. The hybrid communication scheme, with synchronization occurring after every two passes through the reconstruction space, was used to provide the necessary synchronization. For this configuration, 180 passes through the reconstruction space were required to achieve the desired energy value. The resultant speedup (over the serial case) was only 7.8. Although twice as many processors were used in this case, the additional synchronization and the greater number of passes through the reconstruction space resulted in less of a speedup than achieved in the 12-processor configuration -- 7.3 vs. 8.3. The 36 processor configuration required an even greater degree of synchronization (utilizing the synchronous communications scheme), but only 130 passes through the reconstruction space to produce an estimate with the target energy level. Although 50 fewer passes through the reconstruction space were required for the 36 processor configuration versus the 24 processor configuration,
an increased speedup (5.4) is not realized because of the increased synchronization required for convergence.

It is worth noting the variation in the number of passes through the reconstruction space required for convergence for the different configurations. The more finely the reconstruction space is decomposed among the processors, the greater the number of times a processor can complete a pass through its local space in a given period of time. Unfortunately, as previously discussed, the more finely the reconstruction space is decomposed, the less current the global information upon which each processor is basing its perturbation-trial decision. More perturbation decisions are made, but many of these decisions are in error until the processors can be appropriately synchronized. It is not that more passes through the reconstruction space are required for finer decompositions, but rather because of the resultant smaller local domains and the speed of the processors, more passes are attainable. As the communication speed catches up to the processing speed, the relative speedup will increase as the reconstruction space is more finely decomposed.

The comparative speedup (measured as a function of constant energy) for the reconstructions of the thyroid phantom for the above configurations is shown in the graph in figure 6.19.

**Brain Phantom -- Experimental Results**

The same procedures used for the reconstructions of the thyroid phantom were followed for the brain phantom. The parallel architecture used for reconstruction was the ring architecture with ring sizes of 12, 24, 36, 48, and 60 processors. For each ring size, all three communications schemes were implemented, and the one that produced an estimate with energy value closest to that achieved in the serial reconstruction, in the
shortest period of time, was considered the most appropriate for that specific configuration.

The initial estimate and the serial reconstruction are shown in figure 6.20a, and the reconstructions and associated reconstruction space decompositions for each configuration are pictured in figures 6.20b through 6.20f. The corresponding graphs tracking the energy of the estimate as a function of time for each reconstruction are shown in figures 6.21a through 6.21f.

As expected, since the reconstruction space for the brain phantom is larger than that of the thyroid, more processors can be used effectively for the parallel implementation of the algorithm. The comparative speedup (measured as a function of constant energy) for the reconstructions of the brain phantom for the above configurations is shown in the graph in figure 6.22.

It is interesting to note the inherent smoothing in the parallel reconstructions. This effect is also due to simultaneous acceptances of perturbations into the estimate. For the more highly parallel cases (36, 48, and 60 processors), the energy of the estimate oscillates as it nears its minimum. In doing so, perturbations are continually accepted into the estimate based on the current global estimate. If the energy of the current global estimate in a particular processor is high, then perturbations will be accepted to lower the energy. Likewise, if the energy of the current global estimate is low, then perturbations will be accepted to raise the energy. This effect, occurring simultaneously in every processor, produces in the smoothing effect noticed in the reconstruction.

It is also worth noting the effect, or lack thereof, of the boundaries that result from the decomposition of the reconstruction space. Although some striations appear in the more highly parallel reconstructions, it is pleasantly surprising that the effect is not more pronounced. It is quite possible that the smoothing effect just discussed is responsible for reducing any boundary effects that might otherwise have resulted.
Summary

To summarize, the results of the parallel implementations of the Monte Carlo search algorithm indicate that there is an optimal configuration - number of processors and global communications scheme - for a specific reconstruction-space size. Due to simultaneous acceptances of perturbations into the estimate, varying amounts of synchronization are required for convergence based on how finely the reconstruction space is decomposed among the processors. Therefore, due to increased synchronization requirements, more processors do not necessarily provide faster results.

