AN ABSTRACT OF THE THESIS OF

Phyllis E. Crandall for the degree of Doctor of Philosophy in
Computer Science presented on April 19, 1994.
Title: Data Decomposition and Load Balancing for Networked Data-Parallel Processing

Abstract approved: ___

Michael J. Quinn

The improving speeds of networks and microprocessors, and the recent interest in heterogeneous parallel computing have given rise to a new parallel architecture, the parallel network. Networked parallelism has much in common with the loosely-coupled multicomputer model of computation, but it poses specific challenges that must be addressed before this paradigm can be of practical value. Three issues are central to this computational environment: high internode communication cost, heterogeneous node-performance capabilities, and fluctuating node performance due to multiuser workloads. This research has examined a number of common partitioning methods to determine those that are suitable for this programming environment. New block decomposition algorithms have been developed to accommodate the heterogeneity of the parallel network, and these are capable of offering better performance than previously proposed partitioning methods. The communication costs associated with typical partitioning techniques have been mathematically characterized in a way that permits evaluation of the relative value of various decomposition schemes for specific applications based on their communication patterns and size. A decomposition advisory system is presented that uses these mathematical characterizations, knowledge about the configuration of the network and its processors, and information about the application problem to provide advice regarding the partitioning method expected to yield the best performance.
Data Decomposition and Load Balancing
for Networked Data-Parallel Processing

by

Phyllis E. Crandall

A THESIS
submitted to
Oregon State University

in partial fulfillment of
the requirements for the
degree of
Doctor of Philosophy

Completed April 19, 1994
Commencement June 1994
APPROVED:

Redacted for Privacy

Professor of Computer Science in charge of major

Redacted for Privacy

Head of department of Computer Science

Redacted for Privacy

Dean of Graduate School

Date thesis is presented April 19, 1994

Typed by researcher for Phyllis E. Crandall
# Table of Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.1 The Parallel Network</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1.2 The Problem Space</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1.3 Research Contributions</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>1.4 Significance</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>1.5 Organization of the Thesis</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Related Work</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>2.1 Distributed Systems</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>2.2 Control-Parallel Processing</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>2.3 Data-Parallel Processing</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>2.4 Analysis of Load Balancing Systems</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>2.5 Workstation Clusters</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>Network Considerations</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>3.1 Ethernet</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>3.2 Token Ring</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>3.3 Fiber Distributed Data Interface (FDDI)</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>3.4 High Performance Parallel Interface (HiPPI)</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>3.5 Asynchronous Transfer Mode (ATM)</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>3.6 Switched Networks</td>
<td>38</td>
</tr>
</tbody>
</table>
7 A Data Decomposition Advisory System

7.1 Description of the System ................................................................. 94
7.2 Experimental results ................................................................. 99
7.3 Value and future directions for the advisory system .................. 101

8 Conclusions

8.1 The Problem .............................................................................. 103
8.2 Significance of this Research .................................................. 104
8.3 Future Directions ................................................................. 105

Bibliography ........................................................................ 107
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>An example of the 2-dimensional grid representation of a Jacobi-style problem space. The communication pattern is a 5-point stencil.</td>
<td>5</td>
</tr>
<tr>
<td>2.</td>
<td>Examples of (a) uniform, (b) non-uniform, and (b) irregular grids.</td>
<td>6</td>
</tr>
<tr>
<td>3.</td>
<td>Notation used to analyze communication time.</td>
<td>11</td>
</tr>
<tr>
<td>4.</td>
<td>An example of mapping a 10 × 10 grid to a 3 × 3 mesh using the technique proposed by Snyder and Socha.</td>
<td>23</td>
</tr>
<tr>
<td>5.</td>
<td>The ripple effect in Dataparallel C. In (a) each processor is marked with the number of processes it has and the number of processes it should have (i.e., processor 0 has 10 and should have 5). (b) shows the necessary data movement to achieve the correct balance, which is shown in (c).</td>
<td>31</td>
</tr>
<tr>
<td>6.</td>
<td>A switched network where the switch may be a crossbar or a high bandwidth connector.</td>
<td>39</td>
</tr>
<tr>
<td>7.</td>
<td>Multiple network connections. (a) A ring with two networks. (b) A ring of rings.</td>
<td>40</td>
</tr>
<tr>
<td>8.</td>
<td>A 3-dimensional hypercube formed from 8 workstations and 6 networks. The 6 networks are (0,1,3,2), (0,1,5,4), (0,2,6,4), (1,3,7,5), (2,3,7,6), (4,5,7,6).</td>
<td>42</td>
</tr>
</tbody>
</table>
9. An example of scatter decomposition for a grid problem on four heterogeneous processors with each colored square representing a data point and the processor to which it is assigned.

10. Contiguous point partitioning of a grid problem for four heterogeneous processors.


12. Two interleaved partitionings for four processors. (a) The homogeneous case. (b) The heterogeneous case with processors whose relative speeds are 4,2,2,1.


15. The types of blocks generated in the general quadrilateral method. Arrows indicate the number of communications. (a) Corner blocks generate a maximum of 6 communications. (b) Edge blocks generate at most 7 communications. (c) Interior blocks may have as many as 8 communications.

16. The toroidal problem space may create additional communications with a non-homogeneous decomposition. (a) A homogeneous partitioning has no extraordinary communications. (b) The general quadrilateral and (c) binary recursive decomposition may incur additional communications on each edge equal to one less than the number of processors on the opposing edge. These grids wrap along both the horizontal and vertical edges.

17. The steps necessary to partition a grid according to BRBD for 5 heterogeneous processors. The relative speed table is given at bottom right.
18. Binary recursive block decomposition for 6 processors with the
given relative speed table.

19. The worst case binary recursive block decomposition for 4 pro-
cessors. (a) shows the 10 communications internal to the grid.
(b) shows the 6 east/west communications due to wrapping and
(c) gives the 4 north/south wrapping communications.

20. Binary Recursive Block Decomposition Algorithm

21. The steps necessary to partition a grid according to FBRD for 5
heterogeneous processors with speeds (3,3,3,2,2).

22. Fair Binary Recursive Decomposition Algorithm

23. A partial homogeneous partitioning for 9 processors with relative
speeds (4,4,3,3,2,2,1,1).

24. Partial Homogeneous Decomposition definitions and preparation
for partitioning.

25. Partial Homogeneous Decomposition partitioning algorithm for
a single group—the base case.

26. Partial Homogeneous Decomposition partitioning algorithm for
multiple groups.

27. A comparison of partial homogeneous partitioning with a binary
recursive partitioning. The number within each sub-block indi-
cates the number of communications required by that sub-block
during each iteration. The BRBD requires 42 communications
and transmits 13n data items. FBRD sends 43 messages and
transmits only 11n items. PHD requires 50 communications and
sends 12n data points. This example assumes 9 processors with
speeds (4,4,3,3,3,2,2,1,1).
28. A comparison of partial homogeneous partitioning with a binary recursive partitioning. The number within each sub-block indicates the number of communications required by that sub-block during each iteration. Here the BRBD generates 28 communications and transmits 11n data items. FBRD requires 32 messages with 11n data items. PHD generates 24 messages and sends 10n items. The 9 processors are assumed to have speeds (4,4,4,3,3,3).

29. Data movement for different communications patterns. (a) Broadcast—a single value is disseminated to all grid points. (b) Reduction—data items are transmitted from all grid points to a single receiving grid point. (c) Systolic—every data item is synchronously sent to its nearest neighbor. (d) Wavefront—each data item depends on the values to the left and above. (e) Successor/Predecessor—neighboring values along one dimension are needed. (f) 5-Point Stencil—every data item needs access to the values of its nearest neighbors to the north, south, east, and west.

30. Time lines for four message transmission scenarios. Cases (a), (b) and (e) incur the maximum communication cost, while (c) and (d) overlap the message preparation latency of one processor with the transmission times of other processors.

31. A systolic communication pattern with the grid divided for 4 processors according to the decompositions methods—(a) scatter, (b) contiguous point, (c) contiguous row, (d) interleaved row, (e) homogeneous block, and (f) heterogeneous block.

32. Typical stencil patterns for communication in grid problems that are amenable to block decomposition. (a) 3-point stencil. (b) 5-point stencil. (c) 9-point stencil.
33. A 5-point stencil communication pattern decomposed by (a) scatter, (b) contiguous point, (c) contiguous row, (d) interleaved, (e) homogeneous block, and (f) heterogeneous block. 

34. Number of communications for the five decomposition methods for homogeneous processors. 

35. Number of communications for the five decomposition methods for heterogeneous processors. 

36. Communication cost for five decomposition methods for the 5-point stencil pattern on an $n \times n$ grid. There 8 processors and a data size of 8 bytes. $\lambda = 2.5$ msec, $\tau = 1.5$ $\mu$sec, MTU = 1460 bytes, frame headers and trailers = 58 bytes. 

37. Minimum number of grid points required for block allocation to be superior to contiguous row or column allocation, as a function of number of processors. Block allocation is never superior to contiguous row or column allocation when there are fewer than 5 processors. 

38. Communication cost for the contiguous and block decomposition methods on heterogeneous processors for the 5-point stencil pattern on an $n \times n$ grid. There are 8 processors and a data size of 8 bytes. $\lambda = 2.5$ msec, $\tau = 1.5$ $\mu$sec, MTU = 1460 bytes, frame headers and trailers = 58 bytes. The relative speeds are (6,4,4,3,3,3,2,1).
39. Top-level view of the partitioning advisory system. There are 4 distinct phases. (1) Estimating processor speeds. (2) Partitioning the problem space. (3) Estimating network performance levels. (4) Applying the mathematical communication cost characterizations to the gathered data. The arrows indicate data going to and coming from the various system segments. The final output consists of the relative performance rankings for the different partitionings.

40. The partitioning phase establishes the size and shape of the data point allocation depending on the relative processor speeds.

41. To estimate the current communication performance capability of the network, several rounds of message passing are launched. The times reported are used to estimate the current values of \( \lambda \) and \( \tau \).

42. The output of the final advisory phase. The values indicate the relative worth of the two partitioning strategies. The method expected to give the best performance has a rating of 1.00. The ratings of the other schemes are estimates of the factors by which the execution time is expected to increase (e.g., 1.39 indicates a 39% increase in execution time).

43. The average ratios of predicted block versus row performance to observed block versus row performance. Two forms of the thermal transfer application were tested. The top graph shows the results when there are only 10 flops per data point per iteration. The second requires 50 flops per data point per iteration. The 95% confidence intervals for these averages are given.

44. A true heterogeneous network.
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>The number of nodes, diameter, bisection bandwidth, and number of connections at each node for the single network, the completely connected workstation network, the cube-connected workstation network, and the 2-dimensional mesh workstation network.</td>
<td>44</td>
</tr>
<tr>
<td>2.</td>
<td>The number of messages generated by the various decomposition methods for a 5-point stencil communication pattern on homogeneous and heterogeneous processors.</td>
<td>87</td>
</tr>
</tbody>
</table>
Parallel processing has become an important tool for solving many of today’s real-world problems. Environmental modeling, genome sequencing, pharmaceutical modeling, genetic engineering, and oceanographic simulation are just a sampling of problems that have such great computational demands that sequential computation is often impractical. Consider a global circulation simulation that requires about 30 billion floating-point operations for every 10 minutes being modeled. Computational requirements such as these cannot be met with standard sequential computing facilities. Given a 100 Mflops (millions of floating-point operations per second) machine, the global circulation model would take 182.5 days to simulate one year’s activity [76]. There are examples that would take tens of thousands of years to execute, even on today’s fastest supercomputers.

The recent interest in heterogeneous parallel computing and improving network speeds have given rise to a new parallel architecture, the parallel network. This new parallel processing paradigm presents challenges beyond those found with traditional parallel machines. Heterogeneity of the nodes, low bandwidth, high message-preparation latency, and fluctuating performance capabilities due to multiple system users are issues that usually do not occur, or occur to a much lesser degree, in typical parallel computers. The adverse effects of these obstacles can be lessened by careful
data partitioning and load balancing. The problem addressed in this investigation is that of determining the decomposition methods that best suit this heterogeneous networked environment for data-parallel applications.

It is not the purpose of this research to predict performance in the heterogeneous parallel network. Rather the goal is to explore data decomposition and load balancing techniques to improve the performance attainable in this environment. In the remainder of this chapter the networked environment is described, the problem space used for the investigation is defined, and the goals and significance of the research are detailed.

It is my thesis that one can mathematically characterize the communication costs of a parallel network to determine the decomposition technique(s) that should yield superior performance for a specific application program.

1.1 The Parallel Network

The parallel network is defined to be a collection of independent computers, each with one or more processors, connected by one or more communication networks, for the purpose of solving a single problem.

Networked parallel processing has much in common with the loosely-coupled multicomputer model of computation. Both systems contain multiple processors that must communicate by message passing. Memory is physically distributed among the processors and each node has its own operating system. Execution is inherently asynchronous, which provides a suitable environment for MIMD (multiple instruction, multiple data) or SPMD (single program, multiple data) programming. There is no global name space, so data partitioning issues must be explicitly addressed. Networked parallel processing, however, poses a number of additional challenges that must be addressed before this computational paradigm can be of practical value.
Traditional parallel computers often have a mesh, hypercube or tree topology, and are specifically designed to optimize interprocessor communications. The actual topology of the parallel network may be some arbitrary configuration or a collection of individual networks. Typically, the connections for a Local Area Network (LAN) based network are Ethernet or Token Ring with bandwidths between 4 and 16 Mbps. However, faster networks are available. FDDI (Fiber Distributed Data Interface) offers a transmission rate of 100 Mbps, and HiPPI (High Performance Parallel Interface) is capable of performance of over 100 MBps [5]. Network bandwidth, however, is not the only component of communication cost. Regardless of the network speed, there is a message handling latency cost that must be paid. Protocols such as TCP/IP are implemented in software that creates a bottleneck in high speed networks. Latency costs for a single message may be 5 msec or greater, as opposed to less than 100 µsec in a parallel computer. Future protocols may be realized in silicon, thereby avoiding much of the software bottleneck; but current implementations are still software based. It is this message handling component of communication cost that has the most severe performance impact in network computing. To reduce this latency cost, it is desirable to keep the number of interprocessor messages as small as possible.

Unlike multicomputers, where all processors normally have the same computing power, parallel networks are often made up of heterogeneous machines with differing computational speeds. Therefore, the data partitioning scheme must consider the capabilities of the candidate machines when apportioning the workload. The problem space must be decomposed in a manner that minimizes the time that the faster processors must wait for slower processors to complete their computations. Ideally, each node is allotted a share of the workload commensurate with its performance capability.

The interconnections in the parallel network are seldom dedicated. It is often the case that the messages generated by a parallel application must contend for bandwidth
with unrelated network traffic. Additionally, the individual nodes are usually multi-user machines. This produces a high variability in the performance capabilities of the individual processors.

A number of freely-available programming platforms, such as PVM [45], P4 [25], and Dataparallel C [50], as well as numerous proprietary systems, facilitate parallel computing in the networked environment. The popularity of these systems attests to the widespread interest in the networked parallel processing paradigm. This concept of the hypercomputer, a virtual parallel machine formed from a network of workstations [26], has substantial appeal due to the availability, increasing speeds, and moderate cost of workstations, and the predicted superior price/performance ratio over supercomputers [16]. The investigation described here has been carried out in the context of the heterogeneous workstation network.

Three main issues, heterogeneity, communication cost, and the multi-user nature of the network, necessitate scrupulous partitioning of the data to achieve acceptable performance.

1.2 The Problem Space

The workstation network, like the multicomputer, is a suitable architecture for MIMD programming. MIMD, or control-parallel, programming is considered to be difficult because the available programming environments are typically very low level [50]. The programmer or the compiler must handle synchronization details and analyze data dependencies to determine an appropriate decomposition of the problem across the participating processors. Race conditions and deadlock are ever-present dangers, and debugging is difficult.

The SIMD (single instruction, multiple data), or data-parallel, model of computation involves the simultaneous execution of a single operation on different data items [50]. This synchronous flavor makes it easier for the programmer to conceptualize

Figure 1. An example of the 2-dimensional grid representation of a Jacobi-style problem space. The communication pattern is a 5-point stencil.

the parallel operations, and to avoid some of problems associated with control parallelism. The SIMD approach is applicable to many real-world problems in engineering and the sciences. Environmental modeling, pharmaceutical modeling, oceanographic simulation, and airfoil problems are just a few examples of areas with problems that are amenable to data-parallel treatment.

Before a program can be executed in parallel mode, it must be divided among the processors. In SIMD processing, the data, rather than entire tasks, are distributed across the processors. The most effective partitioning of the data depends heavily on the particular problem being solved and the relative speeds of the machines to be engaged.

