AFIT/GCS/ENG/99M-06

PARALLEL DATA MINING WITH THE MESSAGE PASSING INTERFACE STANDARD ON CLUSTERS OF PERSONAL COMPUTERS

THESIS

Lonnie P Hammack
Captain, USAF

AFIT/GCS/ENG/99M-06

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 1999	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE
Parallel Data Mining with the Message Passing Interface Standard on Clusters of Personal Computers

5. FUNDING NUMBERS

6. AUTHOR(S)
Lonnie P. Hammack, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
2950 P Street
WPAFB OH 45433-4514

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/99M-06

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dr Robert L. Ewing
AFRL/IFTA
WPAFB, OH 45433-7334
DSN: 785-7438 x3592 COMM: 937-255-7438 x3592

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Thesis Advisor: Dr. Gary B. Lamont, DSN: 785-3625 COMM: 937-255-3625, E-mail: Gary.Lamont@afit.af.mil

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE
A

13. ABSTRACT *(Maximum 200 words)*
Piles of personal computers (PoPCs) have begun to challenge the performance of the traditional Massively Parallel Processors (MPPs) and the less traditional networks of workstations (NOWs) as platforms for parallel computing. Large clusters of PCs have reached and at times exceeded the performance of modern MPPs at a fraction of the cost. Built with commodity components, these clusters can be constructed for about half the cost of a comparable NOW. The primary competing operating systems (O/S) in use on PoPCs are Linux and Windows NT. This thesis investigation compares the performance of an NT cluster with that of a Linux cluster, a NOW, and an MPP. A comparison of the MPI tools available for NT is also accomplished. These comparisons are made using the Pallas benchmark suite for MPI and a parallel data mining algorithm. This data mining technique, known as the Genetic Rule and Classifier Construction Environment (GRaCCE), uses a genetic algorithm to mine decision rules from data. Results from experimentation and statistical analysis have produced three important conclusions. First, NT clusters are viable, cost effective alternatives to Linux clusters, NOWs, and MPPs for parallel computing. Second, the two primary communication libraries currently available for NT-PaTENT MPI and MPI/Pro-are statistically equivalent in performance. Third, the parallel GRaCCE algorithm is capable of relatively good speedup and efficiency, even for significantly unbalanced processor workloads, if the effects of first loop iteration caching are ignored.

14. SUBJECT TERMS
Parallel computing, Message Passing Interface (MPI), Piles of Personal Computers (PoPC), Massively Parallel Processors (MPP), Networks of Workstations (NOW), data mining, Genetic Algorithms (GA)

15. NUMBER OF PAGES
135

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

AFIT/GCS/ENG/99M-06

PARALLEL DATA MINING WITH THE MESSAGE PASSING INTERFACE STANDARD ON CLUSTERS OF PERSONAL COMPUTERS

THESIS

Presented to the faculty of the Graduate School of Engineering

Of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

Lonnie P Hammack, B. S.

Captain, USAF

March, 1999

AFIT/GCS/ENG/99M-06

PARALLEL DATA MINING WITH THE MESSAGE PASSING INTERFACE STANDARD ON CLUSTERS OF PERSONAL COMPUTERS

THESIS

Lonnie P Hammack

Presented to the faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

[illegible]

Dr. Gary B. Lamont
Chairman

[illegible]

Major Richard A. Raines
Member

Acknowledgments

This thesis effort would not have been possible without the tremendous support of my family, friends, advisors, and colleagues. I'd like to start by thanking my wife, Jennifer, and our two daughters, Christy and Tammy, for their encouragement and understanding. I promise to try over the next year to make up for some of the long hours I spent away from home during this time at AFIT.

Next, I'd like to thank my thesis advisor, Dr. Gary Lamont, for his ceaseless patience in assisting me to grasp the concepts of parallel and distributed computing. His noticeable interest in and astounding comprehension of this field of study made my task much easier. I'd also like to thank my academic advisor and committee member, Major Rick Raines, for his contributions to this effort. His classes were always enjoyable and enlightening. My fellow companions in the parallel-computing group also deserve special thanks for all of their assistance to me during this effort. Thanks also to Major Robert Marmelstein who provided the algorithm, which I used and was always willing to answer the many questions I had in working with his code.

Finally, I'd like to thank God with whom I have spent many hours in conversation concerning this effort. He provided the peace and assurance when things looked gloomy and I didn't know how to proceed.

Table of Contents

Table of Figures

List of Tables

Table of Equations

Abstract

Piles of personal computers (PoPCs) have begun to challenge the performance of the traditional Massively Parallel Processors (MPPs) and the less traditional networks of workstations (NOWs) as platforms for parallel computing. Large clusters of PCs have reached and at times exceeded the performance of modern MPPs at a fraction of the cost. Built with commodity components, these clusters can be constructed for about half the cost of a comparable NOW. The primary competing operating systems (O/S) in use on PoPCs are Linux and Windows NT.

This thesis investigation compares the performance of an NT cluster with that of a Linux cluster, a NOW, and an MPP. A comparison of the MPI tools available for NT is also accomplished. These comparisons are made using the Pallas benchmark suite for MPI and a parallel data mining algorithm. This data mining technique, known as the Genetic Rule and Classifier Construction Environment (GRaCCE), uses a genetic algorithm to mine decision rules from data.

Results from experimentation and statistical analysis have produced three important conclusions. First, NT clusters are viable, cost effective alternatives to Linux clusters, NOWs, and MPPs for parallel computing. Second, the two primary communication libraries currently available for NT—PaTENT MPI and MPI/Pro—are statistically equivalent in performance. Third, the parallel GRaCCE algorithm is capable of relatively good speedup and efficiency, even for significantly unbalanced processor workloads, if the effects of first loop iteration caching are ignored.

1 *Introduction*

1.1 From Supercomputers to Clusters of PCs

Starting with the very first computers, the demand for computing power has steadily outpaced the increase in computer performance. To meet this demand, supercomputer manufacturers began to take advantage of the power of multiple processors simultaneously solving the same problem. [Kumar94] This move to parallel computing began with the replacement of the typical single high-speed, high-cost, proprietary processor in supercomputers with a series of lower cost microprocessors connected by a high-speed interconnection network. The speed of these microprocessors over the last decade has typically been within one order of magnitude of the speed of the fastest serial computers. [Kumar94] This change has allowed supercomputer manufacturers to increase performance while maintaining or even reducing costs. Even so, the times required to architect and build these massively parallel processors (MPPs) meant that by the time they reached the consumer, the microprocessors were much slower than those currently being sold. To take advantage of this rapid increase in microprocessor speed, many users have turned to Networks of Workstations (NOW) for running parallel applications. Two important advances that made this possible were the increased performance of interconnection networks and the development of standardized tools and utilities for parallelizing applications. Interconnection networks with speeds of up to 100 megabits per second (Mbps) are readily available at commodity prices and gigabit speed networks, [GEA] which are available now at a much higher cost, are rapidly becoming affordable. For instance, gigabit Ethernet switches can currently be purchased for approximately $1000/port. Similarly equipped Fast Ethernet switches (100 Mbps) sell for roughly one-tenth of this cost or about $100/port. [Provant]

One example of a standard communication tool now available for parallel processing is the Message Passing Interface (MPI). [MPI] A broad base of commercial vendors, academia, and users worked together to develop this standard. Implementations of the MPI standard have resulted in the availability of a number of instantiations. One of the more popular is MPICH. [Gropp96] Developed by Argonne National Labs (ANL) in conjunction with Mississippi State University (MSU), MPICH is a portable version of MPI which works on most UNIX type operating systems.

Using MPICH, NOWs have achieved significant performance improvements over MPPs for such applications as Fast-Fourier Transforms (FFT) at a fraction of the cost. [Gindha97] [Anders95] Even so, users have begun to look at even more inexpensive ways of achieving the same performance. One way this goal has been accomplished is by using commodity personal computers (PCs) in what is commonly referred to as Piles of PCs (PoPCs) or Clusters of PCs (COPs). Since the idea behind using PoPCs is to reduce costs, a common operating system in use on these systems is Linux, [Linux] a UNIX-like operating system freely available, including source code, from the Internet.

A more recent move in PoPC research has been to make use of PCs running the Windows NT operating system. [Windows] One of the primary factors contributing to the use of clusters of NT workstations for parallel computing is the growing number of these machines in use, both in government, academic, and commercial organizations. This growth means that these machines are readily available for research and potential harnessing of idle processor cycles, known as cycle harvesting. Another driving force behind NT PoPCs is the development of tools for parallelizing applications that execute under the Windows environment. This development has been primarily focused on MPI. [MPI] Current research developments have lead to several versions of MPI for NT, which are discussed in detail in Chapter 2.

1.2 AFIT Bimodal Cluster (ABC) of NT/Linux PCs

To facilitate this research effort, a cluster of PCs has been installed in the AFIT Parallel lab. This cluster, known as the AFIT Bimodal Cluster (ABC), currently consists of a 200 MHz Pentium PC and four 333 MHz, six 400 MHz, and one 450 MHz Pentium II PCs connected via a 100 Mbps switched Ethernet network. These machines are dual-bootable under either Linux (Red Hat 5.0) or Windows NT (version 4.0). To avoid confusion, the name ABC-NT is used to refer to the ABC cluster running under NT. An Ethernet switch provides direct connections between every CPU. The research outlined in this thesis utilizes the ABC-NT and ABC-Linux cluster and compares the different MPI versions available for NT described in Section 2.3.3.

Since a comprehensive review of all possible parallel applications is not feasible, this thesis effort focuses on a single area of importance to the Air Force – Data Mining. The amount of data collected by the Air Force continues to grow at exponential rates. [USAFFS95] This growth has highlighted the need for effective means of partitioning this collected data into meaningful subsets for analysis. Current research in the field of data mining is providing some very useful techniques for accomplishing this goal. One such technique is discussed in the next section.

1.3 Data Mining Using Genetic Algorithms

Data mining is "the automated search for interesting and useful relationships between attributes in databases."[Marmel98] The field of data mining is within itself very broad. Researchers have developed many techniques to "mine" information from data including decision trees, neural networks, linear discriminants, genetic algorithms, and nearest-neighbor classifiers. [Weiss91] The primary application used for testing and experimentation in this thesis effort is based on a data mining algorithm developed by AFIT Ph.D. candidate Maj. Robert Marmelstein. This algorithm, known as the Genetic Rule and Classifier Construction Environment (GRaCCE), uses features of each of the aforementioned data mining techniques. Chapter 3 provides a detailed description of the original and modified GRaCCE algorithm.

1.4 Research Overview and Summary

Maj. Marmelstein developed the original GRaCCE code using MatLab. [MatLab] This algorithm produces a less complex rule set than traditional decision tree algorithms such as CART [Briema84] and C4.5, [Quinla93] with about the same accuracy; however, the performance of the original GRaCCE algorithm is much slower. One of the goals of this thesis effort is to convert this code to C++ and parallelize it using MPI. This code is then used to determine if parallelization of the algorithm eliminates the performance disadvantage as expected. The resulting code, along with the Pallas MPI benchmarks described in Chapter 4, is also used to evaluate the performance of the ABC-NT and Linux clusters and the major MPI implementations for NT currently available. To summarize, the objectives of this research are as follows:

1) *Compare the performance of an NT PoPC with that of a Linux PoPC.*

2) *Compare the performance of the various MPI implementations for NT (MPI/Pro & PaTENT MPI).*

3) *Analyze the performance of a parallel C++ version of the GRaCCE algorithm.*

These objectives are accomplished by performing a series of experiments and analyzing the collected data. The results documented in this thesis are written using the following assumptions about the reader. It is assumed:

1) That the reader has at least a general knowledge of the primary areas of the Computer Science/Computer Engineering discipline to include:
    a) *Computer architectures*
    b) *Computer operating systems*
    c) *Parallel and distributed computing*
    d) *General algorithms and algorithm complexity*
    e) *Computer programming*
2) That the reader has general knowledge of probability and statistics.