The difference in calculated speedup, determined from the number of elements processed, and the actual speedup determined from the time required to reach the target energy of the estimate is startlingly large. The primary reason for this is the dynamics of the reconstruction process that produce simultaneous acceptances of perturbations. However, whether or not the energy is a fair measure of the equivalence of two estimates is debatable. The question of just what factors give an accurate measure of subjective visual image is a topic of continuing research and not within the scope of this dissertation.
Fig. 6.1. TRIMM ring configuration. The double lines indicate the transputer links used to form the ring architecture. The single lines represent links that are not connected in this configuration. Communication between the ring and the IBM AT host was done via a T800 transputer located on the Inmos B004 transputer card. The B004 card hosts four T800 transputers and an interface to the AT ISA bus.
Fig. 6.2. Flowchart of the synchronous communication scheme as implemented on the ring architecture.
Fig. 6.3. Flowchart of the asynchronous communication scheme as implemented on the ring architecture.
Fig. 6.4. The binary myocardium.
Fig. 6.5. Configuration for the imaging of the binary myocardium. The face of each modular camera -- a 4"x4" NaI scintillation crystal - can be thought of as being divided into pixels. These "pixels" form the data vector $g$. 
for (j = 0; j < N; j++) /*calculate HTHf*/
    sum += ((float)hPTR->hth[i][j] * (float)f[j]);

if (frandf() < 0.5) /*determine sign*/
    del_f = (-grn_sze); /*of perturbation*/
else
    del_f = (grn_sze);

del_E = (2.0 * ((sum - g[Klim + i]) * del_f) + ((float)hPtr->hth[i][j] * (del_f * del_f)); /*calculate the change*/
      /*in energy of estimate*/

Fig. 6.6. Computer code (C language) for the inner loop of the object-space $\Delta E$ formulation.
if (frandf() < 0.5) /*determine sign*/
    del_f = (-grn_sze);
else
    del_f = (grn_sze);

for (j = 0; j < M; j++) /*zero Hf*/
    Hdel_f[j] = 0.0;

rowx = 0;
for (j = 0; j < hPtr->nelem[i]; j++)
{
    nzrow = (int)hPtr->nzrtag[i][j]; /*identify nonzero*/
    if (j > 0) /*elements of H*/
    {
        if (nzrow < (int)hPtr->nzrtag[i][j-1])
            row++;
    }
    currow = (rowx * 256) + nzrow; /*increment H row ptr*/
    Hdel_f[currow] = (hPtr->u.nzrval[i][j] * (float)del_f); /*calc Hf*/

del_E = 0.0;
for (j = 0; j < M; J++)
    del_E += (((2.0 * (Hf[j] - g [j])) + Hdel_f[j]) * Hdel_f[j]);/*calculate the*/
        /*change in*/
        /*energy of estimate*/