In this investigation a class of data-parallel problems that we refer to as grid problems is considered. Such problems may be represented as a 1, 2, or multi-dimensional grid of data points, whose values change over a number of iterations in response to
the values of some subset of neighboring points. Atmospheric modeling, thermal conduction, and visualization problems are representatives of this class and are amenable to iterative methods such as Gauss-Seidel or Jacobi relaxation, and finite difference or multigrid methods [18, 43, 57]. These problems usually may be represented by a regular grid, and they exhibit regular communication patterns. This proclivity provides an opportunity to partition the data so that the number and/or size of required communications is minimized. Figure 1 depicts a 2-dimensional Jacobi-style problem space, where the communication pattern is a 5-point stencil. Figure 2 gives examples of uniform, non-uniform, and irregular grids. Non-uniform grids maintain rectangularity, while irregular grids are unstructured [71]. Uniform grids are assumed for this research.

There are four standard ways to apportion the data to the physical processors: by individual element, by row, by column, or by block. These may be further described by whether only one of these components is assigned to each processor, or whether some number are allotted in either contiguous, interleaved, or random fashion. Other, more exotic partitionings are possible [77], but have more limited applicability.
Some problems are *embarrassingly parallel* in that the tasks have little or no data dependencies. For these problems there is often little communication until results are finally gathered from the processors. Consider, for example, the Mapped Atmospheric-Plant-Soil System (MAPSS) model of predicting steady state vegetation for future climate conditions as described in [28]. Each site considered by the model may be processed completely in isolation from the others. There are no data dependencies, and no interprocessor communication is required before consolidating the results at the end of execution.

It is more often the case that dependencies exist among the tasks. These may be predictable, as with the nearest-neighbor dependencies in the Jacobi algorithm, or random as in Gaussian elimination with partial pivoting. In Gaussian elimination with partial pivoting, during the $i$th iteration the row with the largest element in the $i$th column is chosen as the pivot row. The dependencies are random in the sense that the order of the pivot rows is not known prior to run time. The communication patterns of the application often suggest the data decomposition method that will produce the best performance.

### 1.3 Research Contributions

The main motivation for parallel processing is speed. The heterogeneous workstation network has certain characteristics that may adversely affect its level of performance. Minimizing execution time means minimizing the communication costs as well as minimizing computation time. While the obvious solution to minimizing computation time is to use a large number of processors, this may actually lead to worse performance. Performance varies inversely with the number and size of communications, so there is a delicate balance between using a large number of machines to achieve high computational performance and small communication size, and using a small number of machines to limit the number of interprocessor communications.
The goals of this research have been to

1. Determine the partitioning methods that are expected to provide the best performance when applied to grid problems in network parallel processing;

2. Develop a block decomposition scheme suitable for the heterogeneous environment;

3. Mathematically characterize communication costs associated with pairing the different decomposition methods with various communication patterns; and

4. Propose a system that advises the programmer, compiler, user, or runtime system regarding the decomposition method expected to yield the best performance.

These goals have been achieved. The standard partitioning methods have been analyzed in the light of the particular needs of grid problems. Block decomposition techniques suitable for the heterogeneous networked environment have been developed that preserve rectangularity and generate fewer communications, in the worst case, than general quadrilateral or near-rectangular methods. The communication costs of the decomposition methods have been mathematically described based on communication pattern, the size of the problem and data items, processor speeds, and network characteristics. The prototype of a decomposition advisory system has been developed that uses information regarding the specific application program, the speeds of the machines, and the network speed and latency to produce a relative ranking of the decomposition methods expected to produce the best performance.

1.4 Significance

The workstation cluster is not a toy parallel architecture suitable only as an academic curiosity. Networks of workstations are currently being used for real applications with
varying degrees of success [1, 35, 41, 59, 99]. Parallel processing in the workstation network environment is most practical when the problem size is large and/or the number and size of communications is small in relation to the computational requirements.

The issues explored in this research provide methods for improving parallel processing performance in the heterogeneous workstation network. The analysis techniques developed permit better choices to be made by the programmer, compiler, user, or runtime system for data partitioning and load balancing.

The mathematical characterizations of the number and size of communications generated by the various data decomposition methods that I have developed are of value for task allocation decisions in any message-passing architecture where a cost is associated with interprocessor communications. I have established bounds on the conditions that must be met for particular partitioning methods to be practical, and these are useful for any parallel system.

The block decomposition methods I have developed can be applied in the homogeneous as well as the heterogeneous case. Therefore, these algorithms have broad applicability across both shared and distributed memory systems. Their particular usefulness, however, is in systems that accommodate heterogeneity, such as Data-parallel C and PVM.

The results of my research are significant, not only because they will enable better performance to be obtained for parallel processing on a workstation network but also because they will provide a mathematical basis for comparing data decomposition methods based on number of processors, problem size, and relative processor speed in any message-passing distributed environment.
1.5 Organization of the Thesis

In Chapter 2 previous work done in data decomposition, load balancing, and networked parallel processing is discussed. An overview of network technology is given in Chapter 3. The completely-connected graph model is compared to other organizations such as real parallel networks, where multiple connections at each machine may physically form a mesh or hypercube. The expected effects of new high-speed technology such as FDDI, HiPPI, and ATM are discussed. Chapter 4 surveys a collection of standard data partitioning methods for parallel processing in a distributed memory environment, and details the storage requirements for any lookup tables necessary to locate the host processor of an arbitrary data point. An algorithm for binary recursive block decomposition is given in Chapter 5 along with variants that exhibit better expected performance under certain circumstances. Mathematical analyses of communication time costs of the various decomposition methods when applied to a number of communication patterns are given in Chapter 6. Chapter 7 describes a data decomposition advisory system that utilizes the mathematical characterizations of communication cost established in the previous chapter, and describes the validation process used to empirically substantiate the accuracy of the model. Finally in Chapter 8 the contributions of this research are summarized and future directions are outlined. Figure 3 gives the notation we use for the analyses described in remainder of this thesis.
\( \lambda \)  
Message handling latency for each communication.

\( \tau \)  
Transmission time in seconds per byte.

\( C_{i,k} \)  
Number of communications from processor \( i \) in direction \( k \).

\( D_{i,j,k} \)  
Total number of bytes transmitted in a single message from processor \( i \) to processor \( j \) in direction \( k \).

\( d \)  
Length of the each data item to be transmitted (i.e., 4 bytes for integer).

\( E \)  
Eastbound communication transmission.

\( F \)  
Number of header and trailer bytes in the data transmission frame.

\( M \)  
Maximum data bytes in a transmission packet—the Maximum Transmission Unit (MTU) minus the header and trailer bytes.

\( m \)  
Number of rows in the grid.

\( N \)  
Northbound communication transmission.

\( n \)  
Number of columns in the grid.

\( p \)  
Number of processors.

\( S \)  
Southbound communication transmission.

\( t_i \)  
Computation time per iteration for processor \( i \).

\( vp \)  
Virtual processor (grid data point).

\( W \)  
Extraordinary wait time due to network contention and load imbalance.

\( W \)  
Westbound communication transmission.

Figure 3. Notation used to analyze communication time.
Chapter 2

Related Work

While the research described here focuses on the decomposition of data-parallel problems that may be represented by a uniform grid, there is a large body of related work in ancillary areas that is of value when considering load balancing issues. In this chapter we explore related work done for load balancing in distributed systems, task assignment for control-parallel problem spaces, and partitioning methods for data-parallel problems. In some cases the processors are homogeneous while the problem space is non-uniform or irregular. In other cases the processors may be heterogeneous while the problem space is uniform. The assignment or decomposition method may be static or dynamic. Static assignment allocates tasks to processors at compile-time. Dynamic assignment defers the assignment until run time. Some of the previous work permits process or data migration either at process creation time or during execution. When run-time migration is allowed, there must be policies in place to handle the acquisition of load information, to decide when to transfer, and to determine the recipient node [82]. Analyses of load balancing methods are cited, and previous work in distributed and parallel computing in the workstation network environment is mentioned.

The terminology used for data decomposition and load balancing is by no means standardized. Also, many methods use combinations of various techniques to improve performance. Consequently, there have been attempts to classify the many decomposition and load balancing schemes. Casavant and Kuhl [27] propose a taxonomy
of general-purpose load distribution systems. They provide both a hierarchical and a flat view of the issue. Under their hierarchical classification, methods are identified as being local or global, static or dynamic, optimal or suboptimal, exact or heuristic, and distributed or non-distributed (referring to the locus of decision-making). The techniques are further categorized according to the algorithmic approach used (i.e., exhaustive search, graph theoretic, mathematical programming, or queueing theoretic). The flat, or non-hierarchical classifications specify whether the scheme is adaptive or non-adaptive, and whether it has a one-time assignment or dynamic migration.

Other summaries and taxonomies are offered by Baumgartner and Wah [12], Kumar et al. [54], Wang and Morris [98], and Zhou [103]. Evaluations of load balancing success are presented in Rommel [78]. Evidence against the efficacy of process migration is given in Eager et al. [39], and an argument against load balancing in general is proposed by Wikstrom et al. [100].

2.1 Distributed Systems

Substantial work has been done in the area of load balancing for distributed, rather than parallel, systems. While parallel processing implies a coupling of some type between processes, distributed processing is more general in that the processes involved might not be related in any way, although they are permitted to be. Common goals for load balancing in distributed systems include efficient resource management, acceptable throughput, and high processor utilization.

Static methods of load distribution usually assume that the job arrival rates, processing requirements of the jobs, and the capabilities of the system are all known a priori. Work entering the system is routed according to an algorithm that is often based on graph theoretic, queueing theoretic, mathematical programming, probabilistic, or heuristic methods. Stankovic [88] proposes a scheduling scheme based
on heuristics and Bayesian decision theory, with validation of the model is provided through simulation results. Lo [64] presents a task assignment algorithm based on Stone’s network flow scheduling technique [91]. Ni and Hwang [68] provide an optimal probabilistic policy and Gao [44] proposes a queueing theoretic method. While static load balancing techniques are easier to implement [38], they do not adequately accommodate the situation found in the real environment of most distributed systems. Baumgartner and Wah [13] mention results that indicate that, even though dynamic load balancing strategies incur higher overhead, they can yield superior results.

In the remainder of this section, some of the more prominent dynamic distributed load balancing techniques are summarized. In some cases single assignment of a process to a node is the only permissible allocation. In others a process may move a number of times before beginning execution or may migrate during execution. Process location policies are often categorized as sender-initiated (overloaded nodes attempt to find lightly loaded processors to whom work may be sent), receiver-initiated (underloaded nodes try to find other processors to provide work), and symmetrically-initiated distribution schemes (both overloaded and underloaded nodes try to find recipients and providers of workload) [86]. Sender-initiated policies are also referred to as source-initiated, and receiver-initiated policies as server-initiated by some researchers.

Lin and Raghavendra [61] propose a dynamic load balancing scheme (LBC) that uses as central job dispatcher. Global state information and a single-queue multiserver model are used to improve average response time and maximize processor utilization. The system assumed for this method is comprised of a collection of homogeneous nodes connected by a network. It is also assumed that at each node there is an additional microprocessor whose sole responsibility it is to handle the overhead of load information gathering and load balancing. Therefore computation, communication,
and load balancing can occur concurrently. This is a globally-controlled version of a symmetrically-initiated algorithm since a node may act as either a sender or receiver.

An adaptive bidding algorithm that migrates processes during execution is presented by Stankovic and Sidhu [89]. This is a sender-initiated scheme in which an overloaded processor transmits a bid request to the other processors. This includes information regarding the estimated resource requirements of the process to be moved, its current state (ready, running or blocked), priority, and number of times moved. Candidate recipients assess the needs of the migrating process in light of their current capabilities. A bid that represents that processor's desire to host the migrant process is returned to the originating processor. The process is migrated to the node that returns the best bid.

The drafting algorithm proposed by Ni et al. [69] assumes a networked distributed system and uses a queueing theoretic approach, as does the LBC policy. This is a receiver-initiated technique; that is, lightly loaded processors send a draft message to processors believed to have excess workload. Each node maintains a table of load levels at the candidate processors (those that have jobs that may be migrated). Loads are expressed as heavy (H), light (L), or normal (N). This table may be somewhat out of date due to communication delay and the reluctance of the algorithm to immediately broadcast changes from the H to the N state. It is felt that this reduces state woggling, where slight changes in load may cause the state to flip-flop back and forth, thereby causing instability. The drafting algorithm is a single-migration dynamic load-balancing method for homogeneous distributed systems.

Stability is the main focus of the algorithm developed by Bryant and Finkel [24]. Some techniques result in processor thrashing, where jobs fruitlessly migrate around the system without being executed. Migration is constrained to follow the point-to-point lines in the distributed system, and temporary monogamous pairings between neighboring processors are instituted. A processor may be in one of three states:
idle, pairing, or migration. In the pairing state, an overloaded processors attempts to locate a willing workload recipient. Once a pair is established, the migration state is entered. The overloaded processor sends a list of jobs to be transferred, followed by the jobs themselves. Simulation results indicate reasonable performance as well as stability.

Three sender-initiated load sharing algorithms are examined by Eager et al. [38]. The random policy transfers jobs from overloaded processors to randomly chosen recipients. The receiving processors may, in turn, transfer the tasks to yet other nodes. A system limit on the total number of allowable transfers prevent the pathological case where a job might travel endlessly. The threshold policy probes a randomly chosen node to find out if it is willing to accept a transferred task. If so, the job is moved. Otherwise, additional nodes are polled until a system-imposed poll-limit is reached. In the shortest policy, a number of nodes are polled at once. The one that admits to the lightest load is chosen.

A clustering algorithm that heuristically groups communicating processes so they may be assigned to the same processor is proposed by Bowen et al. [23]. Their method is agglomerative rather than divisive in that it begins with individual processes and combines them into clusters instead of starting with a collection of all the processes and dividing it into clusters. They are concerned with load balance as well as communication cost, and upper and lower bounds are established for acceptable processor loads. They express the allocation of processes to processors as a quadratic assignment problem. This differs from the linear assignment problem in that the entities being assigned are interconnected (by communication requirements) rather than being independent objects.

Migration is particularly common in receiver-initiated and symmetrically-initiated distribution schemes [86]. While these schemes have the potential to achieve more accurate load balance, they incur the additional overhead of having to transfer the com-
plete state of the process. Shivaratri and Krueger [83] describe symmetrically-initiated and sender-initiated global scheduling algorithms. The symmetrically-initiated algorithm allows for process migration during execution. Eager et al. [39] give evidence that there are no conditions under which this type of migration will yield substantially better performance than good initial-placement schemes.

Chu et al. [34] address the saturation effect that occurs when the number of processors reaches a threshold beyond which communication cost overshadows the savings in computational time. They rely on graph-theoretic and integer programming approaches to the problem. Because they include the cost of intermodule communication, their work also applies to control-parallel processing.

Load balancing of processes in an Ethernet-connected environment is studied by Schaar et al. [80]. They use analysis and simulation to quantify the effect of communication delay in networked situations. Sender-initiated and receiver-initiated methods are considered.

Other load sharing and balancing schemes for distributed systems that permit process migration after the process has begun execution include Leland and Ott [58], Ma et al. [65], Stone and Bokhari [92], and Suen and Wong [94].

2.2 Control-Parallel Processing

As in distributed scheduling, a number of methods may be utilized alone or in combination when trying to apportion the tasks of a control-parallel problem across the participating processors. Graph theoretic, queueing theoretic, mathematical programming, and heuristic methods are just some of the techniques used.

Chaudhary and Aggarwal [32] propose a method that combines graph theoretic, mathematical programming and heuristics to provide a generalized scheme for mapping parallel algorithms onto multicomputers. A graphical representation of the algorithm is mapped to a graph that depicts the topology of the parallel machine.
by means of an extended host graph. Objective functions are used to estimate the optimality of the mapping. Interprocessor communication is taken into consideration in this static scheme. Both simulation and actual testing have confirmed the value of this technique.

The gradient model of Lin and Keller [60] is a receiver-initiated dynamic method of load balancing on multicomputers. It is assumed that a program spawns additional tasks, which in turn spawn even more tasks. These tasks must be spread throughout the system to avoid overload at the parent node. Each node interacts only with its physically immediate neighbors, balancing the load only within its own neighborhood. Global balance occurs through propagation, as the neighbors of a given node interact with their neighbors, and so on. Load balancing begins when an under-utilized node requests work from its surrounding nodes. Interprocessor communication is not considered, but the gradient model may be enhanced to accommodate heterogeneous processors.