1.5 Outline of Chapters II through V

Chapter 2 provides the background on the emergence of PoPCs, the use of NT on PoPCs, and the tools available for use with these systems. Furthermore, an examination of some of the existing NT clusters is provided. This chapter also provides detailed information on data mining techniques and genetic algorithms. In Chapter 3, the algorithm domain for GRaCCE is discussed, as well as, various task decomposition and load balancing techniques. Chapter 4 lays out the methodology and experiment design for this research. Chapter 5 provides an analysis of the results of this thesis effort. This chapter is divided into three sections. First, a comparison of the performance of the ABC-NT PoPC with other parallel systems is discussed. Special attention is given to the comparison between NT and Linux PoPCs. Secondly, a comparison is made between the various MPI NT implementations. The advantages and disadvantages of each, as indicated by scientific experimentation, are discussed. Lastly, the performance of the parallel GRaCCE algorithm is compared to that of the serial version and the original MatLab code. Chapter 6 summarizes the results that were discussed in Chapter 5 and presents conclusions on the suitability of NT PoPCs for parallel computing. The various contributions provided by this research, as well as, recommendations for future research are also included in Chapter 6.

2 *Background*

This chapter provides limited background information on parallel systems. This information is necessary to understand the analysis of the experimental results given in Chapter 5. The chapter begins with a history in Section 2.1 of Massively Parallel Processors (MPP) and a description of the specific MPP used in this research. Sections 2.2 and 2.3 provide detailed information on Networks of Workstations (NOW) and Piles of PCs (PoPCs) and describe some of the current research in these areas. A description of AFIT's NT/Linux PoPC is provided in Section 2.4. The following section provides background information on data mining and some of the current research in this field of study. The final section provides a summary of this chapter. For a general overview of the principals of parallel processing, the reader is referred to Appendix A and [Kumar94].

2.1 Massively Parallel Processors (MPPs)

Rapid advancements in VLSI technology in the early 1980's led to lower prices and an increased demand for personal computers and workstations. As the volume of sales increased, the development costs were amortized over larger numbers of units, reducing the production costs further and driving even greater technological advancements. As PC sales outdistanced supercomputer sales by several orders of magnitude, the price-performance gap widened. Seeking to take advantage of these advancements in microprocessor technology, computer manufacturers introduced the first commercial MPPs in the 1980's. At peak efficiency, microprocessor-based computers such as Intel's Paragon XP/S and MasPar's MP-2 could exceed the speed of traditional single-processor supercomputers, such as the Cray Y/MP and the NEC SX-3. [Quinn94]

Modern MPPs may consist of tens, hundreds, or even thousands of microprocessors connected by a high-speed interconnection network. Because of the use of mass-produced commodity microprocessors, the cost of upgrading these systems is significantly less than that of the traditional supercomputers. Another major advantage of MPPs is that they can be built to achieve an absolute performance, which is unobtainable by a mainframe or supercomputer. Consider the following updated example from Tanenbaum's distributed O/S text: [Tanenb95]

> If 10,000 modern CPU chips, each running at 500 Million Instructions Per Second (MIPS), were used to build an MPP, it would have a total theoretical performance of 5,000,000 MIPS. For a supercomputer with a single processor to achieve this same performance, it would have to execute an instruction in 0.0002 nanoseconds (0.2 picoseconds), a feat which is impossible because of speed of light restrictions.

Other advantages of MPPs, according to [Anders95], are the communication performance and global system view. The high communication performance in MPPs is primarily due to the close proximity of the network interface to the processors. This interface is typically connected to the processor-memory bus, rather than the slower standard I/O bus. The global system view provided by MPPs allows users to run their applications on a large collection of processors as if it were a single entity. By using a global scheduler, the user is granted exclusive access to individual processors as they become available and doesn't have to be concerned with contention for resources.

Although MPPs have significant advantages, especially when compared to traditional supercomputers, they are still lacking in a few areas. A major disadvantage is the engineering lag required in developing the network hardware and proprietary operating systems used by these MPPs. This time constraint means that the microprocessors in MPPs are often a year or two behind the current technology. In addition, the cost of this development is significant when compared to commodity hardware and software prices. These disadvantages have made other parallel computing alternatives, such as NOWs and PoPCs, more attractive.

Although the primary focus of this research is a comparative analysis of NT and Linux PoPCs, some experiments are performed with an MPP and a NOW for additional insight into the differences between MPPs, NOWs, and PoPCs. The MPP used in this research is an IBM SP2 [IBMSP2] maintained by the Aeronautical Systems Command's (ASC) Major Shared Resource Center (MSRC) at Wright-Patterson AFB, OH. [MSRC] The MSRC SP2 is comprised of 256 135 MHz RS/6000 P2SC processors, of which 233 are available for batch computing. The compute nodes each have one gigabyte (GB) of available RAM memory. The interconnection network (ICN) for the SP2 is composed of two High-Performance Parallel Interfaces (HiPPI) with a maximum theoretical throughput of 800 Mbps. The topology of this ICN is essentially that of an *omega*[1] network.

2.2 Networks of Workstations (NOWs)

In the paper "A Case for NOW", [Anders95] the authors present several advantages of NOWs for parallel computing over traditional supercomputers and MPPs. Among those advantages, they cite an average price-performance advantage for workstations, which is a factor of two higher than a comparable supercomputer or MPP. In addition, the authors point out the availability of large amounts of aggregate DRAM in NOWs as a second advantage. In his Master's thesis, [Gindha97] Gindhart compares the performance of a network of Sun workstations with two MPPs – the Intel Paragon XP/S and IBM SP2. He concludes from his experiments that the NOW offers at least 85% of the performance of the MPPs for approximately 50% of the cost – a significant cost-performance advantage. Again, the large sales volume of workstations, as compared to MPPs, along with the faster processors normally found in NOWs, are the primary factors contributing to this price-performance advantage.

The NOW used by Gindhart was constructed at AFIT in October 1996. This system consists of four 175 MHz and two 200 MHz Sun Ultra workstations connected via a 1.28 Gbps Myrinet crossbar switch and a 10 Mbps Ethernet hub. Each of the workstations contains 128 MB of RAM, a 32 KB level 1 cache, a 512 KB level 2 cache,

[1] For more information of omega networks see Appendix A, Section A.1.3.1.

and two 1 GB local hard drives. Since the Myrinet ICN on this NOW has a much higher theoretical throughput (2.56 Gbps in full-duplex mode) than that of ABC's fast Ethernet switch (200 Mbps in full-duplex mode) and the processors are much slower, a completely objective comparison of the two systems cannot be accomplished. However, the Pallas benchmarks and parallel GRaCCE algorithm are run on this system to provide insight into:

a) *The effect of the ICN on the overall communication costs of a parallel system.*

b) *The effect of rapid technological change on parallel system performance, specifically the difference in processor performance in only a two year span between the construction of AFIT's NOW and ABC clusters.*

c) *The effect of messaging layers on the various types of Interconnection Networks.*

d) *The relative price-performance gap between PoPCs, NOW, and MPPs.*

e) *How using speedup as the only performance metric for an algorithm can be deceiving.*

Since PCs have an even higher volume of sales and thus lower unit costs and have comparable processor performance to workstations, PoPCs become the next logical step in capitalizing on the cost-performance advantage. The next section discusses some of the common features of PoPCs and describes some of the current systems in use.

2.3 Piles of PCs (PoPCs)

2.3.1 Beowulf – The first PC clusters

Since the main objective for using PoPCs is to reduce costs, a common operating system in use on these computers is Linux. [Linux] A PoPC running Linux is commonly referred to as a Beowulf, named after the original system created at the National Aeronautics and Space Administration (NASA) in 1994. [Ridge97] Linux is a POSIX compliant operating system kernel that is freely available. The major advantage of this O/S, other than the fact that it is free, is that the complete source code is available. This allows implementers to modify the O/S to best suit their needs for such reasons as I/O driver and messaging layer optimizations.

One of the disadvantages of Linux is the lack of technical support. There are numerous technical books and magazines dedicated to Linux, as well as mailing lists/users groups, [LUGR] which provide some useful troubleshooting information; however, there are no help desks to call, as is the case with commercial software, if one can't resolve a given problem with a "free" version of Linux. A commercial version of Linux is available from Red Hat Software, Inc. [RedHat] Users can purchase technical support with this software at an additional cost.

Another disadvantage of Linux is the lack of available applications that run under the Linux O/S. The number of applications for Linux, though still relatively small, is growing rapidly, due in part to recent announcements from various major computer manufacturers that they will offer Linux as a pre-installed alternative on their systems.

2.3.2 NT Clusters

A more recent move in PoPC research has been to make use of PCs running the Windows NT operating system. [Windows] NT is a commercial, fault tolerant, 32-bit operating system developed by Microsoft Corporation. It supports pre-emptive multi-tasking and threading. It is also POSIX compliant and supports symmetrical multi-processing (SMP). POSIX compliance allows it to run on multiple platforms, including the Intel x86, IBM PowerPC, MIPS, and DEC Alpha. NT is sold as two separate products – NT Workstation (NTW) and NT Server (NTS). NTW is optimized for desktop computers where foreground applications receive the highest priority. NTS, on the other hand, is optimized for background applications and is intended for use on enterprise servers (e.g. to provide mail, file, or print services). Costs for the current version of this operating system start at around one hundred-fifty dollars (NTW price). So, cost is obviously not the driving factor for its use in PoPCs. One of the main factors is NT's current install base and growing popularity, as both an enterprise server and desktop operating system. Microsoft shipped more than 1.3 million copies of NT server in 1997, far outpacing even its nearest competitor Novell, which shipped nine hundred thousand units of NetWare that year. [Festa98] Hence, computers running NT are readily available.

2.3.3 MPI on NT

Another driving force behind NT PoPCs is the development of tools for parallelizing applications that execute under the Windows environment. This development has been primarily focused on MPI. [MPI] Current research developments have lead to the versions of MPI for NT listed in Table 1. Each of these versions is discussed in the sections that follow.

Software Name	*Developed By*
WinMPICH	Engineering Research Center (ERC) at Mississippi State University (MSU)
MPI/PRO^{TM}	MPI Software Technology Inc (MSTI)
WMPI	Department of Computer Engineering, Coimbra University, Portugal
PaTENT WMPI 4.0	Genias Software GmbH
MPI-FM/HPVM	Department of Computer Science, University of Illinois at Urbana-Champaign (UIUC)

Table 1: MPI software for Windows NT

2.3.3.1 WinMPICH and MPI/PRO

WinMPICH, also known as MPICH/NT, [ERC] is a port of MPICH for NT platforms. This software supports both shared and distributed memory architectures. The developers wrote the original WinMPICH libraries to explore threads in the device layer for communication, TCP/IP support was added later. [Baker98] MSU created two designs of the shared memory device code for WinMPICH, one design supports POSIX threads, the other uses polling. Published reports indicate that the threaded version consistently outperforms the polling version. [Hebert98] Because of a lack of funding, MSU no longer supports WinMPICH. The code has been licensed to MPI Technology Inc, which has produced a commercial version of this software called MPI/Pro™. [MST] AFIT has acquired an eight-processor license for this software, which is used in this research on our Cluster of NT Workstations, known as the AFIT Bimodal Cluster (ABC). (See Section 2.4)

2.3.3.2 WMPI and PaTENT MPI 4.0

WMPI is based on MPICH and is a full implementation of the MPI standard for Win32 platforms. WMPI also includes support for the ch_p4 device standard. [CH_P4] This standard defines the communication interface between workstations in a heterogeneous network. Ch_p4 support allows interaction between Windows 95/NT workstations running WMPI and UNIX machines running MPICH. Similar to WinMPICH, a commercial version of WMPI is also available – Parallel Tools Environment on NT (PaTENT) MPI 4.0. [PaTENT] The PaTENT software is loaded on the ABC cluster and used in this research.