Fig. 6.6. Computer code (C language) for the inner loop of the data-space ΔE formulation.
Fig. 6.7. Reconstruction space decomposition among the six transputer modules. Each gray level indicates a unique local domain corresponding to a single processor.
Fig. 6.8. Reconstruction of the binary myocardium (fig. 6.4) utilizing the data-space $\Delta E$ formulation. Each stage of development represents to passes through the entire reconstruction space. The Hamming distance for each stage pictured is 123, 76, 60, and 52, respectively (The maximum value of the Hamming distance is 1521.)
Fig. 6.9. Reconstruction of the binary myocardium (fig. 6.4) utilizing the data-space $\Delta E$ formulation. Each stage of development represents $2$ passes through the entire reconstruction space. The Hamming distance for each stage pictured is 212, 153, 130, 113, 101, and 95, respectively. (The maximum value of the Hamming distance is 1521.)
Fig. 6.10. Digitized version of the thyroid phantom.
Fig. 6.11. Digitized version of the Hoffman brain phantom.
Fig. 6.12. The octagonal imaging configuration for the thyroid and brain phantoms.
Fig. 6.13(a). Speedup vs. number of processors for the ring architecture implementing the asynchronous communications scheme and the data-space $\Delta E$ formulation. Every perturbation is accepted and no positivity is applied.
Fig. 6.13(b). Speedup vs. number of processors for the ring architecture implementing the synchronous communications scheme and the data-space $\Delta E$ formulation. Every perturbation is accepted and no positivity is applied.
Fig. 6.14(a). Speedup vs. number of processors for TRIMM with broadcast mode enabled implementing the asynchronous communications scheme and the data-space $\Delta E$ formulation. Every perturbation is accepted and no positivity is applied.
Fig. 6.14(b). Speedup vs. number of processors for TRIMM with broadcast mode enabled implementing the synchronous communications scheme and the data-space $\Delta E$ formulation. Every perturbation is accepted and no positivity is applied.
Fig. 6.15(a). Reconstruction speedup of the brain phantom. The ring architecture was utilized and both the asynchronous and synchronous update schemes implemented. The time measured is the time required to complete a single pass through the reconstruction space; every perturbation is accepted and positivity is the only additional constraint enforced.
Fig. 6.15(b). Reconstruction speedup of the brain phantom. The TRIMM with broadcast mode enabled was utilized and both the asynchronous and synchronous update schemes implemented. The time measured is the time required to complete a single pass through the reconstruction space; every perturbation is accepted and no additional constraints are enforced.
Fig. 6.16(a). Theoretical speedup of a hypothetical reconstruction utilizing data collected from the proposed brain imager. The ring architecture was utilized and both the asynchronous and synchronous update schemes implemented. The time measured is the time required to complete a single pass through the reconstruction space; every perturbation is accepted and no additional constraints are enforced.
Fig. 6.16(b). Theoretical speedup of a hypothetical reconstruction utilizing data collected from the proposed brain imager. TRIMM, broadcast mode enabled, was utilized and both the asynchronous and synchronous update schemes implemented. The time measured is the time required to complete a single pass through the reconstruction space; every perturbation is accepted and no additional constraints are enforced.
Fig. 6.17(a). On the left, the initial estimate for the reconstruction of the thyroid phantom. The initial estimate consists of the data back-projected into object space. On the right, a serial reconstruction of the thyroid phantom implemented on a single transputer. Positivity is the only constraint applied. Seventy passes through the object space were required and the temperature was set to zero.
Fig. 6.17(b). A parallel reconstruction of the thyroid phantom implemented on the ring architecture consisting of 12 processors is shown on the right. The object-space decomposition is shown on the left, with each gray scale representing the unique local domain that each respective processor is responsible for reconstructing. The asynchronous update scheme was used and positivity the only constraint applied. Approximately 90 passes through the reconstruction space were required to achieve a reconstruction comparable to that achieved serially in 70 passes.
Fig. 6.17(c). Same as fig. 6.17(b) except the ring consists of 24 processors, the hybrid update scheme was used with synchronization occurring after every 2 passes through the reconstruction space, and approximately 180 passes through the reconstruction space were required.
Fig. 6.17(d). Same as fig. 6.17(b) except the ring consists of 36 processors, the synchronous update scheme was used, and approximately 130 passes through the reconstruction space were required.
Fig. 6.17(e). Same as fig. 6.17(b) except the ring consists of 48 processors, the synchronous update scheme was used, and approximately 180 passes through the reconstruction space were required.
Fig. 6.17(f). Same as fig. 6.17(b) except the ring consists of 60 processors, the synchronous update scheme was used, and approximately 340 passes through the reconstruction space were required.
Fig. 6.18(a). Energy vs. time for the reconstruction of the thyroid phantom shown in fig. 6.17(a).
Fig. 6.18(b). Energy vs. time for the reconstruction of the thyroid phantom shown in fig. 6.17(b).
Fig. 6.18(c). Energy vs. time for the reconstruction of the thyroid phantom shown in fig. 6.17(c).
Fig. 6.18(d). Energy vs. time for the reconstruction of the thyroid phantom shown in fig. 6.17(d).
Fig. 6.18(e). Energy vs. time for the reconstruction of the thyroid phantom shown in fig. 6.17(e).
Fig. 6.18(f). Energy vs. time for the reconstruction of the thyroid phantom shown in fig. 6.17(f).
Fig. 6.19. Speedup for the reconstruction of the thyroid phantom.
Fig. 6.20(a). On the left, the initial estimate for the reconstruction of the thyroid phantom. The initial estimate consists of the data back-projected into object space. On the right, a serial reconstruction of the Hoffman brain phantom implemented on a single transputer. Positivity is the only constraint applied. One hundred and thirty passes through the object space were required and the temperature was set to zero.
Fig. 6.20(b). A parallel reconstruction of the Hoffman brain phantom implemented on the ring architecture consisting of 12 processors is shown on the right. The object-space decomposition is shown on the left, with each gray scale representing the unique local domain that each respective processor is responsible for reconstructing. The asynchronous update scheme was used and positivity the only constraint applied. Approximately 50 passes through the reconstruction space were required to achieve a reconstruction comparable to that achieved serially in 130 passes.
Fig. 6.20(c). Same as fig. 6.20(b) except the ring consists of 24 processors, the asynchronous update scheme was used, and approximately 180 passes through the reconstruction space were required.
Fig. 6.20(d). Same as fig. 6.20(b) except the ring consists of 36 processors, the hybrid update scheme was used with synchronization occurring after every 2 passes through the reconstruction space, and approximately 80 passes through the reconstruction space were required.
Fig. 6.20(e). Same as fig. 6.20(b) except the ring consists of 48 processors, the synchronous update scheme was used, and approximately 90 passes through the reconstruction space were required.
Fig. 6.20(f). Same as fig. 6.20(b) except the ring consists of 60 processors, the synchronous update scheme was used, and approximately 70 passes through the reconstruction space were required.
Fig. 6.21(a). Energy vs. time for the reconstruction of the Hoffman brain phantom shown in fig. 6.20(a).
Fig. 6.21(b). Energy vs. time for the reconstruction of the Hoffman brain phantom shown in fig. 6.20(b).
Fig. 6.21(c). Energy vs. time for the reconstruction of the Hoffman brain phantom shown in fig. 6.20(c).
Fig. 6.21(d). Energy vs. time for the reconstruction of the Hoffman brain phantom shown in fig. 6.20(d).
Fig. 6.21(e). Energy vs. time for the reconstruction of the Hoffman brain phantom shown in fig. 6.20(e).
Fig. 6.21(f). Energy vs. time for the reconstruction of the Hoffman brain phantom shown in fig. 6.20(f).
Fig. 6.22. Speedup for the reconstruction of the Hoffman brain phantom.
TABLE 6.1
Configurations for Study