Kalé's scheme for dynamic, distributed load balancing for control parallel processing called Contracting Within Neighborhood (CWN) [52] relies on local information to determine the system load. Each processor piggy-backs its load information onto regular communication packets with neighboring processors. When processes are created, a goal message is immediately sent to the least loaded neighbor. Each goal message has a hop count that keeps track of the number of hops it has traveled from its source. When the hop count equals the allowed radius of the neighborhood, the receiving processor accepts the newly created process. If, however, a processor's neighbors are all more heavily loaded than itself and the message has traveled at least a minimum number of hops, the current processor accepts the task. Once the process is accepted, it is no longer eligible for migration. While unnecessary message hops may occur, this method has been reported to consistently perform better than the gradient model of load balancing.
The CWN technique has been improved by Shu and Kalé [84] by varying the minimum and maximum number of hops that a goal message must migrate before acceptance to reflect the system state. It also redistributes processes to correct load imbalance by moving processes that have already been migrated and accepted. This *Adaptive Contracting Within Neighborhood* (ACWN) method is not only dynamic, but also adaptive, and performs generally better than CWN.

Saletore [79] proposes a neighborhood averaging strategy that builds on ACWN. Neighborhood averaging concentrates on when, how, and how quickly to balance loads. If the decision is made to load balance, a sender-initiated method is used. The individual processors calculate the average load of the neighborhood. If a processor's load is higher than the average, it seeks to off-load work. Neighborhood averaging performs better than both CWN and ACWN.

To avoid the difficulties that arise in purely localized or centralized load balancing schemes, Ahmad and Ghafoor [3] have developed a semi-distributed policy for massively parallel computer systems. The term *massively parallel* generally refers to any parallel computer with more than 1,000 nodes [3]. The system is divided into a collection of spheres, each of which contains a single scheduler whose job it is to balance the load within that sphere. The scheduler also keeps track of load information and handles load exchanges with other spheres. This strategy is equally applicable to distributed systems.

Bokhari [21] uses Stone's network flow algorithm [91] to offer a *sum-bottleneck path algorithm* for efficiently assigning the modules of a control-parallel problem to the processors of a multicomputer. This scheme includes an optimal solution for a number of single-host, multiple-satellite system problems, even though the general case is known to be equivalent to either the NP-complete graph partitioning problem or the multiprocessor scheduling problem [21].
The heuristic MCA (module clustering algorithm) and MRA (module reassignment algorithm) of Efe [40] assigns the tasks of a control-parallel problem to a collection of homogeneous processors in a way that minimizes interprocessor communication and improves load balance. Since these two goals are often in conflict, the algorithm begins with an assignment that results in the least communication using the MCA algorithm, then iterates using the MRA algorithm until the solution converges to a balanced load. Load balance is gauged by queue length and this method is suitable for both parallel and distributed processing.

*Gang scheduling* is a method for requiring a set of *threads* to be executed simultaneously across a number of processors. In this scheme Feitelson and Rudolph [42] describe a system that coordinates context switching across multiple processors to assure that sets of tasks (a *gang*) all execute at the same time. This requires that the number of tasks (threads) in a gang be less than or equal to the number of available processors, and hardware support is needed to give the interactions sufficient speed. The rationale behind gang scheduling is that there exist tasks that have sufficient interaction so that, if they do not run at the same time, excess wait time is incurred by processors who reach the synchronization point first. Instead of blocking at an interaction point, busy waiting may be used instead, since it is known that the other participant to the communication is also currently running. This scheduling technique has been designed for, and tested on, a shared memory machine. However, its authors purport that it is equally useful on distributed memory architectures.

In the SOS (Synthesis Of Systems) approach of Prakash and Parker [74] Mixed Integer-Linear Programming (MILP) is used to address scheduling in a heterogeneous multiprocessor system. They consider a system that contains different types of processors, different processing components, and different connectivities. Their environment could include vector processors, SIMD processors, MIMD machines, special purpose computers, and data-flow machines. SOS specifically addresses heterogeneity
that arises from differing functionality among the processors and differing processor speeds.

A compile-time heuristic scheduling technique, *dynamic level scheduling*, is proposed by Sih and Lee [85]. Interprocessor communication cost is seen as the major impediment to parallel processing performance, and they assume the presence of dedicated hardware that permits the overlap of communication and computation. The environment this system addresses is heterogeneous and may have an irregular connection structure. The goal is to minimize the *makespan*, or schedule length, including all the overhead incurred by interprocessor communication. Since this problem is NP-complete, they rely on heuristics.

Markatos and LeBlanc [66] deal with load balancing in a shared memory multiprocessor environment. They determine that locality is more important than load balance. That is, given a choice between good balance of load on the processors and locating a process close to its data, performance is better in the latter situation. They consider three assignment policies: Load Balancing, Memory-Conscious Scheduling no Migration, and Memory-Conscious Scheduling. The first examines performance under good load balance only, the second considers performance with locality of data only, and the third assumes both. Results using simulation indicate that performance is best when locality is first maximized followed by load balancing considerations.

A probabilistic method for scheduling *task forces*, called *wave scheduling*, is proposed by Van Tilborg and Wittie [97]. A task force is the collection of cooperating tasks that constitute a control-parallel program. The method is static: that is, it is assumed that all the tasks are known prior to scheduling. A hierarchical virtual machine is envisioned, regardless of the actual underlying machine topology, so that the root and interior nodes act as managers (task force masters). The leaves of the tree actually perform the work of the tasks. The task force masters are responsible for making sure that enough nodes are available to handle the workload assigned to
the subtree. When a task force enters the hierarchy, it is moved down or up the tree, depending on whether it can or cannot be handled by the current task master. Evaluating the performance of this task scheduling method is by mathematical analysis only.

Various other load balancing schemes for control-parallel processing are offered by Lo [63] and Willebeek-LeMair and Reeves [101].

2.3 Data-Parallel Processing

The partitioning of data-parallel or SIMD problems for distributed or multicomputer systems has recently generated interest. These architectures have more general applicability than processor arrays and are widely available. Queueing theoretic approaches are not deemed to be suitable, since in this computational paradigm the concern is with the assignment of data points, not entire processes. Graph theoretic methods also do not apply since all processes are assumed to be running synchronously.

Partitioning may be done statically or dynamically. When data points are dynamically assigned, the current state of the system may be considered when partitioning the problem space. Usually there is a single mapping of grid points to processors, but schemes for data point migration during execution have been proposed.

The problem of statically mapping SIMD problems onto processor arrays is examined by Bokhari [20]. Partitioning is accomplished by a combination of heuristic and probabilistic decisions. The efficiency of the method has been demonstrated on the finite element machine (FEM) at NASA Langley Research Center.

A static method for problem space decomposition for data-parallel applications is presented by Gupta and Banerjee [47]. A multicomputer of homogeneous processors is the target system. The critical issue they address is predicting communication costs that will be incurred. They estimate communication time to be a function of the array sizes and the number of participating processors. It is their belief that
partitioning is the most challenging issue in effectively exploiting the parallelism in an application.

Snyder and Socha [87] propose a static method for mapping a two-dimensional grid of size $I \times J$ to a $K \times K$ array of processors in a way that maintains load balance, near-rectangularity, and bulk. They assume that the processors are homogeneous and the problem space is uniform with updated values relying on the values from a subset of nearest neighbors. A problem space represented as a uniform grid implies that all data points have exactly the same operations performed on them. In this domain with homogeneous processors, load balance demands that all processors be assigned an equal number of data points. Since this is not possible, given the grid and processor array dimensions, the partitioning algorithm minimizes the necessary load imbalance. The resulting partitioning is near-rectangular in that a maximum of two jogs is permitted along each boundary (a jog is a deviation from a straight line). The decomposition is bulky in that has a near-maximum ratio of interior to exterior data points. The goal here is to minimize the number of data points on the boundaries, thereby minimizing the amount of data that must be exchanged between processors. An example of this partitioning is given in Figure 4.
An interactive tool to help the programmer partition a uniform data-parallel problem space across homogeneous processors is proposed by Balasundaram et al. [9]. The programmer specifies the desired decomposition of the data structures and the tool provides feedback on the expected performance of the partitioning. This allows experimentation with various decompositions to find the one that is expected to yield the best response time.

Automatic data partitioning by a parallelizing compiler is suggested by Gupta and Banerjee [48]. A constraint-based approach is used in which the compiler analyzes each loop and identifies constraints on the distribution of the data. The compiler assigns quality measures with each constraint and finally combines the constraints for each data structure to minimize overall execution time. Two kinds of constraints are used: parallelization and communication. It is assumed that all loop bounds and the probabilities of executing conditional branches are known to the compiler. This information must be supplied by the programmer.

Automatic array alignment for either distributed memory or non-uniform memory access (NUMA) shared memory machines is addressed by Chatterjee et al. in [29]. They identify three classes of communication: nearest-neighbor, tree (e.g., reduction), and unstructured (e.g., transpose). Communications are further classified as intrinsic (arising from operations that require data movement as part of the operation) or residual (the communication is necessitated by non-local data references). This work is extended in [30] where the Alignment-Distribution Graph (ADG) is described. This graph is used to model residual communication. This tool is used again in [31] where algorithms are presented to determine good mobile alignments. Mobile alignments are those that may be a function of the loop variable.

Nicol and Saltz [70] examine grid problems with varying resource demands. That is, the computational work associated with each grid point may change during execution of the program. Their approach permits remapping, or data point migration
during execution as the computational needs of the grid points change. A stop at rise (SAR) policy is used in which the data is remapped whenever a local minimum in a predetermined degradation function is detected. Simulations are used to show that SAR consistently improves performance over the option of not remapping. Nicol and Reynolds [72] identify a large class of problems characterized by a sequence of phases, each of which requires a different data decomposition for efficient performance. Issues considered are identifying a phase shift, deciding when to remap, and the most promising partitioning to use.

Partitioning non-uniform or irregular grids across homogeneous processors has been considered by a number of sources. Berger and Bokhari [17] propose a binary decomposition where the number of processors is a power-of-2. Their method preserves rectangularity, thereby avoiding the need for large data structures to keep track of the home processor of each data point. While they directly address only static partitioning where there is a priori knowledge regarding the problem domain, they mention that dynamic rebalancing can be done without completely repartitioning the grid. It is this work that inspired the Binary Recursive Block Decomposition algorithm for heterogeneous processors that is presented in Chapter 5. Bokhari et al. [22] have recently proposed a parametric binary dissection method for partitioning non-uniform domains across homogeneous processors. The main consideration is the trade-off of load imbalance with communication costs incurred by the shape used to correctly apportion the load.

Stencil communication patterns and partitionings for grid problems are examined by Reed et al. [77]. They show that nonstandard partitions (e.g., hexagons) may yield lower communication cost than usual square partitions. They formalize a relationship between shape, stencil structure, and machine architecture.

Nicol [71] examines the decomposition of irregular grids into perfectly rectilinear partitions to facilitate their mapping to mesh networks. The partitioning is static
(i.e., done at compile time) and interprocessor communication cost is considered. New algorithms for one and two-dimensional problem spaces are proposed, and it is shown that the three-dimensional grid partitioning problem is NP-complete.

The N4-partitioning strategy of Hinz [51] distorts the block decomposition of a problem space to accommodate irregular or non-uniform grids. He cites the work of Baden [8] to support the thesis that load imbalance, rather than the partitioning strategy or communication cost, is responsible for the majority of the incurred overhead. This is contrary to prevailing opinion, which holds that communication costs may be the single most important factor that limits parallel performance in both multicomputers and multiprocessors [17, 19, 20, 33, 48, 49, 63, 92]. While the N4 general quadrilateral affords a finer-grained load balance than those methods that preserve rectangularity, a greater number of communications between processors is generated in the worst case. This partitioning method is examined further in Chapter 5.

Belkhale and Banerjee [15] propose two dynamic recursive partitioning schemes for a non-homogeneous grid problem space. The slice partitioning technique divides the domain into vertical slices so that each section has approximately the same number of points. The rectangular method first divides the domain across the x direction into slices with about the same number of grid points, then across the y dimension into regions containing approximately the same number of points. Points may be added to or deleted from the problem space during execution. The algorithms have been demonstrated on a hypercube architecture.

Additional exploration of unstructured grids has been performed by Williams [102].

The probability of load balancing success (PLBS) in a homogeneous distributed or multicomputer environment is addressed by Rommel [78]. The mechanics of load balancing may actually reduce performance. The cost of local processing to determine when and with whom to balance, the possibility of processor thrashing, and delays
induced by network overhead may increase response time more than accurate load balance may save.

2.4 Analysis of Load Balancing Systems

Since load balance is of critical importance to the performance of parallel and distributed systems, numerous comparisons and analyses have been published on the various techniques. Kumar et al. [54] investigate scalable load balancing techniques for parallel processing. Their analysis includes the impact of load balancing on hypercube, mesh, and homogeneous workstation cluster topologies for independent tasks. The scalability analysis presented here may be used to determine the efficacy of using additional workstations for solving a particular problem. However, they do not consider interprocessor communication.

Leland and Ott [58] attempt to quantify the benefits attainable from initial placement and migratory techniques in loosely-coupled homogeneous systems. Their findings indicate that judicial initial placement alone or process migration alone provides better performance than no attempts at load balancing. If initial placement is combined with migration, a greater benefit is realized.

A performance evaluation approach is presented by Ahmad et al. [2]. They use a combination of simulation, statistics, and analytical modeling, and take into account a large number of system parameters that can affect performance. Their unified approach allows a number of load balancing strategies to be compared on common ground. They analyze a collection of sender-initiated schemes.

Zhou [103] simulates seven different dynamic load balancing algorithms to examine their behavior. In (1) global balancing the load information of each node is periodically sent to a designated host who then broadcasts the load vector to all participants. (2) Distributed balancing permits each node to broadcast its status to the others directly. In these two schemes, the load balancing decisions are done locally.
The (3) central algorithm serves as a central scheduler and directs the load balancing action. (4) Random load balancing uses only local load information and sends excess work to a randomly selected node. With (5) threshold policies a probe limit is used that establishes the maximum number of nodes that will be polled when attempting to find a willing workload recipient. Algorithms that use a (6) lowest load algorithm poll a set number of processors and send the excess work to the one with the smallest workload. The (7) reserve algorithm is a receiver-initiated method. When a job completes, or leaves for any reason, the node checks its own load situation. If its load is below some threshold, it makes a reservation at more highly loaded nodes for future work. When one of the overloaded nodes receives another job, it pops its stack of reservations and sends it to the node with the most recent reservation. The environment assumed for this analysis is a homogeneous distributed system. The findings include evidence for improved performance due to load balancing over a wide range of conditions. The results indicate that initial placement algorithms perform poorer in receiver-initiated systems, large jobs benefit the most from load balancing, while small jobs do not suffer, and load balancing has a beneficial effect on the performance of every node.

2.5 Workstation Clusters

Computer clusters, especially networks of workstations, have become increasingly important as both distributed and parallel systems. Distributed processing is easily implemented on workstation clusters with the aid of distributed operating systems and standard network protocols. In this environment processes may be created by a user at one node, but actually executed on another, more lightly loaded processor. In most cases work is assigned to remote processors (nodes other than the one that originated the process) only as long as the registered owner of the remote machine is not currently using the machine.
The *Stealth distributed scheduler* of Krueger and Chawla [53] departs from traditional workstation schedulers in that *foreign* processes (those processes not owned by the owner of a workstation) are not automatically pre-empted when the owner is using the machine. Rather, a priority resource allocation is used so that the workstation owner's processes get the CPU time they would expect from a dedicated machine, with the foreign process using otherwise useless cycles. The testbed used to gather empirical results was a network of 199 Sun workstations and 18 Sun fileservers. Files were NFS mounted and the 4.3BSD UNIX operating system supporting only remote procedure call (RPC) and remote shell (rsh) was used. They found that the average unused workstation capacity was 91%, and they tout Stealth as a promising method for utilizing a large portion of this unused power.

Barak and Shiloh [11] present a migratory load balancing policy for a network of independent workstations. They claim that their method is stable: that is, it inhibits *processor thrashing*. They assume a completely connected collection of homogeneous nodes and complete ignorance regarding the expected execution time of the processes involved. A probabilistic load exchange algorithm is used so that each node has information regarding the load status of some subset of the processors which is sufficient to determine when load balancing is necessary and the identities of the nodes that need to engage in balancing.

For distributed processing in a homogeneous workstation network with a network file system (NFS), Kunz [55] showed that the best single workload description is the number of tasks in the run queue. This, of course, ignores the possibility of interprocessor communication and widely varying execution times among the processes.