2.3.3.3 HPVM

The goal of the High Performance Virtual Machines (HPVM) project at UIUC is "to deliver high-performance computing from distributed computational and network resources."[Chien97] The National Center for Supercomputing Applications (NCSA) in conjunction with the Department of Computer Science at UIUC has built a 256-node NT cluster of PCs using the HPVM software. HPVM 1.0 is a collection of high performance parallel computing tools, which includes the following: Illinois Fast Messages (FM), MPI-FM, FM-DCS (Dynamic Coscheduling), Put/Get-FM, and Global Arrays-FM. The

FM library was written for Myrinet networks and provides highly optimized, low latency messaging. [Pakin95] MPI-FM is a high performance implementation of MPI for NOWs with a Myrinet network, built on top of the FM library. [Lauria97] This software is also free and is loaded on the ABC cluster; however, due to the complexity of running this tool without the expensive queuing software required for the Java-based front-end, HPVM is not used in this research. [HPVM] [Platform]

2.4 AFIT Bimodal Cluster (ABC)

The PoPC used in this research was constructed and configured by the graduate students in AFIT's parallel lab. Details about ABC's hardware, software, and configuration are outlined in the sections that follow. For a diagram of ABC, see Appendix B.

2.4.1 ABC's Hardware Configuration

ABC is a cluster of personal computers, consisting of one Dell 200 MHz Pentium computer and one Dell 450 MHz, six Dell 400 MHz, and four Gateway 333 MHz Pentium II single-processor computers connected via an Ethernet switch. Each of these machines is housed in a tower or mini-tower case to allow for expandability. The I/O bus on the Gateways operates at 66 MHz; the Dell's I/O bus is clocked at 100 MHz. The computers are housed in a portable rack, which allows easy access to both the front and rear of the cases.

To narrow the focus of this research and facilitate analysis of the data collected from the experiments outlined in Chapter 4, only a homogeneous subset of the computers in ABC are used for this research. Since the 400 MHz PCs comprise the largest homogeneous group, these were chosen. Each of the Dell processors has 128 MB of 10 nanosecond (ns) Synchronous Dynamic RAM (SDRAM), one 8.4 GB SCSI hard drive for use under NT, and one 6.4 GB EIDE hard drive for Linux. These machines also have a 512 KB Level 2 (L2) cache, a 16 KB instruction and 16 KB data Level 1 (L1) cache. Future plans for ABC include increasing the number of nodes to 32 and possibly 64 computers, to include some Symmetrical Multi-processing (SMP) systems.

2.4.2 ABC's Interconnection Network

Initially the ABC cluster was connected via an 8-port 100 Mbps shared Ethernet hub. The aggregate network bandwidth of this hub (200 Mbps in full-duplex mode) was inadequate for any realistic distributed processing and ABC quickly outgrew the eight ports available. This hub was eventually replaced with a 24-port Intel Express 510T fast Ethernet switch, operating in full-duplex mode. The network interface cards (NIC) used in each of the computers are also full-duplex fast Ethernet. This configuration provides ABC with the equivalent of a crossbar network, providing a maximum theoretical throughput of 200 Mbps per channel between nodes. One disadvantage of this switch, which was discovered during benchmarking of the cluster, is that, as with the original hub, the maximum aggregate network bandwidth of 800 Mbps, although four times greater than the hub is insufficient. Tests using the Pallas benchmark suite show that the network can become saturated when using only six processors for high communication processing.

2.4.3 O/S and Software Tools on ABC

Each machine is *dual-bootable*[2] under either Windows NT 4.0 or Linux 2.0.33 operating systems. Parallel communication/programming is handled through MPI/Pro 1.2.3 or PaTENT MPI 4.09 for Windows NT and MPICH version 1.1.0 for Linux. [MST] [PaTENT] [Gropp96] Two of the NT computers are loaded with Microsoft's Visual C++ version 6.0 compiler. This compiler is used for the GRaCCE conversion/compilation. The Linux implementation provides a variety of compilers including the freeware applications GNU C, G77, and G++, as well as, a commercial Fortran 90 compiler – VAST/f90. [GNUGCC] [VAST/f90]

There are no really effective visualization tools for NT loaded on ABC, however, the Performance Monitor application, which comes with NT, and the Intel Device View application, which comes with the switch, provide some level of insight into the network performance. The Linux system is loaded with Upshot (part of the MPICH distribution) [MPI] and Vampir 2.0, an MPI performance analysis tool developed by Pallas. [Pallas]

[2] *Dual-bootable* implies that a computer is capable of "booting" or starting under either of two different operating systems.

2.5 Data Mining

2.5.1 Introduction

Large sets of data are normally not very useful unless they can be grouped into meaningful sub-sets. This partitioning of data is known as classification. A good example of a large body of data, which is relatively unusable without some type of classifier, is the data available on the World Wide Webb (WWW). The search engines that Internet "surfers" use daily provide a semi-effective form of classification. It is only semi-effective because although a user may be able to find data relating to a particular subject of interest, the specific information needed will still have to be extracted from a large set of unrelated/semi-related data. This type of classification is known as *text mining.* [PMSI]

Another form of classification, which is the subject of this research, is *data mining*. The distinction between data mining and text mining is in the form of the data being analyzed. As the name implies, text mining seeks to extract meaningful relationships from text. The Internet search engines accomplish this task in a number of ways, ranging from simple word counting to computer "world knowledge" techniques. Data mining, on the other hand, also, seeks to extract "higher" knowledge from raw data, but it does this through numerical processing of quantitative and qualitative data.[PMSI] In other words, the data being analyzed, if not already in numerical form, are assigned numerical values based upon some type of encoding scheme. It is noted that this distinction between data mining and text mining is not universally accepted, as some consider text mining just a more specialized form of data mining, while still others even consider classification a form data mining. In addition, researchers have recently proposed a new technique, known as *web mining*, for applying the principles of data mining to Internet searches.

2.5.2 Description

Classification is a subject of interest to many different disciplines. In engineering, it is often referred to as *pattern recognition.* In computer science and artificial intelligence, it is also known as *machine learning.* It is not possible to discuss classification methods for computer systems without referring to learning systems. According to [Weiss91], "a learning system is a computer program that makes decisions based on the accumulated experience contained in successfully solved cases." From this definition we can then deduce that the objective of classification methods, based on the learning system approach, is to "learn" (i.e. instantiate a given model) from sample data, thus enabling successful classification and prediction on new data. [Weiss91] [Weiss98]

There are two general types of classification methods – unsupervised and supervised. [Weiss91] In supervised classification, the classifier uses a training set to "learn" how to classify data. This training set is a set of sample objects for which the classes are known. A human expert has pre-determined the choice of learning cases in this set. In unsupervised classification, there are no known cases. It is assumed that the sample data contains natural statistical groups of patterns that represent particular types of identifiable features. This type of classification is much more difficult and potential for success very limited. In this research, we focus on the following supervised classification methods: decision trees, neural networks, linear discriminants, and nearest-neighbor classifiers.

2.5.3 Decision Trees

Currently the most highly developed classification method is the decision tree. [Weiss98] A decision tree consists of nodes, which represent a single test or decision, and branches. For binary trees, the decision at each node may be true or false. This decision may also be whether a single parameter is greater than some constant. As with regular trees, the starting node is known as the root. As a feature traverses down the tree, it branches right or left based on the decision at each node, until it reaches a leaf node of the tree, which corresponds to a specific class. An example of a binary tree where the decisions are true or false is shown in Figure 1. [Weiss91]

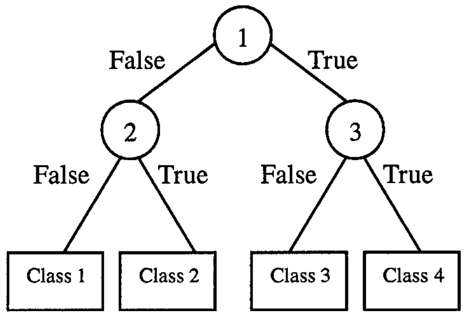


Figure 1: Binary Decision Tree

An advantage of the decision tree method is that it is usually much faster, both during the training and application phase, than many of the other classification methods. One disadvantage is that they are not as flexible at modeling complex distributions as either neural networks or nearest neighbor methods. Another disadvantage, which is discussed in more detail in Section 2.5.8, is that decision tree methods often produce overly complex classification rule sets.

2.5.3.1 CART

One of the most popular classification algorithms, based on the decision tree method, is the Classification and Regression Trees (CART) algorithm.[Briema84] CART, which uses binary decision trees for both prediction and classification, is used for "classification or regression analysis of large, complex data sets containing many variables."[UCLA] This algorithm recursively searches the sample data to produce an optimal set of decision nodes in an extremely large tree. The tree is then "pruned" of any branches that impair the overall accuracy. The result is the simplest tree that gives the maximum accuracy. The main advantage of the binary tree produced by CART is that its structure is easy to understand, interpret, and use. Even so, the rule set is still overly complex compared to that produced by GRaCCE. [Marmel98]

2.5.3.2 C4.5

Another decision tree algorithm, which was proposed by J. Quinlan in 1993 is C4.5. [Quinla93] Similar to CART, this algorithm generates a classification-decision tree for the given data set by recursive partitioning of data. It does this using a depth-first strategy. The algorithm constructs the tree by searching over all of the sample data, generating a set of possible decisions, and selecting the set of tests that produce the optimal classification. A single test is performed on attributes containing a discrete number of values. If the values of an attribute are continuous, then binary tests involving every distinct value are performed. [Joshi97] The C4.5 algorithm is not especially fast in comparison with other serial decision tree algorithms; however, a freeware parallel version, PC4.5, developed at New York University, is available for downloading from the Internet. [PC4.5] In addition, as shown in [Marmel98], the rule set produced by this method can be more complex than necessary for an accurate description of the data.

2.5.4 Neural Networks

Neural networks (or nets) are probably the most researched classification method today. They are loosely based on the dense neural connections of the human nervous system. The simplest neural net device is the single-output perceptron, which decides whether an input pattern belongs to one of two classes. As with linear discriminants, this decision is based on a weighted scoring function. A simple example of a single-output perceptron is illustrated in Figure 2. [Weiss91] In this figure, I1 and I2 are the inputs and W1 and W2 are the weights assigned to each branch. The weights of a perceptron are constants and are "learned" by training on the sample cases. This is done by evaluating each sample case sequentially and adjusting the weight if an output is incorrect.

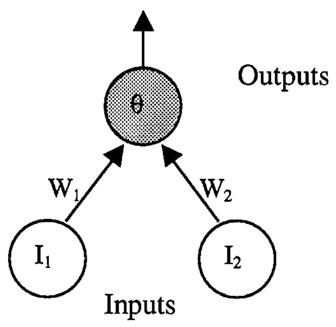


Figure 2: Neural Network Perceptron

The major advantage of neural networks is that they are general in nature. They can handle complex problems with a large number of parameters. Unfortunately, neural networks are very slow, especially in the training phase. [Weiss91] Another disadvantage is that it is difficult to determine the nature of the decision process in neural networks. This restricts their usefulness in the feature selection phase of data classification. [White97]

2.5.5 Linear Discriminants

Probably the most common form of classifier, linear discriminants are quite simple in structure. As the name implies, this type of classifier uses a linear combination of the evidence to separate (discriminate) among the classes and to select the class assignment for a new case. If the problem contains two features, then the classifier can be graphically depicted as a line (partition) between two classes or clusters. This is illustrated in Figure 3.