<table>
<thead>
<tr>
<th>ARCHITECTURE</th>
<th>GLOBAL UPDATE IMPLEMENTATION</th>
<th>ASYNCHRONOUS</th>
<th>SYNCHRONOUS</th>
<th>HYBRID</th>
</tr>
</thead>
<tbody>
<tr>
<td>RING</td>
<td></td>
<td>Binary Myocardium</td>
<td>Binary Myocardium</td>
<td>Thyroid Phantom</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Thyroid Phantom</td>
<td>Thyroid Phantom</td>
<td>Hoffman Brain Phantom</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hoffman Brain Phantom</td>
<td>Hoffman Brain Phantom</td>
<td>Hoffman Brain Phantom</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Brain Imager Data Set</em></td>
<td><em>Brain Imager Data Set</em></td>
<td></td>
</tr>
<tr>
<td>TRIMM with Broadcast Enabled</td>
<td>Thyroid Phantom</td>
<td>Thyroid Phantom</td>
<td>Hoffman Brain Phantom</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hoffman Brain Phantom</td>
<td>Hoffman Brain Phantom</td>
<td>Hoffman Brain Phantom</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>Brain Imager Data Set</em></td>
<td><em>Brain Imager Data Set</em></td>
<td><em>Brain Imager Data Set</em></td>
<td></td>
</tr>
<tr>
<td>TRIMM with Broadcast Disabled</td>
<td>Thyroid Phantom</td>
<td>Thyroid Phantom</td>
<td>Hoffman Brain Phantom</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hoffman Brain Phantom</td>
<td>Hoffman Brain Phantom</td>
<td>Hoffman Brain Phantom</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>Brain Imager Data Set</em></td>
<td><em>Brain Imager Data Set</em></td>
<td><em>Brain Imager Data Set</em></td>
<td></td>
</tr>
</tbody>
</table>