Cheung and Reeves [33] examine some of the issues peculiar to parallel computing on a heterogeneous workstation cluster. They consider the main concerns in this environment to be network latency, processor selection, and load balancing. A performance estimation method and performance metric are proposed to address
the processor selection and load balancing issues. Because of the high costs associated with network latency, they contend that the network cluster is suitable only for problems with statically predictable communication patterns. These communications may be prescheduled. A data prefetch option should be available to take advantage of prescheduling and to help offset the message preparation overhead. Their performance prediction scheme relies on a computational equivalence metric, which is the cost, in terms of arithmetic operations, of executing a parallel operation on some data structure. Using this metric, the execution time of an entire algorithm may be estimated for a given architecture. Their load balancing scheme takes the heterogeneous nature of the cluster machines into account, but not the multiuser system load. They propose static partitioning only, and the sole decomposition pattern available is a contiguous rectangular block assignment.

Excessive parallelism is the concern of Pekergin [73]. Empirical results show that, while transmission time may safely be ignored for interprocessor communication in a token ring environment, message handling latency is the critical overhead factor. The aim is to estimate speedup and the optimum degree of parallelism allowable for a given algorithm. Their model assumes a MIMD problem space and round-robin scheduling of tasks. While the model used is overly simple, it is expected to be expandable and adaptable to become more useful in determining the value of exploiting all the available parallelism in an application.

Atallah et al. [7] use co-scheduling or gang scheduling in conjunction with gleaning idle cycles on a network of workstations to solve computationally intensive tasks in parallel. They consider interprocessor communication overheads and assume a data-parallel problem space. Their work focuses on homogeneous processors, but they extrapolate to the heterogeneous environment. Their main concern is effective resource management in a distributed computing environment.
The Dataparallel C programming language and environment for parallel processing on a cluster of heterogeneous workstations has been presented by Nedeljković and Quinn [67]. This permits parallel programming in a relatively high-level language (a variant of C) while making all communications transparent to the programmer. They include a scheme for dynamic load balancing with data migration. They assume a data-parallel problem space and allow the programmer to choose the decomposition method that best suits the particular application. At communication points the participating machines exchange load information and, when it becomes apparent that a load imbalance exists that is severe enough to justify the overhead incurred by migration, data points are *rippled* through the network. Figure 5 shows an example of data movement in this system.
The PARFORM [26] is specifically targeted for the high-performance workstation environment with Ethernet connections. Message passing is handled using TCP and Berkeley sockets. It provides task placement according to actual current load of the workstations. Provisions for dynamic load balancing are included in the PARFORM, and UNIX kernel variables are scanned periodically to check the load situation on the various machines. A heuristic is used to optimize the size and number of subtasks to be performed given the current network load, but the possible variance in the computational speeds of the machines is not considered.

PVM (Parallel Virtual Machine) is not specifically targeted for the workstation environment, but lends itself to a number of different architectures and networks [14, 45]. PVM is a message passing extension to C or FORTRAN, and has no load balancing facilities. The system is completely ignorant of the relative speeds and current workloads of the available machines. The user, however, may request specific machines or architectures [46]. Since communication is by explicit message-passing written into the code, the programmer is charged with the responsibility of data partitioning or task allocation. PVM may be used as a vehicle for either SIMD or MIMD programming. While PVM provides a great deal of flexibility, it leaves the low-level problem space decomposition work to the programmer.

While prior work addresses some of the issues that arise in the heterogeneous parallel network, none does so adequately. The particular sensitivity of this environment to network speed limitations and message preparation latency increases the necessity for scrupulous data partitioning. Previous exploration of data decomposition considers homogeneous processors rather than a heterogeneous, multi-user environment. Therefore, load balancing for differing processor speeds is ignored. The existing models of networked parallel computing do not consider the issue of determining the most promising data decomposition techniques. They assume either a single decomposition method [33], or rely on directives of the programmer [67]. The issue of partitioning
scheme selection, in conjunction with load balancing, for the parallel network is examined and resolved in this thesis.
Chapter 3

Network Considerations

Network technology has consistently lagged behind the advancements made in microprocessor speeds. The emerging interest in connectivity for the Information Superhighway and the national High Performance Communication and Computation (HPCC) initiative has lent impetus to improving the speeds of network connections and the protocols necessary for packetizing, message routing, and addressing.

In this chapter a brief overview of some traditional and emerging network technologies is given. Ethernet, Token Ring, FDDI, HiPPI, and ATM are described. Multiple network connections are also explored. These Multi-Net configurations offer the possibility of less bandwidth contention than is found in a single network situation.

3.1 Ethernet

Ethernet networks have been extremely popular over the years. First introduced by Xerox PARC in 1976, Ethernet connections offer a transfer rate of 10 Mbps [95]. The term Ethernet is often misused to indicate the Carrier Sense, Multiple Access with Collision Resolution (CSMA/CD) network protocol that is typically used with this medium [96]. Strictly speaking, Ethernet refers to the ether or cable that is used for the connection. The cable has two forms: thick Ethernet and thin Ethernet. The thick-net has tap points every 2.5 meters and thin-net uses T-junctions rather than taps. Thin-net, however, can only run for short distances.
With the CSMA/CD protocol, all nodes on the network monitor the cable for a *carrier* (i.e., a transmission), and when a node wishes to transmit a message, it listens to determine if another transmission is already on the wire. If so, it aborts, waits a random period of time, and tries again. If no other transmission is detected, it begins sending the message while still monitoring the cable. If the message it hears is something other than the one it sent, a collision has occurred. In this event transmission is halted, and a *binary exponential backoff* heuristic is used to determine the amount of time each node should wait before attempting retransmission. If all transmitters of the colliding messages waited the same amount of time, they would retransmit at the same time again, thereby producing another collision. Each of the colliding nodes sets a local variable, waits that amount of time, and the value is doubled for each recurring collision.

Ethernet is one of the most popular connections found in LANs. It is inexpensive and numerous transport protocols are compatible with this technology.

The analysis described in the following chapters assumes an Ethernet technology with a CSMA/CD network protocol. This is, however, one of the most difficult network configurations for which to predict transmission time because of contention and collision resolution activity [69].

### 3.2 Token Ring

The oldest and most popular of the ring technologies is the Token Ring that was first defined in 1972 [95]. To handle the problem of contention for the network, a *token* (typically an 8-bit pattern) is circulated around the ring when no transmissions are on the wire.

If a node wants to send a message, it must first seize the token and remove it from the ring. To do this, the node inverts the last bit in the pattern, which changes it into a *connector*. The message to be transmitted is placed on the network immediately
after the connector. This effectively eliminates collisions. When the message has propagated around the ring, the sender removes it and replaces the token.

A typical speed for the Token Ring is 16 Mbps. Some studies have shown that token-ring transmission rates consistently remain above 70% or 80% of peak even with 100 workstations all communicating with each other with packet sizes of up to 2000 bits. In the token ring environment, the high throughput indicates that message handling latency, not transmission time, is the critical factor in communication overhead [73].

3.3 Fiber Distributed Data Interface (FDDI)

*Fiber Distributed Data Interface* (FDDI) offers higher transmission rates than were available in traditional Ethernet or Token Ring networks. FDDI is currently defined to transfer data at 100 Mbps over distances up to 200 km. FDDI is usually set up as a token ring LAN with two fiber rings: one transmitting clockwise and the other transmitting counterclockwise. Nodes may connect to one or both rings, with the latter being the more expensive configuration [96]. While fiber is the usual medium, the definition of FDDI may be extended to include copper, since these speeds can be accommodated by copper technology [5].

Discussion is underway to scale up FDDI to be a Gbps network and FDDI-2 provides support for *isochronous data* (data that must be delivered in regular amounts at regular time intervals) [5].

FDDI is attractive for LAN connections because of both its high speed transfer rate (relative to Ethernet and Token Ring) and its declining cost.

3.4 High Performance Parallel Interface (HiPPI)

The *High Performance Parallel Interface* (HiPPI) protocol was originally defined for peripheral interfaces, but is now being used for high data-transfer rates in LANs.
HiPPI is an I/O channel capable of transmitting 800 Mbps with a bus width of 32 bits, and 1.6 Gbps with a 64 bit bus width. While it was originally intended as a point-to-point channel between the CPU and its peripherals, it is now commonly used between CPUs.

The HiPPI protocol is concerned mainly with the physical layer, leaving transport considerations unaddressed. Some feel that existing protocols can be modified to fit the need. Others believe that new, lightweight protocols that can be realized in silicon will be necessary to avoid the software bottleneck. Lightweight protocols have fewer protocol layers than the standard ISO/OSI seven layer model. They include multicasting capability and support for fixed header/trailer encapsulations [81]. One such candidate protocol is the Xpress Transfer Protocol (XTP). While this has been designed to be an on-chip protocol, it has, so far, been utilized only in software [93].

Fiber optic versions of HiPPI are emerging, but it was originally meant to utilize large bundles of cooper wires over a short 25 meter run. Recently HiPPI extenders have become available to avoid the distance limitation. Distances up to 15 km may be available if standard parallel HiPPI signals are converted to serial fiber optic signals [6].

HiPPI is still relatively expensive, and therefore in limited use for LAN connections.

3.5 Asynchronous Transfer Mode (ATM)

Asynchronous Transfer Mode (ATM) is a packetizing and switching protocol that provides a common format for both the bursty high-speed transmissions of data communications and typical telephone traffic [56]. The packet specified by ATM is called a cell and it contains 53 bytes: 5 bytes for the header, and 48 bytes for data. While this incurs additional overhead for pure data transmissions, it enables the same technology to be used for data, voice, and video service.
While some of the other communications technologies specify transmission rates, ATM is scalable. The standard defines only the 53 byte cell, not the transmission speed. Therefore, ATM is usable on networks with vastly different speeds. Virtual networking is used to route cells between two ATM switches. ATM is connection oriented in that every cell in a transmission travels over the same route.

In an ATM LAN, a centralized switch has a dedicated connection for each node. When a message is sent to the switch, it is routed to the destination indicated by the header. Each node has exclusive access to its port in the switch, thereby reducing contention considerations that occur when a shared medium is used as in Ethernet, Token Ring, or FDDI.

ATM is still an emerging technology. However, it is already been popularized by its promise of providing a common technique for transmitting a full range of multimedia information.

3.6 Switched Networks

The new high speed switch technologies present a topology that may almost be considered to be a completely connected network. In a completely connected network $p/2$ communications can occur simultaneously. The ATM switch mentioned previously is an example of this technology. Each node has a dedicated connection to the switch, which provides high speed routing of messages to the other nodes (see Figure 6).

The problems of network contention will be greatly reduced, although not entirely eradicated. Contention will still occur when multiple nodes attempt to communicate with a single node or when multiple users generate non-related network traffic. Even without completely eliminating this problem, the reduction in traffic and the high speed of the switch are expected to provide performance close to that of a completely connected network. The overhead of message preparation latency still remains, however, and must be addressed by advances in transport protocol technology.
3.7 Multi-Net Configurations

Multiple network connections can increase message throughput by reducing network contention. Typically the routing protocol is implemented in software rather than hardware, and network selection can be established either statically or dynamically. In static routing, a table is established that indicates the network to use between two specific host machines. In dynamic routing, daemon processes broadcast their availability to act as routers to specific hosts [95].

There are limits on the number of network connections a host machine can accommodate. That limit on older Sun workstations, for example, is 3. Other machines may be able to handle more. If we assume a sufficient number of connections and a routing protocol that permits flexible path selection, multiple messages can be transmitted concurrently, thereby increasing throughput.
Figure 7. Multiple network connections. (a) A ring with two networks. (b) A ring of rings.

We will consider two separate situations where each workstation is connected to multiple networks:

1. All workstations are connected to the same multiple networks, and

2. Workstations may be connected to different multiple networks.

Figure 7(a) shows the first situation, and Figure 7(b) gives an example of the second.

Network contention is an important issue in any message-passing system. It is particularly critical in a synchronous environment when a number of cooperating processes concurrently try to transmit data over the physical medium. When a single network is used, only one process is usually the bus master. That is, only one message can be transmitted over the physical medium at any given time. This causes processors to idle while waiting for the resource.

The availability of multiple networks allows concurrent transmission, which increases throughput by decreasing time spent waiting for network resources. The extent of concurrency depends on the routing protocol of the system. If, for example, $k$ paths are available for transmitting $p$ messages of $d$ bytes each, then instead of a transmission time of $pd\tau$, the concurrent transmission time is $(pd\tau)/k$, where $\tau$ is the bandwidth expressed as seconds per byte.
Now consider the situation where the machines are connected to different networks. Figure 7(b) shows one such topology, a ring of rings. But the possibilities are legion. This situation is not uncommon. A parallel program running on the Computer Science research Suns and the Oceanography IBM RS/6000s executes across two separate networks, and the internetworking is transparent to the user. In fact, Metropolitan Area Networks (MANs) and Wide Area Networks (WANs) are all constructed as networks of subnetworks.

While a single virtual network is formed when executing across multiple networks, each sub-network can be considered to be a sphere and one machine on each sub-network may be designated as the central control point for that sphere [4]. If the problem space can be decomposed in a way that limits most communications and load balancing to the local sphere, the additional cost associated with gateway crossings can be avoided, and communication may proceed concurrently for each subnetwork.

The linear, ring, and spine topologies are some of the most common single network LAN configurations [95]. These, however, are not designed for high performance parallel computing. While any topology can be simulated on these configurations through the use of a communication mechanism like the socket interface, actual transmission time is influenced by contention for the single physical network.

With the availability of multiple networks, machines can be connected into some of the more traditional multicomputer topologies such as hypercube or mesh. In the following discussion the additional cost of crossing gateways is ignored, but it is important to realize that some additional communication time is incurred when moving between networks.

A \( k \)-dimensional hypercube topology can be constructed from \( 2^k \) workstations. Figure 8 shows this configuration with 8 machines using 6 networks and 3 connec-
Figure 8. A 3-dimensional hypercube formed from 8 workstations and 6 networks. The 6 networks are \((0,1,3,2), (0,1,5,4), (0,2,6,4), (1,3,7,5), (2,3,7,6), (4,5,7,6)\).

tions on each machine. A broadcast of \(d\) bytes from one to all on the hypercube configuration can be done in time

\[(\lambda + d\tau)\log_2 p\]  \hspace{1cm} (3.1)

instead of the single-network time of

\[(p - 1)(\lambda + d\tau)\]  \hspace{1cm} (3.2)

where \((p \geq 8)\) is the number of processors, \(\lambda\) is the message passing latency, and \(\tau\) is the bandwidth expressed as seconds per byte. Similar savings can be realized for reduction operations. Note that the cube-connected workstation network does not necessarily inherit all the characteristics of the traditional hypercube. The 3-dimensional cube in Figure 8 requires 6 networks as opposed to 12 connecting wires in the hypercube, and the diameter of the network is reduced to 2 instead of \(k\). The true hypercube, however, is capable of transmitting \(2^k\) messages at any given time, while the cube-connected network is constrained to the number of individual networks (e.g., 6 for a three-dimensional cube). Like the traditional hypercube, the cube-connected network is not easily scalable since an increase in the number of machines requires that an entire additional dimension be added (i.e., going from 8 to 16 machines).
The 2-dimensional mesh is another possible topology for the workstation cluster. An $k \times k$ mesh may be constructed from $k^2$ workstations and $2k$ networks. The time on the workstation mesh for a broadcast communication is

$$2(k - 1)(\lambda + d\tau)$$

rather than the single-network time of

$$(k^2 - 1)(\lambda + d\tau)$$

As in the cube-connected topology, the workstation network configured in this manner does not necessarily inherit the characteristics indigenous to the traditional multicomputer mesh. For example, there may be $k^2$ concurrent communications in a traditional mesh, but only $2k$ are permitted in the network version of this topology.

Other topologies such as the shuffle-exchange network, pyramid, and hypertree can be constructed, as can a ring of rings, a ring of cubes, a cube of meshes, and so on.

Routing protocols provide a service that establishes the necessary path for a message at the time it is generated. This means that the number of hops (gateway crossings) between 2 communicating processors may largely be ignored. Note that the maximum hops in the $k$-dimensional cube-connected workstation network is 2, as opposed to $k$ in the traditional hypercube. The maximum hops in an $k \times k$ workstation mesh is 2, rather than $2(k - 1)$ in a multicomputer mesh. Even though there is some cost associated with network crossings, the number of crossings is modest.

There is one other possibility that deserves consideration: the completely connected graph model. In this topology, one network is required for every machine in the cluster. This is, of course, unreasonable if there are many machines or if the network is geographically widespread, but it may be feasible for a small cluster. With the completely connected model, any topology can be embedded in the physical network. Instead of a single processor taking $(p - 1)(\lambda + d\tau)$ time to send $(p - 1)$ separate
Table 1. The number of nodes, diameter, bisection bandwidth, and number of connections at each node for the single network, the completely connected workstation network, the cube-connected workstation network, and the 2-dimensional mesh workstation network.