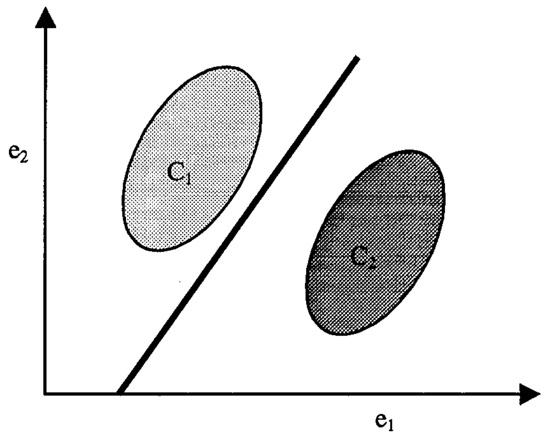


Figure 3: Idealized Class Separation by partition

Since a linear discriminant simply implements a weighted sum of the values of the observations, the equation can be written in the general form shown in Equation 1. [Weiss91]

$$w_1e_1 + w_2e_2 \cdots + w_de_d - w_0$$

Equation 1: Linear Discriminants

2.5.6 Nearest Neighbor Classifiers

This method uses the assumption that an object is most likely to belong to the same class as its nearest neighbor in the N-dimensional *feature space*. In reality, these algorithms don't use the single nearest neighbor, but a constant number, k, of nearest neighbors. Hence, this method is normally referred to as the k-nearest neighbor (k-NN) algorithm. This method is completely nonparametric, that is, nothing is assumed about the population. The class that appears most frequently among the k neighbors is chosen. To avoid the possibility of a tie, an odd number of neighbors is always chosen. The most commonly used measures of distance in this algorithm are absolute distance, euclidean distance, and various other normalized distances. [Weiss91]

Nearest neighbor algorithms are very easy to implement and can produce good results if the features are chosen carefully. They do however, have several disadvantages. First, like neural networks, they don't simplify the objects to a comprehensible set of parameters. Instead, they retain the entire training set as a description of the object distribution. Also, if the training set is large, this method is very slow. Lastly, the most significant disadvantage is their susceptibility to the presence of irrelevant parameters.

Even a single random parameter can cause misclassification of a large number of data points. [White97]

2.5.7 *Genetic Algorithms*

Although genetic algorithms are not classifiers per se, they have been used successfully to assist in feature selection and identification of partitions in linear discriminant classifiers.[PMSI] Genetic algorithms are optimization techniques based on Darwin's theory of natural selection. In a population, the best fit individuals survive and reproduce, while those least fit for the environment die off. Each individual is uniquely defined by the set of chromosomes, which is a subset of the union of those of its parents. In data mining, the chromosomes are binary numbers representing the parameters (traits) which describe an individual. Once a population has been established, a mechanism for evaluating the "fitness" of each individual must be devised. This is generally known as the fitness function. Individuals are selected to "mate" and produce offspring based on this fitness function in combination with some basic probability. It is expected that over time the entire population will evolve to a state of higher fitness. Other operators such as *mutation* and *inversion* are often used to emulate the randomness experienced in nature. The selection/reproduction/evaluation process is repeated over several generations until the population has converged to a relative fitness. [Whitle94] [Goldbe89]

In his 1989 text, Goldberg introduces a two stage GA, known as the Simple Genetic Algorithm (SGA). This algorithm starts with a *current population.* Selection is applied based on fitness to create an *intermediate population.* This is followed by recombination and mutation to create the *next population*. The pseudocode for Goldberg's original SGA is shown in Figure 4. The original version was written in Pascal. A version of this SGA, written in C, is used in the converted C++ GRaCCE algorithm. For detailed information on the SGA, the reader is referred to Goldberg's text [Goldbe89] and the GA tutorial from Colorado State University. [Whitle94]

```
1) Randomly generate initial population
2) Evaluate fitness of all population members
while i <= maximum generations AND
        stopping condition != TRUE
        3) Select Individuals
        4) Perform Crossover
        5) Perform Mutation
        6) Evaluate fitness of all population members
end while
```

Figure 4: Pseudocode for Goldberg's Simple Genetic Algorithm

2.5.8 GRaCCE

The Genetic Rule and Classifier Construction Environment (GRaCCE) [Marmel98] is an algorithm, developed with MatLab, [MatLab] for extracting classification rules from data. Developed at AFIT by Ph.D. candidate – Maj. Robert Marmelstein, it uses genetic search to initially select features from the data set. The fitness of each feature is based on the accuracy achieved against a kNN classifier. The GRaCCE algorithm takes a set of unstructured data and partitions the data into *class homogenous*[3] regions such as that shown in Figure 5. Each of the resulting partitions represents a classification rule. The primary advantage of this algorithm, as compared to some of the more popular decision tree algorithms such as CART [Briema84] and C4.5 [Quinla93] is that it is capable of producing a simpler set of classification rules with approximately the same or better accuracy. The GRaCCE algorithm is examined in detail in the next chapter.

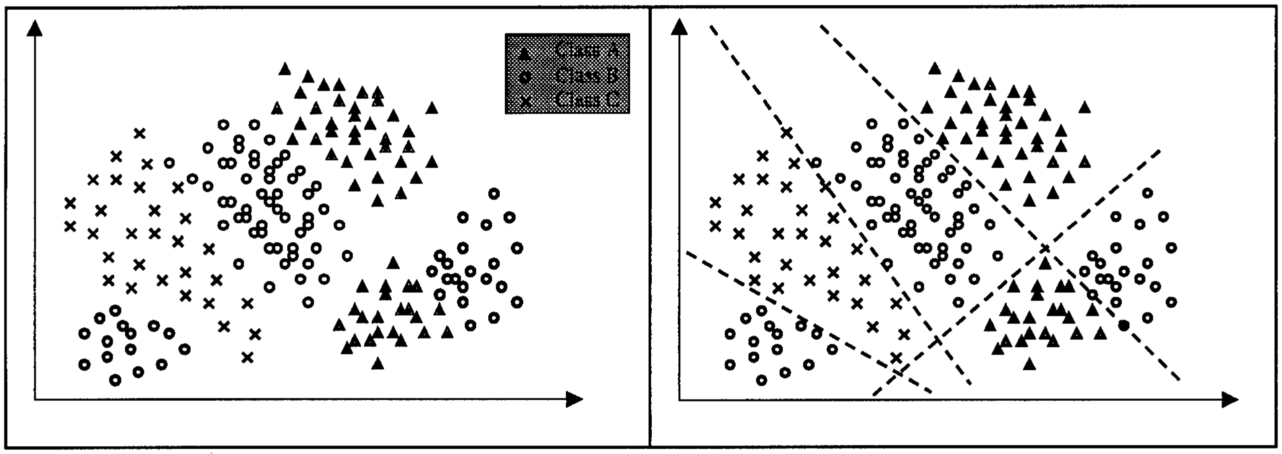


Figure 5: Data Before and After Partitioning

2.6 Summary

This chapter has attempted to provide the reader with a clear picture of the history of parallel computing that has lead to the transition from the traditional one-processor supercomputer to the commodity-based piles of PCs. Advantages and disadvantages of MPPs, NOWs, and PoPCs (Linux and NT) are discussed. Since one of the main objectives of this research is the evaluation of parallel communication libraries for NT, a

[3] Subsets of neighboring data in which the majority of the data points are from a single class.

considerable amount of discussion is spent on the ongoing research in this area. Lastly, the reader is presented with background information on data mining/classification techniques to include the general methods incorporated into the GRaCCE algorithm. This information is necessary to understand the decomposition of GRaCCE presented in the next chapter.

3 *GRaCCE Algorithm Design/Decomposition*

3.1 Overview

The first step in coming up with a parallel solution to a problem is understanding the problem domain, then, decomposing the problem into individual tasks and identifying which tasks can be executed concurrently. Next, the tasks must be mapped to an algorithm. Once this has been completed the algorithm must be decomposed and any further parallelizations identified. This chapter provides an in depth discussion in Section 3.3 of the GRaCCE algorithm, including task decomposition, task scheduling, and load balancing. However, before decomposing the algorithm, the problem domain is briefly discussed in Section 3.2. Section 3.4 records the decisions used in parallelization of GRaCCE and provides the pseudocode for the final parallel code. The chapter concludes with a prediction of the algorithm's performance in Section 3.5 and conclusions in Section 3.6.

3.2 Problem Domain

As mentioned in Chapter 1, a decision was made to use a specific real world algorithm — GRaCCE, in conjunction with the Pallas benchmark suite, for evaluating the performance of NT and Linux clusters and the MPI tools for NT. The problem, which this algorithm addresses, is *the extraction of accurate and understandable rules that define useful relationships between attributes in a database.* [Marmel98] This process is known as *data classification* or *data mining*. Several methods for attacking this problem, including decision tree, neural network, nearest neighbor, and linear discriminant algorithms, are discussed in the previous chapter. The general approach used by each of these methods is to separate the data into homogenous regions based on the *class* assigned to each data point. These regions are then used to define the rule set. The specifics of how GRaCCE accomplishes these tasks are defined in the next section.

3.3 Algorithm Domain

The current GRaCCE code is written in MatLab. Part of the focus of this thesis work is the conversion of this code to C++ and parallelization with the Message Passing Interface (MPI) standard. Several issues had to be addressed before this code could be parallelized. Most important was the issue of how to coordinate the GA-based searches to eliminate unnecessary work. The MatLab code as written completes each search sequentially. Boundary points are assigned to corresponding clusters at the end of each search. New searches are begun using only unclustered boundary points. Thus, in the serial version of the code, the majority of boundary points may never spawn a search. A direct assignment of each boundary point to a separate processor would cause a large amount of unnecessary work to be accomplished before a solution is found because redundant searches, which isolate identical clusters, would be run. This is illustrated in Figure 6.

In this figure, a search using boundary point b1 would generate the cluster enclosed by partitions p1 to p4. Likewise, searches using boundary points b2 to b4 would generate the same cluster. Once all of the searches are complete, the results would have to be integrated and the optimum solution gleaned from the set of solutions. So, the direct assignment approach doesn't appear offhand to be the most efficient.

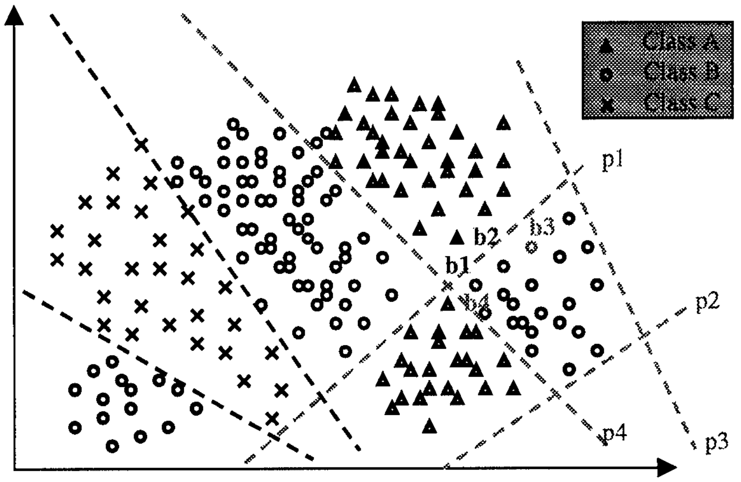


Figure 6: Redundant Cluster Identification

Another possible solution to this problem is to preempt searches as new information becomes available from other completed searches. That is, assign all boundary points (or sets of boundary points if enough processors are not available) to individual processors, and as searches complete, use the information from these searches to determine which boundary point searches are unnecessary. Once determined, the affected boundary point search could be preempted or removed from the processing queue if the search has not yet begun. This of course, would require some type of *dynamic load balancing*, such as the methods described in [Yang90], to ensure that some processors are not idle while others have heavy queue loads. This issue is explored further in the sections that follow.

3.3.1 Data Decomposition

At a data decomposition level, GRaCCE can be divided into the following phases: feature selection, partition generation, data set approximation, region identification, and region refinement. [Marmel98] provides a good description of each of these phases. These phases can be further decomposed into the following tasks:

T1. GA-based feature selection - selects the best m features
T2. Winnowing process – remove all points misclassified by kNN classifier
T3. Estimate class boundaries – use estimates to create partitions
T4. Compute weight for each boundary point
T5. Select target class ω_t which has not yet been evaluated
T6. Choose unassigned boundary point with greatest weight as focus of search
T7. Filter out partitions not related to the class of the chosen boundary point
T8. Measure partition distance to the boundary point
T9. Sort partitions on distance from boundary point
T10. Orient partitions such that boundary point, b, has a positive value
T11. Find initial solution using a greedy search technique
T12. Initialize GA population with results from greedy search
T13. Perform GA-based search
T14. Assign boundary points within best region found
T15. Filter out disproportionately small regions
T16. Test and remove extraneous boundaries
T17. Recompute the covariance matrix of each region

Tasks T6 to T14 are repeated for the remaining boundary points in target class ω_t. Once all boundary points in ω_t have been evaluated, tasks T5 to T14 are repeated for all remaining classes. This decomposition of tasks is graphically illustrated in Figure 5.