Objects reconstructed from measured data

*Theoretical considerations only*
TABLE 6.2
ΔE Formulations: Implementation Factors

<table>
<thead>
<tr>
<th>Number of Inner-Loop Operations</th>
<th>Requisite Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ΔE(Δg)</td>
<td>7M</td>
</tr>
<tr>
<td>ΔE(Δf)</td>
<td>N²</td>
</tr>
</tbody>
</table>

*M = Dimension of the data vector, g
N = Dimension of the object vector, f

* Operations include both additions and multiplications

TABLE 6.3
ΔE Formulations: Experimental Results

<table>
<thead>
<tr>
<th>Speed</th>
<th>Memory Storage Req'd</th>
<th>Reconstruction Precision (Hamming Distance)</th>
<th>Reconstruction Accuracy (Convergence)</th>
<th>Choice for Future Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ΔE(Δg)</td>
<td>+</td>
<td>+</td>
<td>=</td>
<td>+</td>
</tr>
<tr>
<td>ΔE(Δf)</td>
<td>-</td>
<td>-</td>
<td>=</td>
<td></td>
</tr>
</tbody>
</table>

+ Superior performance
- Inferior performance
= equivalent performance
TABLE 6.4
Thyrid Phantom: Experimental Results

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Domain Decomposition (pixels/processor)</th>
<th>Global Update Scheme</th>
<th># Passes Through Object Space</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>252</td>
<td>ASYNC</td>
<td>90</td>
<td>8</td>
</tr>
<tr>
<td>24</td>
<td>126</td>
<td>HYB</td>
<td>180</td>
<td>7</td>
</tr>
<tr>
<td>36</td>
<td>84</td>
<td>SYNC</td>
<td>130</td>
<td>6</td>
</tr>
<tr>
<td>48</td>
<td>63</td>
<td>SYNC</td>
<td>180</td>
<td>5</td>
</tr>
<tr>
<td>60</td>
<td>50</td>
<td>SYNC</td>
<td>340</td>
<td>4</td>
</tr>
</tbody>
</table>
TABLE 6.5
Hoffman Brain Phantom: Experimental Results

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Domain Decomposition (pixels/processor)</th>
<th>Global Update Scheme</th>
<th># Passes Through Object Space</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>420</td>
<td>ASYNC</td>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td>24</td>
<td>210</td>
<td>ASYNC</td>
<td>60</td>
<td>18</td>
</tr>
<tr>
<td>36</td>
<td>140</td>
<td>HYB</td>
<td>80</td>
<td>12</td>
</tr>
<tr>
<td>48</td>
<td>104</td>
<td>SYNC</td>
<td>90</td>
<td>9</td>
</tr>
<tr>
<td>60</td>
<td>84</td>
<td>SYNC</td>
<td>100</td>
<td>6</td>
</tr>
</tbody>
</table>
CHAPTER 7
SUMMARY AND SUGGESTIONS FOR FUTURE WORK

The work discussed in the previous chapters reflects the potential of parallel processing in both hardware and software. The following summarizes the work done in this dissertation and suggests avenues for future research.