<table>
<thead>
<tr>
<th>Topology</th>
<th>Nodes</th>
<th>Diameter</th>
<th>Bisection Bandwidth</th>
<th>Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Network</td>
<td>$p$</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Completely Connected</td>
<td>$p$</td>
<td>1</td>
<td>$p$</td>
<td>$p$</td>
</tr>
<tr>
<td>Cube-Connected</td>
<td>$p = 2^k$</td>
<td>2</td>
<td>$2^{k-1}$</td>
<td>$k$</td>
</tr>
<tr>
<td>Mesh</td>
<td>$p = k^2$</td>
<td>2</td>
<td>$k$</td>
<td>2</td>
</tr>
</tbody>
</table>

messages for a broadcast, a hypercube-style broadcast can be used, taking only time $\log_2 p (\lambda + d\tau)$. As in a typical workstation network, a message between any two processors takes $(\lambda + d\tau)$ time, but $p$ processors can transmit concurrently. Thus, $p$ messages can be sent in time $\lambda + d\tau$ instead of time $\lambda + pd\tau$. All aspects of communication as well as computation can proceed in parallel.

When comparing networks, it is common to consider the diameter, bisection bandwidth, and number of connections at each node. Table 1 shows this information for the single network, the completely connected model, the cube-connected network, and the mesh network.

Assuming the availability of networks, a flexible routing protocol, and sufficient machine connection ports, the best network topology is the completely connected model, since it allows parallelism to extend completely to communication as well as computation. It also has the best diameter and bisection bandwidth. Failing this, the best choice depends on the available resources and the goals of the system. An almost completely connected network may be a reasonable compromise (as in the switched networks mentioned in the previous section). If the number of connections is the overriding constraint, mesh should be considered. If, on the other hand, bisection bandwidth and broadcast time are critical, a cube-connected network may be
advantageous. It is assumed, however, that the network protocols alleviate the cost associated with multiple hops between processors.

Multiple network configurations offer some advantages over typical single-network installations. Concurrent communications are permitted and the bisection bandwidth is higher than in a single LAN. The advent of the switched network, with its almost-completely-connected performance levels, has to a great degree lessened the interest in multi-nets. This investigation assumes that the multiple network configurations described in this section are not available. As mentioned previously, the work presented here is based on single Ethernet connections with a CSMA/CD protocol.
Chapter 4

Data Decomposition Methods

Partitioning the data according to the communication needs of both the application program and the relative speeds of the participating processors is a critical factor in obtaining maximum performance in the networked environment.

This chapter examines a number of standard decomposition schemes. The effectiveness of each method depends on the communication patterns of the particular problem, the size of the problem, and the overhead associated with transmitting data over the network. The partitioning methods considered are scatter, contiguous point, contiguous row, interleaved, and homogeneous block.

While regular communication patterns are assumed for the problem space dealt with here, there may be times when a data item must be transmitted to an arbitrary grid point. In these cases there must be a method for locating the host processor for that point. Often a lookup table must be utilized that contains some or all of the grid points and their locations. For each decomposition method presented here, a comment is made regarding the storage size and lookup time required.

Unless otherwise specified, it is assumed for this investigation that the problem space is a uniform two-dimensional grid with dimensions $m \times n$. It is further assumed that the grid is toroidal: that is, all edges wrap.
4.1 Scatter Partitioning

Scatter partitioning decomposes the problem space without regard to the communication patterns in the application program, but it provides a very flexible framework for systems that offer dynamic load balancing with data migration. This is a fine-grained decomposition in which the individual elements of the problem space are randomly sprinkled or scattered among the nodes of the parallel machine [43]. If the nodes are homogeneous, every processor is allocated the same number of data items. In heterogeneous systems each processor is assigned an amount of data commensurate with its relative speed. Figure 9 gives an example two-dimensional grid that has been partitioned for heterogeneous processors by the scatter method.

The bookkeeping overhead for scatter decomposition is substantial and may preclude the use of this partitioning method for large problems. Since there is no regularity in the assignment of data elements to processors, the only way to determine on which processor a particular data point lies is to keep a table that contains this information. Assuming that the number of the processor associated with a data element can be expressed as a single byte, a table of \( mn \) bytes will be needed. If this
table exists, however, standard array indexing will locate the entry for any element in constant time.

One way to reduce the table size is to fragment the lookup table among the processors. A simple hash function can be used to decide into which table a data element should be entered. This, however, may require an interprocessor communication when determining the physical processor to which a message must be sent. In the worst case, the $p$ processors will each initiate $p-1$ messages and each must respond to $p-1$ messages. Considering the high communication cost associated with the network parallel processing model, table fragmentation merely exacerbates this problem.

Scatter decomposition is of value when load balance is more important than communication cost [43]. It is of particular value when dynamic data migration is used to balance the load during execution. Overloaded processors may directly off-load to underloaded processors without regard to contiguity of the data elements. Unfortunately the storage requirements of the lookup table limit the usability of this partitioning scheme.

A more coarse-grained variant of scatter decomposition scatters entire rows, columns, or blocks of the problem space among the processors. This substantially reduces the necessary size of the lookup table. If the decomposition unit is column or row, merely storing the identifier of the beginning element is sufficient. If the unit of partitioning is a uniform-sized block, recording the identifier of the top-left element of each block permits calculation of identifiers of the elements each contains.

Scatter decomposition is unable to take advantage of regular communication patterns to reduce communication costs. As shall be seen later in this paper, this partitioning method incurs such a large worst-case communication cost for the grid problems we consider that it is impractical to use unless dynamic load balancing is of prime concern.
4.2 Contiguous Point

Contiguous point partitioning affords the opportunity for fine-grained load balancing just as scatter decomposition does, but it offers a much lower worst case communication cost. Contiguous point partitioning by row is described here, but the characterization is analogous for decomposition by column. Data elements are contiguously assigned to processors according to their relative speeds so that all communications to neighbors lying to the east and west require no off-processor communication, with the possible exception of the first and last data items of the first and last rows on each processor. If the processors are homogeneous and the number of processors evenly divides $m$, this becomes a contiguous row partitioning. Otherwise, there are at least two rows in the grid that have elements on different processors. For the analysis, we assume that $p - 1$ rows of the grid have data points on different processors so that no processor is assigned a simple contiguous row partition, and that at least $n$ data points are assigned to each processor. Figure 10 gives an example of a grid partitioned by contiguous points.

The storage required by the lookup table is minimal. For each processor it is merely necessary to maintain the identifiers for the beginning data elements. If the
identity of the data items may be stored as an integer, the size of the requisite data structure is $4p$ bytes. This may be stored as a one-dimensional ordered array and finding the processor on which any element resides may be accomplished using a binary search in $O(\log p)$ time.

This partitioning scheme offers fine-grained decomposition on par with the scatter method but avoids the drawbacks of high communication cost and lookup table storage requirements. While search must be used to locate the processor number on which a data item resides, the list is small and ordered. So negligible time is required for processor lookup.

### 4.3 Contiguous Row Partitioning

This larger-grained version of contiguous partitioning uses the row (or column) as the unit of decomposition. This reduces the number of communications without increasing the message size or lookup table size. In the homogeneous case, an equal number of rows is assigned to each processor. For the heterogeneous network, each processor receives a number of rows proportional to its relative speed. No off-processor
communication is necessary for east/west communication patterns. Figure 11 depicts a heterogeneous contiguous row partitioning.

The lookup table maintained for this method is the same as for contiguous point partitioning. It is sufficient to maintain merely the identifier of the beginning data element on each processor. So the same size of $4p$ bytes and lookup cost of $O(\log p)$ applies as in contiguous point.

Throughout this dissertation references to contiguous row partitioning apply equally to contiguous column decomposition.

4.4 Interleaved Partitioning

Interleaved (round-robin) partitioning is particularly useful for problems such as Gaussian elimination where matrix rows are eliminated from computation operations one-by-one. Figure 12(a) shows a partitioning where entire rows are assigned to homogeneous processors. In the best case the number of rows in the grid is a multiple of $p$ so that all processors have the same number of data points, otherwise the processors begin with an unbalanced workload.
There is no need for a lookup table with this partitioning scheme, since the processor location of any grid point is accessible through direct computation using a \( \text{div} \) and a \( \text{mod} \) operation \(((vp \ \text{div} \ n) \ \text{mod} \ p)\).

The decomposition is more complex when the processors are not homogeneous. The simplest heterogeneous interleaving method doles out data items (points, rows, columns, or blocks) in round-robin fashion. Figure 12(b) gives an example of row interleaving with four processors with relative speeds 4,2,2,1. Notice that the relative speeds of the participating processors form a pattern that is repeated across the problem space until the entire grid has been partitioned. Lookup for this form of heterogeneous interleaved partitioning may be handled either by a table of size \( 4m \) bytes or by calculation based on the interleaving pattern.

As has been previously mentioned, the problem space considered here usually exhibits regular nearest neighbor communications. Interleaved partitioning intentionally places neighboring points on different processors. This decomposition method is attractive mainly because of its simplicity and the balanced computational workload it generates for certain problem classes, but may result in an inordinate amount of communication time for the communication patterns typical of grid problems. The analysis considers only interleaved row partitioning.

4.5 Homogeneous Block Partitioning

The goals of block partitioning are to maximize the number of neighboring data points residing on a single processor, while minimizing the perimeter and, therefore, the amount of data that must be transmitted to neighboring processors. The degree to which these goals are realized is highly dependent on the communication patterns indigenous to the particular problem, and the number and abilities of the participating processors.
Figure 13. A homogeneous block decomposition for 9 processors.

When the problem space may be expressed as an \( n \times n \) grid, the processors are homogeneous, and their number is a square where \( \sqrt{p} \) evenly divides \( n \), the number of data points on the perimeter of a block is minimized. Figure 13 shows a homogeneous block decomposition for 9 processors.

There is no lookup table necessary for homogeneous block partitioning. Each data point may be located through direct computation. To avoid the necessity of computing data point locations, the identities of the beginning and ending rows and columns for each processors can be stored in a lookup table. This would require 4 integers per processor and 1 \textit{div} and 1 \textit{mod} operation to find the location of any arbitrary point.

Unfortunately, the conditions of grid size and number of processors that are required for homogeneous block decomposition often cannot be met. In the next chapter the effects of processor heterogeneity and grids that cannot be divided into \( p \) equally sized blocks are considered.
Chapter 5

Heterogeneous Block Decomposition

Block partitioning has generated considerable interest due to its potential for reducing the number of data items that must be communicated at the expense of an additional number of communications. Unfortunately, optimal partitioning for the general case is NP-hard [75]. Therefore heuristics are usually employed.

In this chapter three new algorithms are presented: Binary Recursive Block Decomposition (BRBD), Fair Binary Recursive Decomposition (FBRD), and Partial Homogeneous Decomposition (PHD). Each uses a slightly different heuristic to partition the problem space. The FBRD and PHD are variants of the original BRBD algorithm and may provide better performance than BRBD in certain circumstances.

The mathematical motivation and cost analysis for block decomposition is deferred until Chapter 6. Upper bounds, however, are established on the number of communications generated for a 5-point stencil communication pattern by the binary recursive method and a competing technique, the N4 General Quadrilateral [51].

5.1 The General Quadrilateral

A block decomposition method has been proposed by Hinz [51] that distorts the blocks of a grid partitioning to permit a more accurate apportionment of data items to each processor. This allows a fairly fine-grained balance of load, but may generate an excessive number of communications.
A typical General Quadrilateral partitioning is shown in Figure 14. Note that, due to non-rectangularity, processor 4, for example, must communicate south with processors 3, 5, and 7. In the worst case a 5-point stencil pattern will require \(8p - 4\) communications per iteration with this decomposition methods. A constructive proof is offered for this result.

**Theorem 1** Let \(p\) be the number of processors to be used in executing a grid problem and assume the grid may be divided into \(p\) rectangles of equal size. Then the worst case number of communications necessary for a single iteration of nearest neighbor 5-point stencil interaction for a general quadrilateral partitioning is

\[8p - 4\] (5.5)
Figure 16. The toroidal problem space may create additional communications with a non-homogeneous decomposition. (a) A homogeneous partitioning has no extraordinary communications. (b) The general quadrilateral and (c) binary recursive decomposition may incur additional communications on each edge equal to one less than the number of processors on the opposing edge. These grids wrap along both the horizontal and vertical edges.

Proof

This bound is proved by classifying the types of quadrilaterals that appear in a decomposition, identifying the maximum number of communications associated with each class, and determining the number of quadrilaterals in each class that must occur in a partitioning.

Classes of Quadrilaterals and Their Costs

1. Corner quadrilaterals (Figure 15(a))—6 communications.
2. Edge quadrilaterals (Figure 15(b))—7 communications.
3. Interior quadrilaterals (Figure 15(c))—8 communications.
4. Additional communications due to toroidal problem space—$4(\sqrt{p} - 1)$ communications. Figure 16 shows how additional communications become necessary for the non-homogeneous cases.

The Number of Quadrilaterals in a Decomposition

Corners—4

Edges—$4(\sqrt{p} - 2)$
Interiors\(\quad p - 4(\sqrt{p} - 2) - 4 = p - 4\sqrt{p} + 4\)

Total Number of Communications:

\[
(4 \times 6) + ((4(\sqrt{p} - 2) \times 7)) + ((p - 4(\sqrt{p} - 2) - 4) \times 8) + (4(\sqrt{p} - 1)) = 8p - 4
\]

(5.6)

With the general quadrilateral method complete lookup information must be maintained. As in scatter decomposition, this method requires a complete table of size \(mn\) to associate data elements with physical processors. The size of this structure, along with the possible additional communications generated may make this approach unattractive.

5.2 Binary Recursive Block Decomposition (BRBD)

The method of block partitioning described here is inspired by the binary decomposition scheme of Berger and Bokhari [17]. Their work deals with partitioning non-uniform grids across a collection of homogeneous processors. A binary decomposition of the domain is used to divide it into regions of equal computational effort.

For the purposes of this research, however, instead of dividing the domain into regions of equal computational effort at each iteration, the domain is partitioned according to the relative speeds of the processors. This algorithm can be applied in either the homogeneous or heterogeneous case, and it accommodates any number of physical processors. This partitioning method is not, of course, optimal. The main consideration in block partitioning is to reduce the overall computation and communication time in relation to other decomposition schemes. This is done when the savings in the number of data items transmitted at each communication point outweighs the latency cost of additional messages generated by block partitioning.
Figure 17. The steps necessary to partition a grid according to BRBD for 5 heterogeneous processors. The relative speed table is given at bottom right.

Relative Speeds

P0 = 5  P3 = 3
P1 = 4  P4 = 2
P2 = 4

Figure 18. Binary recursive block decomposition for 6 processors with the given relative speed table.

Relative Speeds

P0 = 6  P3 = 2
P1 = 5  P4 = 2
P2 = 4  P5 = 1
The binary recursive block decomposition method given here exhibits a lower number of communications than the general quadrilateral in the worst case. The grid is sectioned recursively until each processor has a rectangular block of data items commensurate with its processing speed. Figure 17 shows the steps necessary to partition a grid for 5 heterogeneous processors with speeds (5,4,4,3,2). The processors are formed into two lists: (5,4,4) and (3,2). In Figure 17(b) the grid is split vertically to apportion each list its share of the space. The algorithm then recurses on the first list and its region of the grid. The (5,4,4) list is split into (5,4) and (4). At each recursive call the direction of the next dividing cut changes. Therefore, the region is divided horizontally to give the (5,4) list its share of the region in Figure 17(c). The algorithm recurses again on (5,4) and makes a vertical cut (Figure 17(d)) to assign regions to these processors. The recursion unwinds until the (3,2) list is finally considered. The right-hand region is divided horizontally to give each of these processors their fair share as shown in Figure 17(e). Figure 18 shows the partitioning for 6 processors with speeds (6,5,4,2,2,1).

The number of communications generated for a 5-point stencil in the worst case is $6p - 4$. An inductive proof of this bound is offered, but first we establish that the worst case number of communications for binary recursive block decomposition for 4 heterogeneous processors is $6p - 4 = 20$. This is shown by construction in Figure 19 where we depict the worst case partitioning for 4 processors, and indicate (a) the number of interior communications, (b) the communications due to wrapping on the vertical grid edges, and (c) those due to wrapping along the top and bottom edges. This is used as the base case for the proof.

It is also necessary to see that, for a number of processors greater than 4, a maximum of 6 more communications are associated with each node. Each additional line cutting the grid creates at most 3 more line segments, itself and 1 each on the 2
Figure 19. The worst case binary recursive block decomposition for 4 processors. (a) shows the 10 communications internal to the grid. (b) shows the 6 east/west communications due to wrapping and (c) gives the 4 north/south wrapping communications.