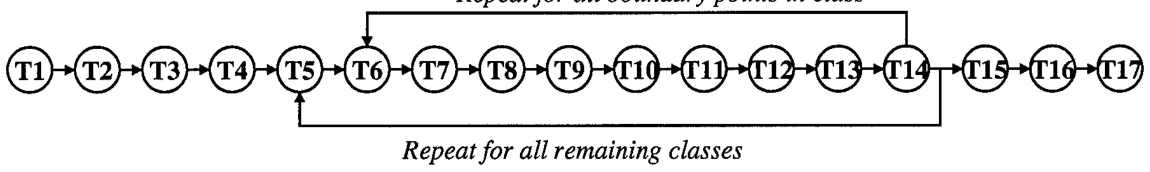

Repeat for all boundary points in class
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
T11
T12
T13
T14
T15
T16
T17
Repeat for all remaining classes

Figure 7: Data Decomposition of GRaCCE Algorithm

The region identification phase (T5-T14) is the best candidate for parallelization since each of the class evaluations are independent and all candidate partitions are known a priori. A proposed parallel decomposition of the above tasks is illustrated in Figure 8. This figure shows both the decomposition of searches for clusters of points belonging to a particular class and the separate searches associated with different classes.

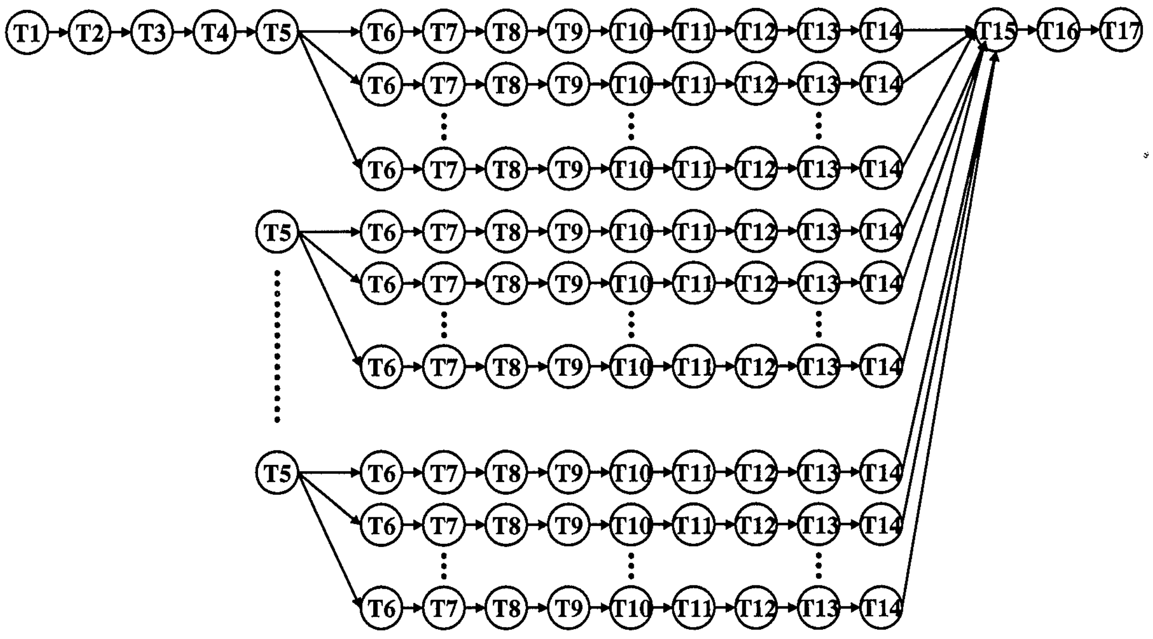


Figure 8: Parallel Data Decomposition of GRaCCE Algorithm

Further parallelization may be possible, by decomposing the GA-based search. The decomposition of tasks is as follows:

T12.	Initialize GA population
T13a.	Mutate population
T13b.	Evaluate fitness of population
T13c.	Assign fitness
T13d.	Select individual for breeding
T13e.	Recombine individuals
T13f.	Mutate individuals
T13g.	Evaluate offspring

T13h. Reinsert offspring into population

In this decomposition, tasks T13c to T13h are repeated as long as there is improvement in the population as compared to the results of the previous greedy search. If we parallelize this loop, we get the decomposition as illustrated in Figure 9. This level of decomposition would not yield significant improvements in overall performance. In reality, it may actually degrade the performance since the amount of interprocessor communication would increase by a factor of G/n, where G is the number of generations for each GA search and n is the number of processors.

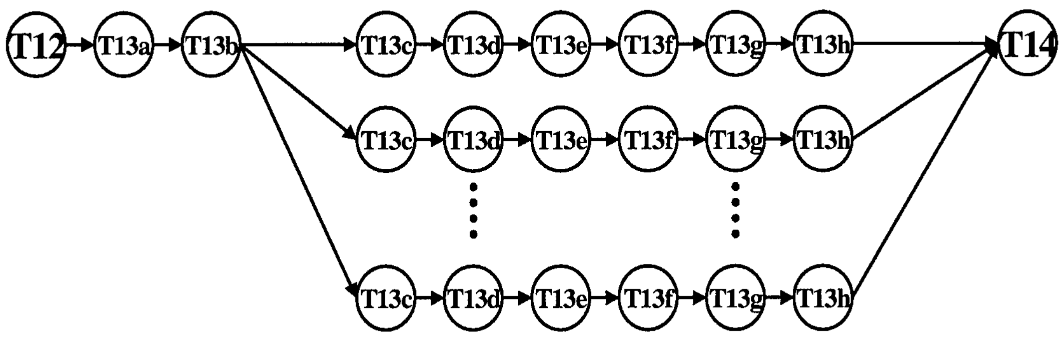


Figure 9: Parallel decomposition of GA-based search

3.3.2 Task Decomposition

As has been mentioned previously, the current code for the GRaCCE is written in MatLab. To limit the scope of this research to a feasible problem size, the conversion of the current code to the C++ programming language and subsequent parallelization is limited to the *region identification* section of the algorithm (i.e. tasks T5-T14). This section of the code accounts for approximately 90% of the application's execution time and offers the highest probability for improving the overall performance. The winnowed data set, list of generated partitions, boundary points and calculated boundary point weights used in the region identification phase are provided by the original GRaCCE algorithm. A preliminary review of the code for a serial conversion to C++ yields the pseudocode shown in Figure 10. This pseudocode is used in the decomposition of the algorithm that follows. It is noted that this pseudocode is not a complete algorithmic decomposition of the MatLab code. A complete decomposition is not necessary to discuss task distribution and load balancing.

```
Program cGRaCCE
Begin
 Get_data ();                              /* load data from output files
 For target_class = classmin to classmax   /* loop through all classes
  Get_target_class_data ();                /* load data for this target class
  While (unclustered_bpts > 0)             /* loop until all tc bpts have been clustered
    Curr_bpt = Max (bpt_set);              /* select bpt with the greatest weight
     For count = 1 to num_bnds             /* loop through partition set
      If (partition IN target_class)
        Add partition to boundary subset;  /* filter out unrelated partitions
        Compute Distance;                  /* measure distances to partitions
      End If
    End For
    Sort Distance_Array;                   /* sort partitions on distance from bpt
    For counter = 1 to ringer_size         /* greedy search
      Initialize ringer[counter];
      Calculate Value;
      Compare with Ringer Score;
      Assign Ringer Value & Score;
    End For
    If Ringer_Score < Search_Threshhold
      Initialize GA population;            /* initialize with greedy algorithm results
      Mutate population;
      Evaluate fitness;
    End If
    While (Generations < Min_Gen)          /* perform GA search
      Assign fitness;
      Select individuals for breeding;
      Recombine & Mutate individuals;
      Evaluate & Reinsert offspring;
      Update Score;
      Break if no improvement;
    End While
    If (GA_Sol > Ringer_Score)             /* Use best results of two searches
      Use GA results;
    Else
      Use Greedy Search results;
    End If
    Simplify Region ();                    /*remove unnecessary partitions
    Assign_Bpts ();                        /* assign appropriate bpts to cluster
    Update_Bpts ();                        /* rebuild bpt list from original data set
  End While                                /* all tc bpts have been evaluated
 End For                                   /* all classes have been evaluated
End Program;
```

Figure 10: Pseudocode for Serial GRaCCE algorithm

The primary control structures in the cGRaCCE pseudocode are the first "For" and "While" loops. The first "For" loop iterates through the set of classes for a data set.

Tests with various ordering of class evaluations have shown that the particular order doesn't affect performance or accuracy. This is understandable since class evaluations are independent with the exception of assigning higher priorities to previously used partitions. This exception does not affect accuracy or performance, but contributes to reducing the final rule set. Thus, for the serial algorithm, the classes are evaluated in the order in which they are read from the data structure. This ordering may become significant in the parallel version of the code, as the number of boundary points in each cluster affects the load balance on each processor. This of course, is only a factor if the tasks are allocated to processors based on class.

The "While" loop iterates through the boundary points within a cluster. Unlike class ordering, the order of the boundary point evaluations does effect performance and/or accuracy, even in the serial algorithm. By evaluating boundary points with the greatest weights first, other boundary points should be clustered more rapidly, eliminating the need for redundant evaluations of boundary points enclosed by the same partitions. For the parallel version, this brings us back to the problem of redundant cluster identification, discussed in the last section, and illustrated in Figure 6.

The other major sections of this pseudocode include the greedy search, GA search, and cluster refinement. These basically follow the previously discussed data decomposition. Although there are calculations, which are performed in the various sections of the algorithm, these calculations are trivial when compared with the data decomposition and potential communication costs. That is, an attempt to decompose this algorithm based on the lower level calculations significantly increases the amount of communication necessary between processors. Thus, this particular decomposition of the algorithm doesn't actually reveal a significant difference in an algorithmic decomposition as compared to the data decomposition, and so, we shall focus on the tasks previously identified.

Since it is evident from the serial pseudocode that the primary task decomposition for the parallel code remains the same, the parallel pseudocode can be written as shown in Figure 11. The two main loops are replaced with assignments of the tasks using a breakdown by class, boundary point, or a hybrid of the two, to available processors.

```
Program Parallel-GRaCCE
Begin
  Get_data ();          /* load data from files output by MATLAB GRaCCE
algorithm
  Broadcast_data ();    /* distribute to appropriate processors
  Assign_processor;     /* assign each class/boundary point to a processor

  Execute Code          /* execute code used by serial algorithm on each processor

  Gather_data;          /* gather data from each processor and analyze results
  Output results;       /* print solution to screen
```

Figure 11: Pseudocode for Parallel GRaCCE algorithm

3.3.3 Task Scheduling

Task scheduling for this algorithm depends on how the problem of the GA-search coordination is handled. As was mentioned previously, we could allow preemption of a task if a previously completed search has nullified the need for the current or scheduled task. If tasks are scheduled by boundary points, this could lead to a large increase in communication costs. If scheduled by class, then the communication requirements would be very low, but since the number of classes is normally much less than the total number of boundary points, this might significantly reduce the overall scalability of the algorithm. Each of these options is discussed in the sections that follow.

3.3.3.1 Scheduling by Class

If the boundary point evaluations for a particular class are all allocated to the same processor, then the results from each evaluation can be used to eliminate unnecessary boundary point evaluations without significantly increasing communication costs. There is, however, some increase in communication, as the partitions, which have been selected for a cluster, are given a higher priority for use in other cluster/class evaluations. This information must be shared with all other processes where the partition may be evaluated for use in bounding a cluster. Thus, there has to be some type of all-to-all broadcast, based on the inclusion of a partition in a particular class or boundary point partition set.