The primary contribution of the work presented in this dissertation was the parallel implementation of the special case of the simulated annealing algorithm with the temperature set to 0, i.e., the Monte Carlo search algorithm. The algorithm was successfully implemented on the TRIMM parallel processor. Two formulations of the inner-loop of the algorithm were investigated, with the data-space formulation shown to provide superior performance. Three global communications schemes, each with a different degree of synchronization, were implemented in the reconstruction process. The decomposition of the reconstruction space among the processors was investigated and it was found that the more finely the reconstruction space is divided, the more synchronization between the processors is required.

The primary reason for the achieved speedup, the allowance of simultaneous acceptances of perturbations into the estimate, is also the factor that limits the achievable degree of parallelism. This is because the more finely the reconstruction space is divided, the more inaccurate are the decisions to accept perturbations into the estimate. In addition to requiring a greater degree of synchronization to achieve convergence, it was shown that the reconstruction space must be traversed more times. The more inaccurate decisions that are made, the more the estimate is forced to wander about the structure defined by the energy values, not always traveling in the direction of the global estimate.

The reconstructions resulting from the parallel implementations indicate that for a specific size of problem there is an optimal number processors that should be used. This
decision is based on the speedup, as measured by the level of energy achieved, and the resulting reconstruction in comparison to the serial reconstruction. Because the optimal configurations for the objects reconstructed here utilized relatively few processors, it may be inferred that TRIMM, in its present state, is equipped with more than adequate processing power but that insufficient memory may provide a constraint in the future.

Suggestions for Future Work

The most immediate question to be addressed is the incorporation of the broadcast mode into the global communications and the investigation of the associated dynamics governing the evolution of the estimate.

At the heart of the problem, is the energy functional to be minimized. It has been shown that an energy functional that is not dynamically dependent on the global-update scheme is applicable to the reconstruction problem. Further investigation into various different energy functionals and regularization and noise-control terms and their respective behaviors in a parallel environment is necessary toward gaining a better understanding of the functional parallelism of the reconstruction problem.

Because of the strong interplay between hardware and software, and the flexibility of TRIMM, all of the variables investigated in this dissertation -- inner-loop formulation, reconstruction-space decomposition, and global communications -- will require further investigation depending on the hardware configuration used. Particularly of interest will be the adaptation to fully three-dimensional reconstructions. In the three-dimensional case, the boundaries become surfaces instead of lines and communications may have to be provided for another dimension. A ring configuration for implementing a three-dimensional reconstruction could be envisioned as an architecture consisting of a ring of rings where x number of processors, configured in a ring, are responsible for the
reconstruction of a single two-dimensional slice and these rings are connected in a three-dimensional ring to provide communications between the slices.

In anticipation of problems where the energy functionals to be minimized contain local minima, and hence the solution requires annealing, an appropriate annealing schedule should be investigated; however, determining an optimal cooling schedule for even the serial implementation of the simulated annealing algorithm is a continuing topic of research. Implementing the algorithm in parallel complicates the situation further by introducing even more variables into the problem. Most important of these variables is the allowance of perturbations that would normally be rejected, to be accepted into the estimate via acceptance decisions made on simultaneous perturbation trials. These additional acceptances will greatly affect the statistics used to determine when an estimate has reached equilibrium for a given temperature; to determine just how great this effect is will require further research. The acceptance of perturbations that decrease the energy of the estimate on a local level but increase the energy of the global estimate, suggests that there may be a property analogous to the temperature in conventional simulated annealing (perhaps the degree of synchronization) associated with the parallel implementation; for instance, possibly increasing the degree of synchronization in the parallel implementation would have the same effect as lowering the temperature in the serial implementation.
REFERENCES


REFERENCES -- Continued


REFERENCES -- Continued


REFERENCES -- Continued