... lines it intersects. Since data must flow in both directions across the boundaries, at most 6 communications are generated.

**Theorem 2** Let $C_p$ be the total number of communications in a recursive binary block partitioned grid problem for $p$ processors. Then for $p \geq 4$

$$C_p \leq 6p - 4$$

**Proof**

**Basis**

Let $p = 4$.

Then by construction (Figure 19) the maximum number of communications is 10 for interior communications, 6 for vertical edges, and 4 for horizontal edges.

$$C_p \leq 6p - 4 = 20.$$

**Induction Hypothesis**

Assume $C_p \leq 6p - 4$ holds for $4 < p \leq k$. 
Induction Step

\[
C_{k+1} \leq C_k + 6 \\
\leq \frac{6k - 4 + 6}{C_k} \\
= 6k + 6 - 4 \\
= 6(k + 1) - 4
\]

Therefore \(C_p \leq 6p - 4\).

\(\square\)

Since rectangularity is preserved in the binary recursive method, the lookup table to locate an arbitrary data point is much smaller than with the General Quadrilateral. It is merely necessary to record the identities of the top-left and bottom-right data points. It is then easy to calculate whether or not a data point is local to a given processor.

The algorithm appears in Figure 20.

5.3 Fair Binary Recursive Decomposition (FBRD)

In this variant of the original algorithm, more care is taken to assure that the blocks are as \textit{chunky} as possible. The original algorithm apports a region to the \([p/2]\) fastest processors, with the remainder going to the \([p/2]\) slowest processors. This may have a tendency to generate long skinny regions for the slower processors, thereby avoiding the unique opportunity that block decomposition offers to substantially reduce the number of transmitted data items.

As in the BRBD algorithm, the list of processors is assumed to be presented in descending order by relative speed. The FBRD assigns the processors to 2 lists using strict alternation until the addition of a processor to a list would increase the total speed for that list beyond the total relative speeds divided by 2. At this point
Initial conditions:

- The current grid is the entire problem space.
- \( \text{dimension} = \text{vertical} \).
- \( P \) = the list of processors being used in the computation in descending order of relative speed.
- \( p \) = the total number of processors to be used (the cardinality of \( P \)).

\[\text{Partition(current grid, dimension, P, p)}\]

1. If \( p = 1 \), end.

2. Sum the relative speeds of the processors in \( P \) and consider the \([p/2]\) fastest processors.

3. Divide the current grid into two partitions along the indicated dimension according to the ratio that the relative speed total of the fastest \([p/2]\) processors bears to the sum of all the relative speeds of the processors in \( P \).

4. If \( p/2 > 1 \)
   
   (a) Set \( t = p \).
   
   (b) Set \( T = P \).
   
   (c) If \( \text{dimension} = \text{vertical} \), set \( \text{dimension} = \text{horizontal} \) else set \( \text{dimension} = \text{vertical} \).
   
   (d) Set \( p = [t/2] \).
   
   (e) Set \( P \) to the first \( p \) processors.
   
   (f) Partition(first partition, dimension, P, p).
   
   (g) Set \( p = t - p \).
   
   (h) Set \( P = T - P \).
   
   (i) Partition(second partition, dimension, P, p).

5. End.

Figure 20. Binary Recursive Block Decomposition Algorithm
Figure 21. The steps necessary to partition a grid according to FBRD for 5 heterogeneous processors with speeds (3,3,3,2,2).

remaining processors are assigned, one at a time, to the list with the lowest total list relative speed.

Figure 21 shows an example for processors with speeds (3,3,3,2,2). In Figure 21(a), the processors are split into two lists (3,3) and (3,2,2) and the problem space is divided across the initial dimension. The left region is divided along its longest dimension for the (3,3) list in Figure 21(b). The (3,2,2) list is subdivided into (3) and (2,2). In Figure 21(c) the right-hand region is cut across the longest dimension to accommodate the speeds of these two lists. Finally in Figure 21(d) the grid is divided, again across the longest dimension, for the (2,2) list.

The algorithm is given in Figure 22.
Initial conditions:

- $G$ = the current grid space.
- $P$ = the list of processors in descending order of relative speed.
- $p$ = the cardinality of $P$.
- dimension = the longest dimension (defaulting to vertical).

Partition($P$, grid)

1. If $p = 1$, end.
2. Total = Sum of the relative speeds of the processors in $P$.
3. Divide the processors into 2 sublists, $P_1$ and $P_2$ according to the following method:
   
   Until an assignment would increase the total speed of a sublist to more than $\frac{\text{Total}}{2}$
   
   Alternatively assign the processors in $P$ to $P_1$ and $P_2$

Repeat

While processors remain, assign next processor to sublist with lowest total speed.

If the vertical edge of the current grid is longer than the vertical edge then dimension = vertical.

Else dimension = horizontal.

Divide the grid into $G_1$ and $G_2$ across dimension according to speed of $P_1$ and $P_2$.

Partition($P_1$, $G_1$)

Partition($P_2$, $G_2$)

4. End.

Figure 22. Fair Binary Recursive Decomposition Algorithm
5.4 Partial Homogeneous Decomposition (PHD)

Frequently more than one machine has the same estimated relative speed, but complete homogeneous partitioning is not possible unless all machines have the same speed. Grouping the processors by relative speed and assigning regions of the grid to these groups instead of to individual processors allows homogeneous decomposition to be used within these regions. This partial homogeneous decomposition can reduce the message size while keeping the number of communications low.

An example of this partitioning method is given in Figure 23. It is assumed that there are nine processors with relative speeds (4,4,3,3,3,2,2,1,1). These are formed into four groups with group speeds (9,8,4,2). The total relative speed for all groups is 23, and a call to the multi-group partitioning algorithm divides the groups into two sets with relative speeds (9,2) and (8,4). Figure 23(b) shows the division of the current grid for these two sets. The first set is then partitioned by another call to the multi-group algorithm (Figure 23(c)). Two calls to the single group partitioning algorithm handle the grid decomposition for the processors with relative speed 3 (Figure 23(d)), and the processors with relative speed 1 (Figure 23(e)). Multi-group partitioning is again invoked, this time with the set (8,4), which is further divided into 2 sets of (8) and (4), and the current grid is decomposed as shown in Figure 23(f). The single group algorithm is called twice to yield the partitioning in Figure 23(g) and (h).

This partitioning algorithm is outlined in Figures 24, 25, and 26, which give the top-level definitions and partitioning procedure, and the single-member group and multiple-member group partitioning algorithms. The partitioning subroutines call themselves recursively. The base case occurs when there is only one group remaining to be partitioned across the current grid, and the number of processors in the group is 1 or is a non-prime number. Otherwise, further subdivision is required. In this algorithm, the term group is used to denote a collection of processors with the same
Figure 23. A partial homogeneous partitioning for 9 processors with relative speeds (4,4,3,3,3,2,2,1,1).
Definitions:

Group = A collection of processors with the same relative speed.
Set = A collection of groups.
Region = A grid subdivision allocated to a group.
Block = A section of the grid allocated to a single processor.

Successor(dimension)

\[
\begin{align*}
\text{vertical} & \quad \text{if vertical dimension of current grid is greater} \\
& \quad \text{than the horizontal dimension} \\
\text{horizontal} & \quad \text{otherwise}
\end{align*}
\]

Preparation for partitioning:

Sort the processors into groups by relative speed and sum the relative speeds into a grand total.
Total the relative speeds of each group and assign that total as the group speed.
Sort the group speeds in descending order.
Set current grid to the entire grid.
Set current dimension to vertical.
Partition(groups, grand total, current dimension, current grid)

If there is only 1 group, Single Group Partition(groups, grand total, current dimension, current grid)
Else Multi-group Partition(groups, grand total, current dimension, current grid)

Figure 24. Partial Homogeneous Decomposition definitions and preparation for partitioning.
Single Group Partition (groups, total, dimension, grid)

If the cardinality of the group is 1,

Return.

Else if the cardinality of the group is not prime,

Find the two largest factors whose product equals the cardinality of the group.
Divide the longest edge of grid by the largest of these factors.
Divide the shortest edge of the grid by the smallest factor.

Else

Divide the group into 2 subgroups:

Assign one of the processors in the group to subgroup 1.
Assign the remaining processors in the group to subgroup 2.

Divide the grid proportionally according to the total speeds of each set along the current dimension.
Set the current grid to the section assigned to subgroup 1.
Partition(subgroup 1, speed of set 1, current dimension, current grid).
Set the current grid to the section assigned to subgroup 2.
Partition(subgroup 2, total speeds of set 2, current dimension, current grid).

Figure 25. Partial Homogeneous Decomposition partitioning algorithm for a single group—the base case.
Multi-group Partition(groups, total, dimension, grid)

Divide the groups into 2 sets:

While set 1 contains groups with speeds totaling less than grand total / 2, assign the next group to set 1.
While set 2 contains groups with speeds totaling less than grand total / 2, assign the next group to set 2.
While groups remain, assign the next group to the set that has the lowest total speed.

Divide the grid proportionally according to the total speeds of each set along the current dimension.
Set current dimension to successor(dimension).
Set the current grid the the section assigned to set 1.
Partition(set 1, total speeds of set 1, current dimension, current grid).
Set the current grid to the section assigned to set 2.
Partition(set 2, total speeds of set 2, current dimension, current grid).

Figure 26. Partial Homogeneous Decomposition partitioning algorithm for multiple groups.

relative speed. The term set is used to indicate a collection of groups. There should be no confusion with the mathematical definitions of these terms.

Since block decomposition may yield superior results only when the cost of additional communications is outweighed by bandwidth savings due to smaller message size, it is important to keep each block as square as possible to minimize the number of data items that must be transmitted. The partial homogeneous partitioning method tries to minimize the perimeter/area ratio of the regions by always dividing across the longest dimension. Block dimensions are the two largest factors whose product is the number of processors assigned to the group. This assures that the blocks are kept as square as possible. An open question regarding the grouping of processors is determining how close the speeds of two nodes must be before they are considered to have the same relative speed. Simple rounding may be used, but this may result in lost opportunities for better homogeneous subgroupings.
5.5 Expected Performance of the Algorithms

Figure 27 demonstrates the three different partitioning methods. While the worst case maximum number of messages for is the same for all methods, in this example BRBD requires 42 messages per iteration, and FBRD has 43, and PHD needs 50. However, only $11n$ data items are transmitted by FBRD, $12n$ are sent by PHD, and $13n$ data items are transmitted by BRBD. Obviously, if $n$ is small, the number of communications is most critical factor. However, as $n$ increases, FBRD should yield superior performance.

In Figure 28, PHD has the upper hand, providing a lower number of communications and fewer items to be transmitted than either BRBD or FBRD.

The algorithm that yields the best expected performance depends entirely upon the specific collection of machines to be used in the computation.
Figure 27. A comparison of partial homogeneous partitioning with a binary recursive partitioning. The number within each sub-block indicates the number of communications required by that sub-block during each iteration. The BRBD requires 42 communications and transmits $13n$ data items. FBRD sends 43 messages and transmits only $11n$ items. PHD requires 50 communications and sends $12n$ data points. This example assumes 9 processors with speeds (4,4,3,3,2,2,1,1,1).

Figure 28. A comparison of partial homogeneous partitioning with a binary recursive partitioning. The number within each sub-block indicates the number of communications required by that sub-block during each iteration. Here the BRBD generates 28 communications and transmits $11n$ data items. FBRD requires 32 messages with $11n$ data items. PHD generates 24 messages and sends $10n$ items. The 9 processors are assumed to have speeds (4,4,4,3,3,3,3).
Chapter 6

Mathematical Characterization of Communication Costs

In this chapter the relative time cost of the various decomposition schemes is mathematically characterized for a number of common communication patterns. The patterns we consider are broadcast, reduction, random and three forms of nearest neighbor communications—systolic, wavefront, and 5-point stencil. Figure 29 depicts the movement of data for all these communication patterns except random communications.

The communication costs for broadcast, reduction, and random patterns are insensitive to the partitioning method used. The others, however, have costs that depend upon the way in which the problem space is decomposed across the network. In some cases the cost for heterogeneous processors is no higher than for homogeneous processors, but this is by no means typical. For heterogeneous scatter, interleaved, and block decomposition, for example, the number of communications and communication costs cannot be known without also knowing the exact configuration of the network.

When the processors in the network have differing performance capabilities, the best computational balance is achieved when the number of data points of a uniform problem space assigned to each processor $i$ is

$$\frac{mn(speed_i)}{\sum_{j=0}^{p-1} speed_j}$$

(6.8)
Figure 29. Data movement for different communications patterns. (a) Broadcast—a single value is disseminated to all grid points. (b) Reduction—data items are transmitted from all grid points to a single receiving grid point. (c) Systolic—every data item is synchronously sent to its nearest neighbor. (d) Wavefront—each data item depends on the values to the left and above. (e) Successor/Predecessor—neighboring values along one dimension are needed. (f) 5-Point Stencil—every data item needs access to the values of its nearest neighbors to the north, south, east, and west.
where $speed_i$ is the relative speed of processor $i$, and the divisor is the sum of the relative speeds of all the participating machines. This, however, is an oversimplification. The best partitioning of the data depends on the number of processors, problem size, network speed, bookkeeping overhead, and the problem itself. Some problems are *embarrassingly parallel* in that they have little or no data dependencies. More often dependencies do exist, in which case, the number and size of interprocessor communications becomes an important factor in system performance.

Using the notation from page 11, the cost of a single interprocessor message from processor $i$ to processor $j$ in a virtually completely connected graph may be expressed as:

$$\lambda + \tau(D_{ij} + F\lceil D_{ij}/M \rceil) + W$$

(6.9)

where the first term represents the message preparation latency, and the second term represents transmission time. The third term, $W$, represents any additional wait time due to network contention (i.e., collision resolution overhead or unrelated network traffic) or load imbalance. Within the second term, the $\lceil D_{ij}/M \rceil$ component determines the number of packets needed when the number of bytes to be transmitted exceeds the Maximum Transmission Unit (MTU) and multiple data frames must be generated. The number of bytes transmitted is increased by $F$ bytes for each required data frame. For Ethernet, the size of the data frame header is 14 bytes, the trailer is 4 bytes, and the MTU is 1500 bytes [90]. If the TCP/IP protocol is used, this is reduced by the 40 bytes required for the TCP and IP headers. In data-parallel processing, operations are synchronous. Therefore, if the workload is balanced, all processors will be attempting to communicate at roughly the same time. Additional communication cost may be incurred due to network collision or waiting to seize the token on a Token Ring.

A parallel communication is defined to be a synchronization point where at least one processor is transmitting a message to one or more other processors. The cost of
a parallel communication in a single direction (the cost of messages transmitted by all processors) may then be expressed as:

\[
\lambda(\max_{i\in\{0...p-1\}} C_i) + \tau \sum_{i=0}^{p-1} \sum_{j=0, j\neq i}^{p-1} (D_{ij} + F\lceil D_{ij}/M \rceil) + W \quad (6.10)
\]

where the first term represents the message preparation latency based on the maximum number of messages sent from any single processor during a communication round, and the second term expresses the transmission time due to the size of all message and packet frames. Latency is charged according to the maximum of any participating processor since we assume that messages are prepared in parallel.

The analyses in the following sections include the following simplifying assumptions:

1. The network consists of a single LAN, and only one message is transmitted at any given time.

2. The problem space is uniform and may be represented as a two-dimensional toroidal \( m \times n \) grid.

3. The message transmitted from a grid point consists of a single data item of determinable length \( d \).

4. There is a single dominant communication pattern to be considered.

5. The effects of contention and collision resolution can be captured in the \( \lambda \) and \( \tau \) factors.

These assumptions are not unrealistic. A collection of cooperating machines often resides on a single bus or ring. The analysis applies to non-toroidal problems, but the mathematical characterization is slightly different. Requiring each transmitted data item to have a set size \( d \) is not overly limiting, since this permits entire arrays or structures. Other than broadcast and fan-in of values at the end of execution,
Figure 30. Time lines for four message transmission scenarios. Cases (a), (b) and (e) incur the maximum communication cost, while (c) and (d) overlap the message preparation latency of one processor with the transmission times of other processors. Grid problems are often characterized by a single dominant communication pattern. For problems that experience phase shifts so that the patterns change as execution progresses, communication cost may be analyzed by considering each pattern separately. While the time necessary to handle contention varies considerably with the network traffic, averages can be estimated by timing test programs that generate the communication patterns examined in this research.