The major drawback with scheduling by class is the limit on scalability imposed by this approach. For instance, if a particular data set is composed of four classes of data with an average of sixteen boundary points per class, we could only use a maximum of four processors. By scheduling the same data set according to boundary points, we could possibly use up to sixty-four processors.

3.3.3.2 Scheduling by Boundary Point

When scheduling by boundary point, all boundary points belonging to the same class should be allocated to neighboring processors if possible to reduce communication costs. For instance, if we wish to schedule the tasks, listed in Table 2 below, onto the hypercube in Figure 11, we might end up with the scheduling shown in Table 3. As you can see, by trying to reduce communication, we may end up with a load imbalance among the processors. Thus, we must also, consider the ratio of computation cost to communication cost when developing a scheduling method.

	Class			
Boundary Points	ω1	ω2	ω3	ω4
	T1	T7	T14	T18
	T2	T8	T15	T19
	T3	T9	T16	T20
	T4	T10	T17	T21
	T5	T11		
	T6	T12		
		T13		

Table 2: Tasks (boundary points)

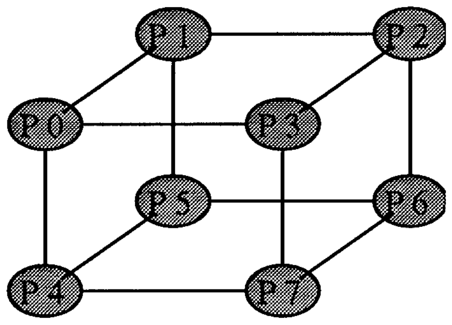


Figure 12: Processors in a Hypercube

	Processor							
Task	P0	P1	P2	P3	P4	P5	P6	P7
	T1	T4	T7	T10	T14	T16	T18	T20
	T2	T5	T8	T11	T15	T17	T19	T21
	T3	T6	T9	T12				
				T13				

Table 3: Scheduled Tasks for Hypercube

3.3.4 Load Balancing

As with task scheduling, load balancing also depends on the method of search coordination. It was illustrated in Table 3 that scheduling by boundary point with communication costs as the deciding factor could lead to a load imbalance. One possible method of avoiding this problem would be to use the duplication-scheduling heuristic (DSH) outlined in [El-Rew94]. Table 4 shows how tasks might be scheduled using DSH. Task T13 is duplicated to processors 6 and 7, which are neighbors of processors 2 and 4, respectively. Thus all of the tasks (boundary points) in class 2 now have access via neighboring processors to data produced by intra-class boundary points. This example does not follow the strict definition of DSH, which refers to duplicating a task on the processor (not neighbor) where the data is needed, but the same principle applies.

	Processor							
	P0	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**
Task	T1	T4	T7	T10	T14	T16	T18	T20
	T2	T5	T8	T11	T15	T17	T19	T21
	T3	T6	T9	T12	T13		T13	T13

Table 4: Scheduling using Duplication

One factor of scheduling by boundary point that was ignored in the above example was the fact that the boundary points within a particular class are evaluated based on their weight. To achieve this objective, it is necessary to use some type of list scheduling technique that schedules tasks based on a priority scheme. One such method is the Heavy Node First (HNF) technique outlined in [Sharaz95]. This technique basically assigns a weight to each task at each level. The heaviest nodes (tasks) are assigned to processors with the smallest accumulated execution time. The communication delay can also be factored into the decision of on which processor a task should be placed.

Both of these methods, DSH and HNF, are static list scheduling techniques. They don't address the issue of how to dynamically remove tasks that are no longer necessary or how to dynamically redistribute the load after previously scheduled tasks have been removed from a processor's queue. The first issue can't be directly handled by any of the generic dynamic scheduling schemes. It requires that some type of additional processing occur that decides based on data from completed tasks which tasks should be removed from the appropriate queues. This processing could be done on a central (i.e. supervisor) processor that would analyze the data broadcast by each processor, remove the appropriate tasks, and redistribute the workloads. This, however, would incur a high communication cost. A better method might be to let each processor analyze its own data and report to the supervisor the tasks that need to be removed. Since, this list of null tasks normally resides on the same or neighboring processor as the processor which is reporting the list, it may still be better to just allow the processor to remove the null tasks and only report their individual queue lengths to the supervisor. If this technique proves to be efficient, then it may be beneficial to take it one step further by instead using either the "LOWEST" or "THRHLD" algorithms discussed in [Yang90]. These algorithms should reduce communication costs even further while maintaining a better balance of workloads.

3.4 Parallel GRaCCE (cGRaCCE) Algorithm

After careful consideration of all factors involved, a decision was made to parallelize the outer loop of the GRaCCE algorithm (refer to Figure 10). That is, each processor is assigned a class or set of classes (if the number of processors is smaller than the number of classes) to evaluate for CH regions. The primary reasons for this decision are 1) low communication required, 2) more compatible with the size of clusters used in this research (maximum of 12 processors for ABC and 6 for AFIT NOW), and 3) less complex static scheduling scheme could be used. The pseudocode for the final parallel algorithm is shown in Figure 13. This pseudocode also shows the timings taken to compute the computation and communication ratios.

```
Program cGRaCCE
Begin
  Initialize MPI
  Record time (T1)     /* use MPI_Wtime to record starting time
  Get_data ();         /* load data from files output by MATLAB GRaCCE
algorithm
  Broadcast_data (); /* distribute to all processors
  Record time (T2)     /* used to determine I/O & communication overhead
  Assign_processor; /* assign class(es) to all processors

  Execute Code         /* execute code used by serial algorithm on each processor

  Record time (T3)     /* used to determine actual computation time
  Gather_data;         /* gather data from each processor and analyze results
  Record time (T4)     /* used to determine total execution time
  Output results;      /* print solution to screen
```

Figure 13: Pseudocode for final cGRaCCE code

3.5 Predicted Performance

The MatLab version of GRaCCE can execute in a matter of seconds on very small data sets; however, it can take more than a day to complete when using relatively large data sets. As mentioned previously, the rule set generally produced by GRaCCE is much simpler than that of CART or C4.5 and the accuracy is essentially equivalent. The performance (i.e. speed), on the other hand, is much slower than the competing algorithms. This is the primary driving factor behind converting and parallelizing the existing code. Since each class evaluation is essentially independent, one would expect an almost linear speedup. However, as shown in Chapter 5, this is not the case. The amount of work required to isolate all of the clusters for a particular class varies, thus, leading to load imbalances among the processors. In addition, the effects of caching and communication overhead contribute to a sublinear speedup.

3.6 Summary

This chapter has briefly touched on some issues concerning the data/task decomposition of the GRaCCE algorithm, as well as, issues concerning task scheduling and load balancing of this decomposition. Also, discussed are decisions that influenced the final parallel C++ algorithm. The next chapter presents the metrics and methodology used to analyze the performance of this algorithm, as well as, the performance of the NT and Linux clusters and MPI tools for NT.

4 *Methodology and Design of Experiments*

This chapter describes the tools and techniques used to evaluate the performance of ABC, the parallel GRaCCE algorithm, and the various MPI tools for NT. It begins with a general discussion of performance metrics for parallel systems in Section 4.1. Section 4.2 provides a description of the design for each of the three experiments used to analyze the aforementioned performance. The chapter concludes with a description of the statistical technique used to validate the results of these experiments.

4.1 Measuring Performance

Unlike serial algorithms, a parallel application can't be evaluated simple in terms of its execution time and input size. One must also take into consideration the number of processors used and the architecture of the parallel system. The combination of an algorithm and the parallel architecture on which it is implemented is known as a parallel system.[Kumar94] In comparison to serial programming, parallel programming is still a rather new field. Many of the tools/techniques for parallelization are still under development and are not always dependable. The MPI standard has helped to alleviate some of these problems, but it is still not uncommon for the development of an efficient parallel algorithm for a specific problem to take years.

Once a parallel algorithm has been developed, it must be tested using metrics that evaluate the degree to which parallelization has been reached. The most common metric is the parallel run time. Often abbreviated T_p, this is the time that elapses from the start of the program until the last process finishes. This metric is compared to the run time (T_s) of the fastest serial algorithm for the same problem to determine the speedup, efficiency, and isoefficiency of the parallel algorithm. These three metrics are discussed in the sections that follow.

4.1.1 Speedup

There is some ambiguity over the question of what is speedup. Most authors, however, agree on the following definition: speedup (S) is "the ratio of the run time of the fastest known serial program on one processor of the parallel system to that of the parallel program running on p processors of the parallel system."[Pachec97]

$$Speedup = \frac{Serial..run..time}{Parallel..run..time} = \frac{T_s}{T_p}$$

Equation 2: Speedup

This metric basically gives us an idea of how much performance was gained by parallelizing a particular algorithm. In theory, a program that runs in time T on a single processor could run in time T/p on p processors (i.e. p times faster). This is known as linear speedup.[Kumar94] In practice, however, the speedup is usually sublinear primarily due to the added communication overhead required to distribute the program to multiple processors and because programs normally contain sections of code which are inherently sequential and cannot be parallelized.

If the observed speedup is greater than p, known as superlinear speedup, this normally indicates that the serial algorithm used was not the fastest or, perhaps, that a greater portion of the problem domain fits within cache memory and does not have to be swapped to and from the hard disk. A stochastic search algorithm may also report superlinear speedup since the probability that a search reaches a solution in a fixed amount of time is greater when multiple paths are taken in the search. One can argue, however, that this again is not superlinear since a serial algorithm could be written to begin the search along the same path. This phenomenon of superlinear speedup was observed for certain trials of the parallel GRaCCE algorithm. The contributing factors are discussed in Section 5.4.2. Speedup alone doesn't provide an accurate picture of a parallel algorithm's performance. A look at an algorithm's efficiency is also necessary to complete the picture.

4.1.2 Efficiency

Although a parallel program may continue to experience an increase in speedup as the number of processors upon which it is run increases, the amount of this speedup eventually tapers off. This results from the fact that processors can not devote 100 percent of their time to computation, but must allot time to operating system tasks as well as communication requirements. Efficiency measures the benefit of adding more processors and is defined as the ratio of the speedup to the number of processors. [Kumar94]

$$Efficiency(E) = \frac{Speedup}{\#\,processors} = \frac{S}{p} = \frac{T_s}{pT_p}$$

Equation 3: Efficiency

Since, the speedup, in theory, can't exceed p, the efficiency, in theory can never exceed one. An algorithm is generally considered *scalable* if a fixed efficiency can be maintained as the number of processors is increased and the problem size is also increased. The rate at which the problem size must be increased to maintain a fixed efficiency varies among different parallel systems and determines the degree of scalability for that system. This rate is defined by the *isoefficiency function.* [Kumar94]

4.1.3 Isoefficiency

The isoefficiency function given in Equation 4 is simply a measure of the rate at which the workload W (i.e. problem size) must be increased for a particular system to maintain a fixed ("iso") efficiency. This function is dependent upon the overhead (T_0) incurred as a result of the parallelization. Scalable systems have small isoefficiency functions since the workload only has to be increased at a relatively slow rate to maintain a fixed efficiency. If a fixed efficiency can not be maintained no matter how fast the problem size is increased on a particular system, then that system is unscalable. [Kumar94]

$$W = \frac{E}{1-E} T_0(W, p)$$

Equation 4: Isoefficiency

4.2 Experiments and Benchmarks

In the article "Experimental Models for Validating Technology", the authors state that approximately 40 to 50 percent of the more than 600 published software engineering papers they reviewed contained no validations of the claims that were made. [Zelkow98] This thesis research makes no preliminary hypothesis on the outcome of the analysis included in this research; however, all resulting assertions have been backed up with experimental validation through statistical analysis of the collected test data. As was noted in Chapter 1, the three primary objectives of this research are to:

4) *Compare the performance of an NT PoPC with that of a Linux PoPC.*

5) *Compare the performance of the various MPI implementations for NT (MPI/Pro & PaTENT MPI).*

6) *Analyze the performance of a parallel C++ version of the GRaCCE algorithm.*

To meet these objectives, a series of three experiments are performed. Since one of the primary goals of experiment design is reproducibility, [Barr95] all of the parameters used in these experiments are outlined in the sections that follow. Detailed information on the configuration of all hardware and software used in these experiments is also provided.