Using these assumptions, the cost of a single iteration may then be generally expressed as:

$$\lambda \sum_{k \in \{N,S,E,W\}} (\max_{\ell \in \{0...p-1\}} C_{i_k}) + \tau \sum_{k \in \{N,S,E,W\}} \sum_{i=0}^{p-1} \sum_{j=0, j \neq i}^{p-1} (D_{i,j} + F [D_{i,j} / M])$$  \hspace{1cm} (6.11)

where $N$, $S$, $E$, and $W$ are the possible directions of a communication.

Since message preparation may be performed entirely in parallel by the participating machines, the associated latency is not directly affected by the number of processors. The communication costs may actually be less than those indicated due to the effects of overlapping message preparation time on one processor with transmission time of other processors. Figure 30 depicts four processors, each generating one communication. The MTU is exceeded by one processor in (b) and (c), and by all processors in (d) and (e). In (a), (b) and (e) the maximum cost as expressed in
equation (4) is incurred. In cases (c) and (d), however, advantage is taken of the message preparation overlap so that overall communication time is less than the upper bound. The worst case as depicted in 30(a), (b) and (e) is assumed in our analysis.

Minimizing the number of communications, the size of communications, and the computation time are conflicting goals. In the following sections we analyze the impact that partitioning methods have on performance depending on the communication patterns involved.

6.1 Broadcast

In a broadcast communication a single processor transmits a message to the other \( p - 1 \) processors (see Figure 29(a)). Therefore, there are \( p - 1 \) communications regardless of the data partitioning methods used. By definition, the same message is sent to all processors, so the communication cost may be expressed as

\[
\lambda(p - 1) + \tau(p - 1)(d + F\lceil d/M \rceil)
\]

regardless of the decomposition method. No lookup table is required for broadcast messages since a message is sent to every participating processor.

Multicast is a variation of the broadcast pattern that communicates with only a subset of the other \( p - 1 \) processors [62]. Multicast communication cost is completely dependent on the application. Some problems may be decomposed so that all intended recipients reside on the same processor, or at least a small number of processors. In other cases, there may be no a priori way to determine recipient sets in order to optimize the communication costs. Unlike broadcast, which distributes data indiscriminately, multicast needs a mechanism to determine on which processor the intended recipient resides. In the worst case the cost of a multicast may be greater than a regular broadcast since the message may have to be sent to every other processor in addition to incurring lookup time.
If the available hardware and software provide automatic multicast service, only one message must be transmitted. This investigation does not assume the availability of this service.

6.2 Reduction

Reduction patterns are similar to broadcast, but instead of a single processor generating \( p - 1 \) messages, \( p - 1 \) processors all generate a message to a common recipient. Figure 29(b) shows this pattern. Typically this communication pattern is used to determine maximum or minimum values among all the data points, or to perform some operation, such as summation, across all data points. Local reduction is handled at each node so that only a single data item is actually transmitted from each processor. The message preparation latency is amortized in parallel across the entire set of transmitting processors, so message-passing overhead is lower than in broadcast. Thus, the cost of a reduction can be expressed as

\[
\lambda + \tau(p - 1)(d + F([d/M]))
\]

As in broadcast, this cost is independent of the partitioning scheme.

6.3 Random

Some problems have random communication patterns; that is, it is not possible (or it is impractical) to determine message passing patterns prior to the time of communication. In the worst case, each processor must send a message to every other processor, regardless of the decomposition method used, so the worst-case number of communications is \( p(p - 1) \). Such a communication will have cost

\[
\lambda(\max_{i \in \{0...p-1\}} C_i) + \tau \sum_{i=0}^{p-1} \sum_{j=0, j \neq i}^{p-1} (D_{ij} + F[D_{ij}/M])
\]

which is the same as Equation 6.10.
6.4 Systolic

In systolic array problems, as in a heart pulse, data is pumped from each grid element to the neighboring element [76]. Figure 31 shows this pattern for different decomposition techniques.

If scatter partitioning is used, it is again possible that every processor must communicate with every other processor, so the worst case number of communications per iteration is \( p(p-1) \) and the maximum number of data items transmitted is \( mn \) (Figure 31(a)). The communication cost cannot be estimated until the exact partitioning of the data is known. So it may be expressed as

\[
\lambda \left( \max_{i \in \{0, \ldots, p-1\}} C_i \right) + \tau \sum_{i=0}^{p-1} \sum_{j=0, j \neq i}^{p-1} (D_{ij} + F[D_{ij}/M])
\]  

(6.15)

With contiguous point partitioning (Figure 31(b), both the number and size of communications may be dramatically reduced since processors numbered 1 through \( p-2 \) must send 2 data items in the direction of the flow, and processors 0 and \( p-1 \) send 1 item each. The number of communications generated is \( 2(p-1) \). The resulting communication cost is

\[
2\lambda + 2\tau(p-1)(d + F[d/M])
\]  

(6.16)

If the problem space is decomposed by contiguous row or interleaved row, there are no off-processor communications at all for systolic operations. Figure 31(c) and (d) show this case.

If the conditions for simple block partitioning are met, the straightforward decomposition shown in Figure 31(e) may be used. In which case, the number of communications is \( p \), the size of each message is \( dn/\sqrt{p} \) and the cost is

\[
\lambda + \tau p \left( (dn/\sqrt{p}) + F[(dn/\sqrt{p})/M] \right)
\]  

(6.17)
Figure 31. A systolic communication pattern with the grid divided for 4 processors according to the decompositions methods—(a) scatter, (b) contiguous point, (c) contiguous row, (d) interleaved row, (e) homogeneous block, and (f) heterogeneous block.
For binary recursive decomposition (Figure 31(f)), because of the strict alternation in the direction of cuts made in the grid, the number of communications is bounded by

\[ 6 + \sum_{i=4}^{p-1} \begin{cases} 
 2 & \text{if odd (} \lfloor \log i \rfloor \text{)} \\
 1 & \text{otherwise}
\end{cases} \]  

(6.18)

The general quadrilateral method generates \(2p-1\) communications. For systolic communication, each processor sends one message in the direction of information flow (\(p\) messages). The non-perpendicularity of interior boundaries gives rise to another \(\sqrt{p}(\sqrt{p}-1)\) communications, and there are \(\sqrt{p}-1\) possible additional communications due to the toroidal wrap. Therefore the worst case number of communications for the general quadrilateral is

\[ p + \sqrt{p}(\sqrt{p} - 1) + (\sqrt{p} - 1) = 2p - 1 \]  

(6.19)

See Section 6.6 for proof of the number of communications needed for the more general stencil communication pattern. The cost may be expressed by the same equation as for scatter partitioning.

6.5 Wavefront

Wavefront communication patterns take the form shown in Figure 29(d). Each data point interacts with two of its nearest neighbors at 90° angles from each other. The case considered here assumes east- and south-bound communications. The arguments are analogous for other directions.

If scatter decomposition is used, the number of communications generated may be as great as \(2p(p-1)\) and the worst-case communication cost is

\[ \lambda \sum_{k \in \{E,S\}} \max_{i \in \{0, \ldots, p-1\}} C_i + \tau \sum_{k \in \{E,S\}} \sum_{i=0}^{p-1} \sum_{j=0, j \neq i}^{p-1} (D_{ijk} + F[D_{ijk}/M]) \]  

(6.20)
Contiguous point partitioning localizes most of eastbound communication and generates at most \( p + 2(p - 1) \) communications. The associated worst-case communication cost is

\[
3\lambda + \tau(d(np + 2(p - 1)) + F([dnp/M] + 2(p - 1)[d/M]))
\]

(6.21)

When the problem is decomposed according to contiguous row, \( p \) communications of \( n \) data items each are generated. The communication cost, therefore, is

\[
\lambda + \tau p(dn + F[dn/M])
\]

(6.22)

If partitioning is by interleaved row, \( mn \) data items are involved. The communication cost is

\[
\lambda + \tau p(d[mn/p] + F[d[mn/p]/M])
\]

(6.23)

For heterogeneous interleaving, the same \( mn \) data items must be transmitted as with homogeneous partitioning. However, the worst-case number of communications is \( 2(p - 1) \). Since the fastest and slowest processors each communicate with one other processor, and the others communicate with at most two other processors, the communication cost is

\[
\lambda(\max_{i \in \{0, \ldots, p-1\}} C_i) + \tau(dmn + F[p \sum_{i=0}^{p-1} \sum_{j=0, j \neq i}^ {p-1} [D_{ij}/M])
\]

(6.24)

This expression assumes that all messages from processor \( i \) to processor \( j \) are bundled into a single transmission affected only by the MTU limits.

Simple block decomposition incurs \( 2p \) communications, and the cost may be expressed as

\[
2\lambda + 2\tau p(d(n/\sqrt{p}) + F[d(n/\sqrt{p})/M])
\]

(6.25)

The number of communications for heterogeneous block decomposition is

\[
\sum_{k \in \{e, s\}} \sum_{i=0}^{p-1} C_{ik}
\]

(6.26)
Figure 32. Typical stencil patterns for communication in grid problems that are amenable to block decomposition. (a) 3-point stencil. (b) 5-point stencil. (c) 9-point stencil.

where $C_{ik}$ indicates the number of messages transmitted from processor $i$ in direction $k$ and $k$ ranges from east to south. The cost may be characterized as

$$\lambda \sum_{k \in \{E,S\}} \max_{i \in \{0,...,p-1\}} C_{ik} + \tau \sum_{k \in \{E,S\}} \sum_{i=0}^{p-1} \sum_{j=0, j \neq i}^{p-1} (D_{ijk} + F[D_{ijk} / M])$$

(6.27)

6.6 Stencil

Stencil communication patterns are usually of the successor/predecessor form (3-point stencil) shown in Figure 32(a) or the north/south/east/west patterns (5-point stencil) in 32(b). Other forms occur, such as the 9-point stencil in Figure 32(c). In an application with an $k$-point stencil pattern, each data element interacts with $k - 1$ other elements at each communication point. If care is not taken in the partitioning of the problem space, an unreasonably large communication cost may be generated. Only the 5-point stencil is considered for this analysis.

Scatter partitioning for stencil patterns incurs a very high communication cost. In the worst case, each processor must communicate with the remaining $p - 1$ processors for each of the 4 communication directions. Therefore, $4p(p - 1)$ communications are generated and $4mn$ data items must be transmitted. Figure 33(a) shows 5-
Figure 33. A 5-point stencil communication pattern decomposed by (a) scatter, (b) contiguous point, (c) contiguous row, (d) interleaved, (e) homogeneous block, and (f) heterogeneous block.
point stencil communications with a scatter partitioning. The following expresses the associated cost.

\[
\lambda \sum_{k \in \{N, S, E, W\}} \max_{i \in \{0, \ldots, p-1\}} C_{ik} + \tau \sum_{k \in \{N, S, E, W\}} \sum_{i=0}^{p-1} \sum_{j=0, j \neq i}^{p-1} (D_{ijk} + F([D_{ijk}/M]))
\]  

(6.28)

Continuing the assumption that at least an entire grid row is assigned to each processor in contiguous point partitioning (Figure 33(b)), there will be \(p\) northward communications and \(p\) southward communications. The first processor has 2 communications with the second processor, the last processor has 2 communications with the previous node, and the others exchange 2 messages with each of their predecessor and successor nodes. \(2np\) items are transmitted north and south, while \(4(p - 1)\) items move east and west. Therefore, the total number of communications is \(6p - 4\). This generates a communication cost of

\[
6\lambda + 2\tau(p(dn + F([dn]/M])) + 2(p - 1)(d + F[d/M]))
\]

(6.29)

If contiguous row decomposition is used for a 5-point stencil pattern, east/west communications are local to each processor, so each processor must send a row of data north and south. The cost is

\[
2\lambda + 2\tau p(dn + F([dn]/M))
\]

(6.30)

The number of communications is \(2p\). Figure 33(c) shows this situation.

With contiguous point and contiguous row partitioning, the cost is the same for both homogeneous and heterogeneous processors.

Interleaved partitioning for applications that exhibit stencil patterns increases the number of data items that must be transmitted, even when the processors are homogeneous and \(p\) evenly divides \(n\). While only \(2p\) communications are generated if entire rows (or columns) are distributed in this fashion, all east- and west-bound communications are local to each processor, but \(2mn\) data items must be transmitted at each iteration for messages going north and south. Interleaved partitioning for the
5-point stencil is depicted in Figure 33(d). The communication cost may be expressed as

\[ 2\lambda + 2\tau p \left( \frac{d m n}{p} + F \left[ \frac{(d m n / p)}{M} \right] \right) \]  

(6.31)

Note that \( p \), the number of processors does not influence the amount of data that must be communicated. The large amount of data transmitted for interleaved partitioning imposes a severe bottleneck.

For worst case heterogeneous interleaving, the fastest processor must communicate with the other \( p - 1 \) nodes (see Figure 12(b)) on page 51. The fastest processor communicates with one neighbor to the south, and \( p - 1 \) neighbors to the north. The slowest communicates with one northern neighbor and one southern neighbor. The others communicate with one neighbor to the north, and at most two to the south. This generates a total of \( 4(p - 1) \) communications. In the worst scenario the communication cost is

\[ \lambda \sum_{k \in \{N,S\}} \left( \max_{i \in \{0..p-1\}} C_{i_k} \right) + \tau \left( 2d m n + F \sum_{k \in \{N,S\}} \sum_{i=0}^{p-1} \sum_{j=0,j \neq i}^{p-1} \left[ D_{i_k} / M \right] \right) \]  

(6.32)

If simple block partitioning can be used, the number of communications is \( 4p \) and the cost is

\[ 4\lambda + 4\tau p \left( \frac{d n}{\sqrt{p}} + F \left( \frac{(d n / \sqrt{p})}{M} \right) \right) \]  

(6.33)

The homogeneous situation offers the most efficient block decomposition. Assuming that the problem space and the computer network meet the conditions for the simple block decomposition shown in Figure 33(e), there exists an event horizon that is a function of problem size and number of processors, above which block decomposition should be superior to other partitioning methods for a stencil application [10, 36, 37].

If the processors are heterogeneous, the cost may be characterized as

\[ \lambda \sum_{k \in \{N,S,E,W\}} \left( \max_{i \in \{0..p-1\}} C_{i_k} \right) + \tau \sum_{k \in \{N,S,E,W\}} \sum_{i=0}^{p-1} \sum_{j=0,j \neq i}^{p-1} \left( D_{i_k} + F \left[ D_{i_k} / M \right] \right) \]  

(6.34)
Table 2. The number of messages generated by the various decomposition methods for a 5-point stencil communication pattern on homogeneous and heterogeneous processors.

<table>
<thead>
<tr>
<th></th>
<th>Scatter</th>
<th>Interleaved</th>
<th>Point</th>
<th>Row</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homogeneous</td>
<td>$4p(p-1)$</td>
<td>$2p$</td>
<td>$6p-4$</td>
<td>$2p$</td>
<td>$4p$</td>
</tr>
<tr>
<td>Heterogeneous</td>
<td>$4p(p-1)$</td>
<td>$4(p-1)$</td>
<td>$6p-4$</td>
<td>$2p$</td>
<td>$6p-4$</td>
</tr>
</tbody>
</table>

6.7 Comparisons

As has been stressed by a number of sources, including [9, 10], the number of communications generated in a high latency environment such as the parallel network is often more important than the size of the messages. Table 2 shows the number of messages associated with the different decomposition methods and communication patterns for homogeneous and heterogeneous processors. The graph in Figure 34 provides a different view of this information for homogeneous processors. Figure 35 gives the same information for heterogeneous processors.

Notice that for some decomposition methods, the number of processors used does not affect the worst-case number of data items to be transmitted. Scatter and interleaved partitioning fall into this category. In some cases, such as stencil patterns, the communication costs cannot be predicted in the heterogeneous case without knowing the exact configuration of the parallel network; that is, it is necessary to know the relative speeds of all the participants before anything meaningful can be said about expected performance.

Figure 36 shows the 5-point stencil communication costs for scatter, interleaved contiguous point, contiguous row, and block decomposition for 8 homogeneous processors. The two contiguous methods have almost indistinguishable cost, which indicates that a choice between these methods depends on the desired granularity of load balance. Again, scatter shows the highest cost. This is caused by both the high number
Figure 34. Number of communications for the five decomposition methods for homogeneous processors.

Figure 35. Number of communications for the five decomposition methods for heterogeneous processors.
Figure 36. Communication cost for five decomposition methods for the 5-point stencil pattern on an $n \times n$ grid. There 8 processors and a data size of 8 bytes. $\lambda = 2.5$ msec, $\tau = 1.5$ $\mu$sec, MTU = 1460 bytes, frame headers and trailers = 58 bytes of communications and the number of data items that must be transmitted. The same problem holds for interleaved partitioning. Block partitioning may be expected to return the best performance if the constraints for homogeneous partitioning are met.