4.2.1 ABC-Linux vs. ABC-NT

To analyze the overall performance of a parallel system, one needs to test the system with a variety of algorithms which are representative of the applications normally run on these systems. This, of course, is no small undertaking, as source code must be located, compiled, and tested for many types of parallel problems. A better approach is to run a benchmark suite that tests the parallel functions, which are most often used by these algorithms. The Pallas MPI Benchmarks (PMB) [Pallas] provides this functionality. The objectives of the PMB suite, as outlined in the accompanying User's Guide, are:

- Provide a concise set of benchmarks targeted at measuring the most important MPI functions.

- Set forth a precise benchmark methodology.

- Don't impose much of an interpretation on the measured results: report bare timings instead. Show throughput values, if and only if these are well defined.

This software is free and can be downloaded from the web at [Pallas]. This code has been loaded on ABC and compiled under NT using the MPI/Pro and PaTENT MPI libraries. It has also been compiled under Linux using the standard MPICH 1.1 library.

4.2.2 Benchmarking MPI Tools

The major advantage of using the PMB suite for evaluating the performance of the NT and Linux cluster is that it also allows evaluation of the MPI implementations for NT using the same collected data. Hence, no additional experiments were necessary to accomplish the second objective of this research.

4.2.3 Parallel GRaCCE Performance

In order to accurately analyze any performance improvement gained by parallelization of the GRaCCE algorithm, it is necessary to test the new algorithm on a number of data sets varying in the following characteristics: size of data set, number of classes, number of boundary points, and feature size (dimensionality). The parameters used for each test are outlined in Table 5.

Data Set Name	*Class*	*Dim*	*Data Points*	*Bnd Points*	*Bnds*	*Procs*	*Gens*	*Iterations / trial*	*Total Iterations*
Checker	2	2	1000	105	66	1,2	10,100, 1000	30	180
TH513	5	2	799	61	48	1,2,3,4 ,5	10,100, 1000	30	450
Glass	4	4	97	23	25	1,2,3,4	10,100, 1000	30	360
Cancer	2	9	523	17	11	1,2	10,100, 1000	30	180
Wine	3	3	130	21	17	1,2,3	10,100, 1000	30	270

Table 5: Parameters for Experiment III

All of the trials in Table 5 were repeated on ABC under NT and Linux using a variety of MPI tools, on the AFIT NOW, and on the MSRC's IBM SP2. The MPI software used on these systems is as follows: MPI/Pro 1.2.3 and PaTENT MPI 4.09 on NT, MPICH 1.1.0 on Linux, MPICH 1.0.13 for Myrinet on the AFIT NOW, a proprietary implementation of MPI on the SP2. The best results from the all trials were used to evaluate the performance of the parallel GRaCCE algorithm. These trials were also used to compare the performance of a real world application with that of the Pallas benchmarks using the same MPI implementations.

4.3 Statistical Validation

Analysis of experimental results without statistical validation of the collected data can lead to inaccurate conclusions based on faulty data. There are a number of methods for validating results, such as the tests of means, tests of variances, Bernoulli tests, Chi-Square tests, Empirical Distribution Function (EDF) tests, and the Analysis of Variance (ANOVA) tests. The method used in this research effort is the ANOVA test. One of the primary reasons for choosing this method is that it accomplishes the main goal of validating the experimental data, it is available in most standard spreadsheet applications, and the results are conclusive and easy to interpret. Microsoft Excel, which was used to create the performance charts for the analysis of cGRaCCE, provides the built-in ANOVA function that is used.

Since the ANOVA method assumes a normal distribution of data, a generally accepted "large" sample size of 30 trials [Allen90] is used to approximate a normal distribution. The ANOVA test is only applied to the analysis of the parallel GRaCCE performance. Statistical validation of the Pallas benchmark suite would have been redundant and is not necessary. For validation of the collected data, a 95% *confidence interval* (C.I.) is used. This is a commonly used C.I. for this type of experiment and is sufficient since the collected data is only used to provide close approximations of the speedup and efficiency of the parallel version of GRaCCE. These approximations are sufficient for determining the performance gain and scalability of the algorithm.

5 *Analysis of Results*

5.1 Overview

This chapter presents a detailed analysis of data collected from the experiments discussed in Section 4.2. This analysis is accomplished with regard to the three main objectives of this research, as presented in Chapter 1. The results of experiments with the Pallas MPI Benchmark suite and with the parallel/concurrent GRaCCE algorithm, referred to as cGRaCCE, are used in Section 5.2 to compare the performance of Linux and NT PoPCs. Using the same data, a comparison of the two primary MPI tools for NT clusters, MPI/Pro and PaTENT MPI, is accomplished in Section 5.3. An analysis of the performance of the parallel cGRaCCE algorithm concludes the chapter.

5.2 Linux Cluster vs. NT Cluster

This section compares the performance of the NT and Linux clusters. To maintain homogeneity and thereby reduce the complexity of this analysis, only the six Dell 400 MHz PCs in the ABC cluster are used. This comparison is based on an analysis of the data produced by the Pallas MPI benchmark suite. The results from the best runs of the cGRaCCE algorithm are also used to compare the results of a real world application with those predicted by the benchmarks.

Although the primary focus of this section is an analysis of the performance of NT and Linux clusters, comparisons are also made with data collected from similar experiments on the AFIT NOW and the MSRC's IBM SP2. It is noted, however, that the latter comparison is not intended to provide conclusive results about the performance of PoPCs versus NOWs versus MPPs in general. That type of comparison is beyond the scope of this research; however, a price-performance comparison of the specific platforms used in this research is performed.

The remainder of this section is divided up as follows. Section 5.2.1 details the factors that contribute to the general performance of each of the tested parallel platforms. In Section 5.2.2, optimizations for each of the different compilers used in this research are discussed. The results of trials with the Pallas benchmark are analyzed in Section 5.2.3. Section 5.2.4 compares the run time performance of cGRaCCE on each of the platforms. The following section provides a price-performance comparison of the ABC, AFIT NOW, and IBM SP2. In the final section, conclusions about the performance of Linux/NT PoPCs, NOWs, and MPPs are presented.

5.2.1 Factors Affecting System Performance

The primary factors affecting the performance of each of the systems tested include the processor speed and performance, memory size and speed, compiler optimizations, and ICN throughput and latency. A good indicator of processor performance is given by the Standard Performance Evaluation Corporation's (SPEC) CPU benchmarks. [SPEC] This benchmark suite includes a numerical rating of processor performance based on floating point operations (SPECfp95) and integer operations (SPECint95).

Compiler optimizations are discussed in Section 5.2.2. The remaining parameters are listed in Table 6. The SPEC benchmark rating is given as the ratio of the execution time of a processor to that of a reference machine. [SPEC] Thus, a higher number for the SPEC benchmarks indicates greater performance. One interesting parameter to note in Table 6 is the SPECfp95 rating for the SP2. Although the clock speed for the SP2's P2SC processor is approximately one-third that of the ABC's Pentium II processor, this benchmark indicates a 40% advantage in performance for floating point operations on the SP2. This higher rating is a due to the powerful pipelined floating point units in the P2SC processor, which are capable of executing up to four floating point operations per clock cycle. [IBMSP2] This provides the SP2 with a significant advantage for applications, which contain a large number of independent floating point operations.

Processor	*SPEC fp95*	*SPEC int95*	*Instruction/Data Cache Size (KB)*	*Level 2 Cache (MB/ns)*	*RAM (MB/ns)*	*ICN Speed (Mbps)*
Dell 400 MHz Pentium II (ABC)	12.4	15.3	16 / 16	512	128 / 10	200
Sun 167 MHz UltraSPARC (NOW)	9.06	6.26	16 / 16	512	128 / 60	2560
IBM 135 MHz P2SC (SP2)	17.6	6.17	32 / 128	None	1024 / 70	800

Table 6: System Parameters for ABC, NOW, and SP2

5.2.2 Compiler Optimizations

The C compilers used on each of the test systems for cGRaCCE are listed in Table 7.

Parallel System	*Operating System*	*Compiler*	*Optimization Used*
ABC-NT	Windows NT 4.0	MS Visual C++ v. 6.0	"Maximize speed"
ABC-Linux	Linux 2.0.33	GNU g++ 2.7.2.3	O3
AFIT NOW	Solaris 5.5.1	GNU g++ 2.7.2	O3
IBM SP2	AIX 4.1	IBM AIX xlC	O3

Table 7: C++ Compilers and Optimizations used

It was originally decided that the default optimizations for each compiler would be used. This proved to be a bad decision for the following reasons: 1) the default for MSVC++ is "maximum" optimization for speed and 2) the default for g++ [GNUGCC] and xlC is no optimization. This significantly impacts the performance of cGRaCCE on the platforms using g++ and xlC and makes it difficult to make a fair comparison between the NT cluster where the code was optimized and the other systems. Subsequently, it was decided to use the maximum optimization for each system. Figure 14 shows the performance difference between the unoptimized and fully optimized versions of cGRaCCE on the SP2, AFIT NOW, and Linux ABC cluster.

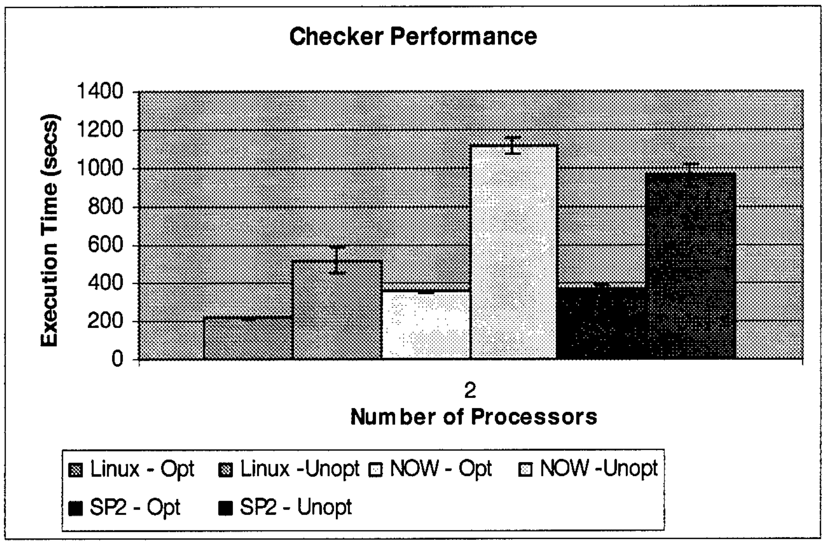


Figure 14: Effects of Compiler Optimizations on cGRaCCE Performance

As the chart shows, the performance more than doubles when the maximum compiler optimization is used. This increase can be attributed to the large number of optimizations used by modern compilers. For instance, the GNU g++, used by ABC-Linux and the AFIT NOW, uses the following optimizations: [GNUGCC]

- Automatic register allocation
- Common sub-expression elimination (CSE)
- Invariant code motion from loops
- Induction variable optimizations
- Constant propagation and copy propagation
- Delayed popping of function call arguments
- Tail recursion elimination
- Integration of in-line functions & frame pointer elimination
- Instruction scheduling
- Loop unrolling
- Filling of delay slots
- Leaf function optimization

- Optimized multiplication by constants
- The ability to assign attributes to instructions
- Many local optimizations automatically deduced from the machine description

5.2.3 Pallas MPI Benchmarks (PMB)

As was mentioned in 4.2.1, the PMB suite measures the performance of the most important MPI functions. It does this by measuring the latency of each function and the throughput for specific functions. The suite consists of eleven benchmarks as follows: 1) PingPong, 2) PingPing, 3) Sendrecv, 4) Exchange, 5) Allreduce, 6) Reduce, 7) Reduce_scatter, 8) Allgather, 9) Allgatherv, 10) Bcast, and 11) Barrier. These benchmarks are divided into the following three categories: single transfer, parallel transfer, and collective functions. It is not necessary to evaluate all of these benchmarks, as many provide redundant information. Therefore, in this section, we concentrate on three of the benchmarks, one from each of the three primary categories. The three evaluated are PingPong, Sendrecv, and Bcast. These benchmarks are well representative of the MPI constructs used in cGRaCCE, which only uses the MPI_Send, MPI_Recv, and MPI_Bcast functions for sharing data.