While block partitioning generates a greater number of communications than contiguous row decomposition for most communication patterns, the savings in the amount of data transmitted makes this partitioning scheme attractive for large problems with stencil communication patterns. This indicates that, for a collection of homogeneous processors, there exists an event horizon beyond which block decomposition may provide performance superior to contiguous row partitioning. This effect has also been mentioned by other sources, including [10].

The boundary beyond which homogeneous block decomposition is expected to outperform contiguous row is established by the following theorem.
Theorem 3 \textit{Block decomposition provides performance superior to contiguous row or contiguous column partitioning in the homogeneous case only when}

\[
\lambda < pdn\tau (1 - \frac{2}{\sqrt{p}}) \tag{6.35}
\]

\textbf{Proof}

Contiguous row decomposition is considered (the argument for contiguous column is identical).

For both contiguous row and block partitioning, each physical processor has \(n^2/p\) virtual processors so the computation time is the same for each decomposition.

The communication time for contiguous row decomposition may be expressed as

\[
2(\lambda + pdn\tau) \tag{6.36}
\]

The communication time for block decomposition may be expressed as

\[
4\left(\lambda + pd\frac{n}{\sqrt{p}}\tau\right) \tag{6.37}
\]

Block decomposition is superior only when its communication time is less than that for contiguous row.

\[
4\left(\lambda + pd\frac{n}{\sqrt{p}}\tau\right) < 2(\lambda + pdn\tau) \tag{6.38}
\]

Solving for \(\lambda\) yields

\[
\lambda < pdn\tau (1 - \frac{2}{\sqrt{p}}) \tag{6.39}
\]
Figure 37. Minimum number of grid points required for block allocation to be superior to contiguous row or column allocation, as a function of number of processors. Block allocation is never superior to contiguous row or column allocation when there are fewer than 5 processors.

Figure 37 graphically shows the conditions on problem size and number of processors that must be met before block decomposition in the homogeneous case is advantageous. The homogeneous case establishes the lower bound. With fewer processors or a smaller problem size than that indicated in Theorem 1, block decomposition should not be used. Assuming that $\lambda = 1.5 \text{ msec}$, $\tau = 1\mu\text{sec}$ per byte for Ethernet and $\tau = .1\mu\text{sec}$ per byte for FDDI, and data length $d = 8$ bytes, Equation 6.35 indicates that block decomposition with any number of processors less than 5 will not give better performance than contiguous row or column partitioning. Note that as the network gets faster, as with FDDI, the number of data items transferred is no longer as critical an issue.

Since homogeneous block decomposition may be capable of superior performance, can the same be expected from heterogeneous block decompositions? Figure 38 shows the expected communication costs for contiguous point, contiguous row, and block
Figure 38. Communication cost for the contiguous and block decomposition methods on heterogeneous processors for the 5-point stencil pattern on an \( n \times n \) grid. There are 8 processors and a data size of 8 bytes. \( \lambda = 2.5 \) msec, \( \tau = 1.5 \) \( \mu \)sec, MTU = 1460 bytes, frame headers and trailers = 58 bytes. The relative speeds are \((6,4,4,3,3,3,2,1)\).

decomposition for 8 heterogeneous processors with relative speeds \((6,4,4,3,3,3,2,1)\).

The point at which block decomposition is expected to outperform the other partitioning methods depends on numerous factors including the number and relative speeds of the processors, the problems size and data size, and the bandwidth and message preparation latency. When the processors are heterogeneous, the relative costs cannot be estimated without analyzing this information.

Having established mathematically that there are conditions under which heterogeneous block decomposition is advantageous, the question remains regarding the comparative value of the three algorithms presented in Chapter 5.

6.8 Use of the Analysis

The communication cost analysis will be useful to the programmer or compiler in making decisions on static data decomposition at compile time when the relative
speeds of the nodes are fixed and the characteristics of the network are known \textit{a priori}. The communication cost characterization may be of value to the user or runtime system to help decide, at execution time, the best decomposition to match the current performance capabilities of the participating machines and the network.
Chapter 7

A Data Decomposition Advisory System

The communication costs established in the previous chapter are useful for advising the programmer, compiler, or run-time system on the decomposition scheme(s) most likely to yield the best performance for a particular application program and a particular network. In this chapter, an automated advisory system is described that ranks the decomposition methods for a specific application program.

7.1 Description of the System

The partitioning advisory system accepts data regarding the application program, the processors, and the network, and produces a ranking of the various decomposition methods. The system automatically ranks the speeds of the available machines on the network so that the ones expected to give the best performance may be used.

The system does not analyze the program to determine the computational requirements between message-passing rounds, nor does it try to discover the communication patterns present. It is assumed that this information is available either from the programmer or some other program analysis tool. The problem space information needed by the system includes grid size, the size of the individual data items that will be transmitted, the dominant communication pattern, and the number of floating point operations per grid point for each iteration.
Figure 39. Top-level view of the partitioning advisory system. There are 4 distinct phases. (1) Estimating processor speeds. (2) Partitioning the problem space. (3) Estimating network performance levels. (4) Applying the mathematical communication cost characterizations to the gathered data. The arrows indicate data going to and coming from the various system segments. The final output consists of the relative performance rankings for the different partitionings.

Information regarding the network characteristics and the available machines must also be provided to the system. This includes packet size and the size of data frame headers and trailers.

A top-level conceptualization of the overall advisory system is shown in Figure 39. The initial input includes a list of the machines on the network, the number of processors to be used, and all of the information regarding the problem space and network described in the previous paragraphs.

The input to the first phase is the list of available computers on the network. A test program is run in parallel on all machines, and each reports the time required for a set number of floating point operations. This information is used to estimate the instantaneous capability of each processor. A Mflops rating is assigned to each machine, and the list is sorted in descending order based on these ratings. The output of this segment is the sorted list, annotated with the estimated Mflops ratings.
Figure 40. The partitioning phase establishes the size and shape of the data point allocation depending on the relative processor speeds.

Information regarding grid size and the number and relative speeds of the participating processors is used in the next phase to partition the problem space. The system determines the share of workload that should be assigned to each processor based on its relative speed. The grid is then partitioned for each decomposition technique. The data generated by the partitioning phase varies among the decomposition methods. In contiguous row partitioning, for example, the identity of the beginning grid point and the number of rows for each processor are recorded. For block decomposition, however, the identities of the top-left and bottom-right grid points are necessary, as well as information regarding the neighboring processors and the number and size of communications along each border [36]. The data generated by this segment is used during the final ranking phase to supply values for the $C_{ik}$ and $D_{ij,k}$.
Figure 41. To estimate the current communication performance capability of the network, several rounds of message passing are launched. The times reported are used to estimate the current values of $\lambda$ and $\tau$.

terms in the mathematical model. Figure 40 depicts possible partitionings generated for 8 heterogeneous processors.

The number of processors to be engaged and the relative speeds list are the required input to the third phase (Figure 41). In this section the network is tested to determine the available performance given the current traffic load. While the maximum capabilities of the network are usually known (i.e., 10 Mbps for Ethernet, 4 or 16 Mbps for Token Ring, 100 Mbps for FDDI, etc.), these advertised peak performance rates are seldom realized. Additionally, the available performance varies with the level of usage. Therefore it is necessary to garner an indication of the current disposable bandwidth. Merely timing messages bounced between two processors does not capture the possibly debilitating effects of the contention that occurs due to the synchrony of communications in data-parallel applications. To include this effect, the collection of processors is viewed as a ring. Each processor sends a message to its virtual right-hand neighbor, and receives a message from the neighbor on the left. A number of rounds of this message-passing pattern and various message sizes are timed, and a least-squares fit is used on the gathered data to estimate $\tau$, the tran-
Application: thermal.c
Rows: 64
Columns: 64
Processors: 4

<table>
<thead>
<tr>
<th>Block</th>
<th>Row</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.39</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Figure 42. The output of the final advisory phase. The values indicate the relative worth of the two partitioning strategies. The method expected to give the best performance has a rating of 1.00. The ratings of the other schemes are estimates of the factors by which the execution time is expected to increase (e.g., 1.39 indicates a 39% increase in execution time).

mission time, and $\lambda$, the latency, for a single message under current network traffic conditions.

The final advisory stage occurs in the fourth phase. Figure 42 gives sample output for the two decomposition methods considered here. All of the information developed from the previous three phases is used in the equations discussed in Chapter 6 to assign relative rankings to the decomposition schemes for the particular application on the current network. The relative speeds and number floating point operations per grid point are used to estimate computation time, $\max_{i \in \{0, \ldots, p-1\}} t_i$. Input from the partitioning phase provides the values for $C_{ik}$ and $D_{ik}$, and $\tau$ and $\lambda$ are provided by the network performance phase. The values of $d, n, p, F, M$ are assumed to be available as part of the initial data supplied by the programmer or user. The relative value assigned to the method that is expected to provide the best performance is normalized to 1.00, with the remaining schemes expressed as non-integral multiples of this amount. The higher the relative value, the worse its expected performance. For example, a rating of 1.5 indicates an expected 50% increase in execution time over the method ranked at 1.00.
7.2 Experimental results

To validate the model of relative computation and communication costs over various partitioning schemes, a low-level message-passing system with sockets has been implemented to benchmark the different decompositions. The testbed includes a collection of Sun and HP workstations of assorted ages and capabilities. The machines are connected by Ethernet. The benchmarking concern is not with the speedup that can be obtained by using the different decompositions, nor is there interest in absolute performance estimations. The goal of the validation process is to compare the rankings given by the system with observed performance.

The application used for validation is a two-dimensional thermal conduction problem, where a heat source is applied along one column of the toroidal grid, and the heat spreads throughout the domain over a number of iterations. This is a fine-grained problem in that the main computational operations are contained in the single line of code that calculates the updated version for each grid point

\[
p_{\text{point}} = \frac{\text{north}(\text{point}) + \text{south}(\text{point}) + \text{east}(\text{point}) + \text{west}(\text{point})}{4}
\]  

(7.40)

where \(\text{north}(), \text{south}(), \text{east}(),\) and \(\text{west}()\) are abstractions that indicate point values obtained from these nearest neighbors.

In Figure 43 the ratio of predicted to observed relative execution times for block versus row partitioning are given as the number of processors varies from 4 to 12. The grid has size \(4096 \times 4096\), and the size of each data item is 8 bytes. The relative speeds of the participating processors were updated before each trial run, as were the estimated values of \(\lambda\) and \(\tau\). The values given for the observed data are averages. The sample size varied from 25 to 50 comparisons. 95% confidence intervals for these means are included.

The data were gathered on a non-dedicated, non-quiescent network. While this caused substantial variation in individual run times, this reflects the typical state of
Figure 43. The average ratios of predicted block versus row performance to observed block versus row performance. Two forms of the thermal transfer application were tested. The top graph shows the results when there are only 10 flops per data point per iteration. The second requires 50 flops per data point per iteration. The 95% confidence intervals for these averages are given.
computing in the networked environment. Applications running on the participating processors at the same time included other parallel programs, compiling sessions, graphics packages, and the usual collection of text-editing and e-mail programs.

For this test program, the predicted relative performance reasonably reflected the observed relative performance. There was a tendency, however, to predict slightly better performance for block decomposition than was borne out by the empirical evidence. It is believed that this is due to contention and collision resolution overhead in the CSMA/CD environment beyond that considered in the model. The block data decomposition method generates a greater number of communications than the contiguous row technique does. While there are fewer data items transmitted in block partitioning, the number of communications remains a critical factor.

7.3 Value and future directions for the advisory system

A general mathematical model of communication costs for networked parallel processing is of value to help determine the problem partitioning methods that will maximize performance. The goal has not been to model network behavior, but to establish general cost characteristics for various decomposition methods. This information may then be used by the programmer, compiler, or runtime system to make better informed partitioning decisions to minimize response time. For compile time use, approximations of $\lambda$, $\tau$, and the relative processor speeds may be substituted for the data generated by the first and third phases of the system.

While the predicted relative performance closely reflects observed relative performance in the validation testing, the effects of network contention and collision resolution should be more accurately expressed. These delays are dependent upon the particular network technology, and should be treated independently in the model. The advisory system currently considers only a single communication pattern. Some applications experience phase shifts where the patterns and computational require-
ments change during execution, and the system should be enhanced to consider a collection of patterns. It is sometimes the case that better performance may be realized by using fewer processors. When additional processors have low performance capabilities and the communication cost is very high, it may be more efficient to limit processing to fewer, faster nodes. Minor changes in the advisory system would allow calculation of various configurations to alert the programmer or user to this situation. Additional uses of the mathematical model include modification for assessing the efficacy of dynamic load balancing. The approximate achievable gain may be determined and weighed against the cost of data movement.
Chapter 8

Conclusions

The parallel network is a real high performance architecture, not an academic contrivance or a mere curiosity. The expanding interest in environments and tools for parallel processing in this environment attests to its increasing popularity. The improving speeds of microprocessors and networks, and the availability of machines have made this computing paradigm an attractive alternative to that offered by traditional parallel computers.

8.1 The Problem

As we have seen, there are specific costs associated with networked parallel processing. Low bandwidth, high message preparation latency, node heterogeneity, and changes in processor capability due to fluctuating workload all conspire to impede performance. Without adequately addressing these issues, the full potential of the system cannot be realized.

The research presented here addresses the problem of determining the decomposition methods most likely to maximize performance in the parallel network. New block partitioning schemes for decomposing uniform problem spaces across heterogeneous nodes have been developed. Different decomposition methods have been examined for their suitability to grid problems, and their costs have been mathematically characterized for a number of typical communication patterns. These characterizations have been used as the basis of an advisory system that, given information about the
network, the participating machines, and the problem space, recommends the most promising decomposition method.

8.2 Significance of this Research

While others have explored decomposition of data-parallel problems for traditional homogeneous multicomputers, none has done so adequately for the heterogeneous networked environment.

This research has shown that certain decomposition methods are unsuitable for uniform grid problems and should usually not be considered. It has also demonstrated that there is a point beyond which the savings in message size overshadows the additional communications associated with homogeneous block partitioning. A mathematical expression has been provided that enables the determination of whether or not this point has been reached.

The block decomposition algorithms developed here are the first ones that maintain rectangularity while accommodating heterogeneous processors. The rectangularity constraint not only reduces the size of the lookup table that is necessary to determine the processor location of an arbitrary data point, it also reduces the worst case number of communications. Bounds on the number of communications for these and other partitioning techniques are established to facilitate comparisons.

The mathematical characterizations presented here provide a basis for making partitioning decisions for networked parallelism. These characterizations are adaptable to the new switched network technologies. The equations may also be modified to apply to the internal connections of more traditional multicomputers. The Meiko CS-2, for example, is a multi-user system where the capabilities of the individual nodes may change over time as their workloads fluctuate. Architectures such as these are candidates for heterogeneous approaches to data decomposition.
While the Decomposition Advisory System proposed here is prototypical, it has been tested on a selection of applications. These application programs have subsequently been executed across a varying collection of machines at a variety of network performance levels to yield empirical validation of the mathematical model. Results are promising, even with a dynamic and unpredictable communication medium such as the Ethernet with CSMA/CD.

The advisory system, once fully developed, may be incorporated into a compiler to do static data decomposition. If the average performance levels of the network and participating machines are known \textit{a priori}, there may not be a need for run-time partitioning. Information regarding the problem space may be provided by the programmer or by a tool that analyzed the program to determine its dimensions, data size, computational requirements, and communication patterns.

Systems that permit run-time decomposition can use the features of the advisory system that estimate the current bandwidth and message preparation latency of the network and the current capabilities of the participating processors to make more accurate partitioning decisions. Again, information regarding the problem space may be provided by the user or may have been already determined by some analysis tool or by the programmer.

This research is the first complete analysis of the decomposition issues peculiar to grid problems in the parallel network. The work presented here provides a basis for achieving acceptable performance levels in this parallel processing environment.

8.3 Future Directions

There are numerous future directions for this research. The mathematical characterizations can be modified to accommodate switched networks and truly heterogeneous networks (see Figure 44).
This work has considered only uniform grids. While this comprises a large set of scientific problems, ways must be found to address non-uniformity and irregularity.

The Decomposition Advisory System should be expanded to provide advice regarding the number and identity of processors that are expected to provide superior performance. It should also be enhanced to handle more than one communication pattern to accommodate those problems that experience phase shifts.

The advent of multi-user parallel computers such as the Meiko CS-2 that permit users to access the individual processors as if they were independent machines highlights the need to develop mathematical characterizations for heterogeneity in more traditional parallel architectures.

As distributed and networked parallel computing become pervasive, the critical importance of problem space decomposition in this environment will become even more evident. This area of exploration has just begun. These are just the first tentative steps toward the hypercomputer [26] of tomorrow.
Bibliography