Given the information that was presented earlier in Table 6, one could make certain predictions about the expected performance of the four systems tested with PMB. One such prediction might be that the AFIT NOW would produce the greatest throughput since its maximum theoretical throughput is three to twelve times greater than that of the other parallel platforms. Of course, as is shown in the sections that follow, theoretical performance is not always a good measure of the true capabilities of a parallel system.

5.2.3.1 PingPong

The PingPong benchmark measures the *startup* and *throughput* of a single message in a network. It does this by sending a message back and forth between two processors and measuring the elapsed time (Δt). The *startup time* (t_s) is then reported as $\Delta t / 2$ µsecs. Using the message size (X), the *channel throughput* (ρ) for PingPong is calculated as follows:

$$\rho = X / 1.048576 / t_s$$

Equation 5: PingPong Channel Throughput

Both of these parameters are used in the sections that follow to analyze the performance of ABC-Linux, ABC-NT, the AFIT NOW, and the MSRC's IBM SP2.

5.2.3.1.1 Startup Time

The startup times for each of the parallel platforms in this experiment are shown in Figure 15. All of the Pallas benchmarks use message sizes ranging from zero bytes to four MB. Since there was no appreciable change in startup times from zero to 256 bytes for this experiment, the values for message sizes less than 256 bytes are not shown in this chart to enhance its readability.

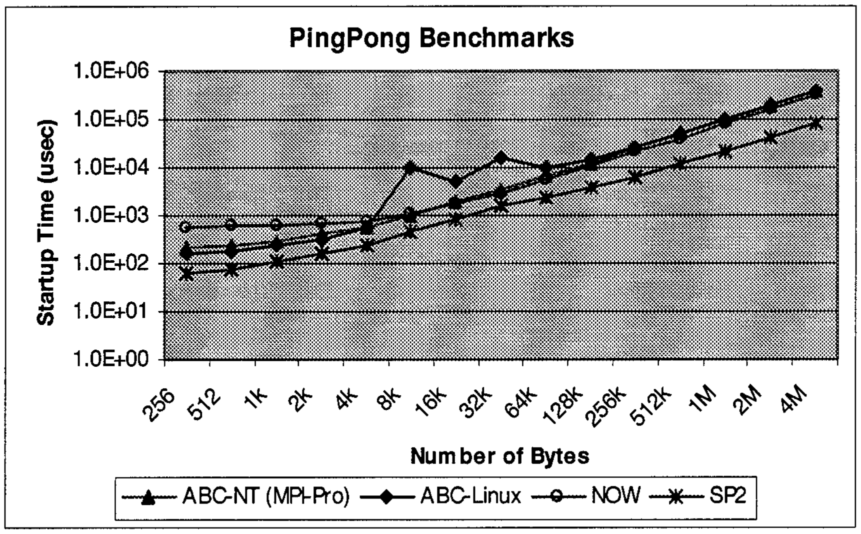


Figure 15: Message Startup Time on ABC, AFIT NOW, and SP2

From this chart, we can see that the SP2 clearly has the lowest startup time throughout the range of message sizes. The differences in startup time for the other platforms is close, with ABC leading the AFIT NOW initially for a 256 byte message size. These differences disappear as the message size increases to 4MB. The erratic behavior of the ABC-Linux platform as the message size changes from 4K to 64K bytes is caused by a problem with the TCP/IP stack on the Linux kernel used in these experiments. [NIST] This problem is discussed in more detail in the next section.

Since the primary focus of this section is a comparison of the NT and Linux cluster, a closer look at the experimental results is necessary. Figure 16 shows the startup times for the two clusters as the message size increases from 256 to 4096 bytes, the point at which the erratic behavior of Linux begins. This range of message sizes is fairly representative of the majority of messages sent by the cGRaCCE algorithm. The results indicate that the Linux cluster has a slightly lower startup time for messages than the NT cluster which leads to a higher throughput as shown in the next section.

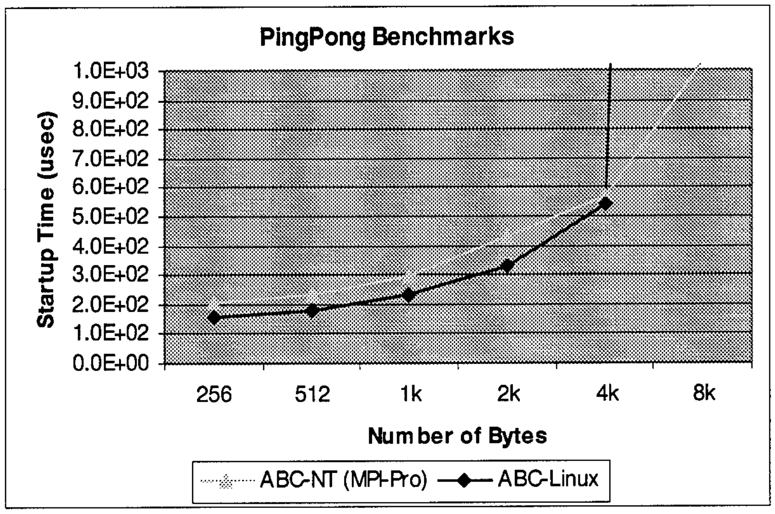


Figure 16: Startup Time for Linux and NT cluster

5.2.3.1.2 Channel Throughput

Figure 17 shows the measured channel throughput for each of the parallel platforms. As with the startup times, the change in throughput for message sizes between zero and 256KB is negligible and is not shown.

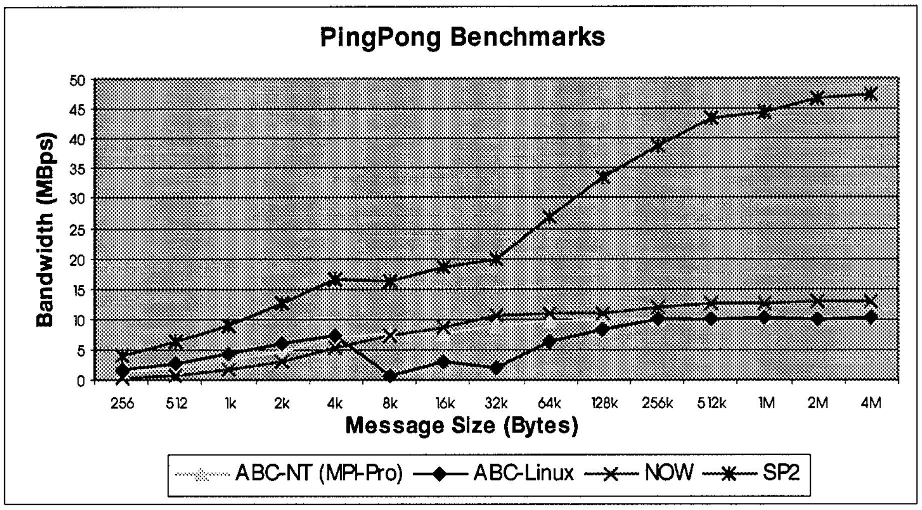


Figure 17: Channel Throughput - PingPong Benchmark

Once again, the SP2 outperforms the other platforms by a significant margin. As was mentioned at the beginning of Section 5.2.3, theoretically the AFIT NOW would be expected to have a much higher throughput than the SP2. This obviously is not the case and the reasons for this lack of performance can be found in [Gindha97]. Gindhardt explains how the network interface on the AFIT NOW's workstations is connected to the I/O bus, known as the SBus. This bus has an effective throughput of 23.9 MBps and thus becomes a bottleneck for the NIC. Gindhardt also points out the inefficiencies in using TCP/IP as the messaging layer for Myrinet. The latency with TCP/IP is at least an order of magnitude higher than with other "leaner" messaging layers such as Illinois Fast Messages (FM),[Pakin95] Berkeley Active Messages (AM),[vonEic92] and MSU's Bulldog Messages (BDM).[Henley97]

The same erratic behavior discovered in the startup time for the Linux cluster was again demonstrated in the throughput performance for messages ranging from 4KB to 64KB. This behavior is the result of a bug in the TCP stack for Linux kernels older than version 2.1.100. A full description of the bug, which is caused by delayed acknowledgements of partial packets, can be found at [NIST], along with a recommended solution. At the time of this writing, the fix had not yet been applied to the ABC-Linux cluster.

For an accurate analysis of the differences in performance of the Linux and NT clusters, it is once again necessary to take a closer look at the results of the experiment.

Figure 18 shows a more detailed chart of the throughput for the NT and Linux clusters. The TCP bug with Linux makes it difficult to compare the two systems; however, it does appear that for message sizes of 4K and below, Linux produces a slightly higher throughput. This comparison is analyzed further in the next section.

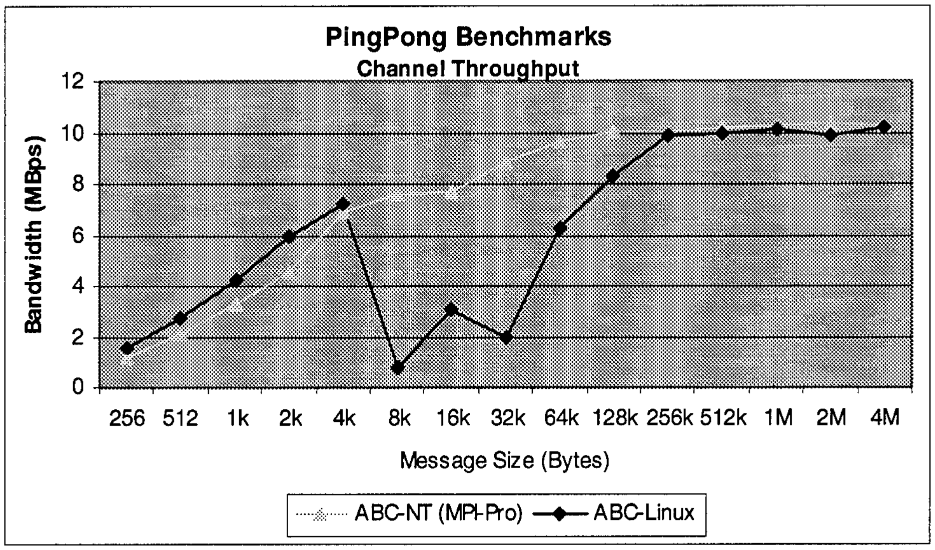


Figure 18: Channel Throughput for ABC-NT and ABC-Linux

5.2.3.2 Sendrecv

The Sendrecv benchmark, as the name implies, tests the performance of the MPI_Sendrecv function on a parallel system. This is a blocking function and is basically a concatenation of the MPI_Send and MPI_Recv functions. The Sendrecv benchmark organizes the processes into a periodic communication chain in which each node sends to the right and receives from the left neighbor in the chain. [Pallas] Since a particular process sends and receives X bytes in time Δt, the throughput is calculated as:

$$\rho = 2X / 1.048576 / \Delta t$$

Equation 6: Throughput – Sendrecv

The chart in Figure 19 again shows the SP2 outperforming the other systems by a significant margin. The AFIT NOW performed very poorly on this benchmark, which shows that high bandwidth is not necessarily effective without a corresponding low latency. The ABC-NT performed better than expected, outperforming the ABC-Linux cluster and the AFIT NOW for most message sizes.

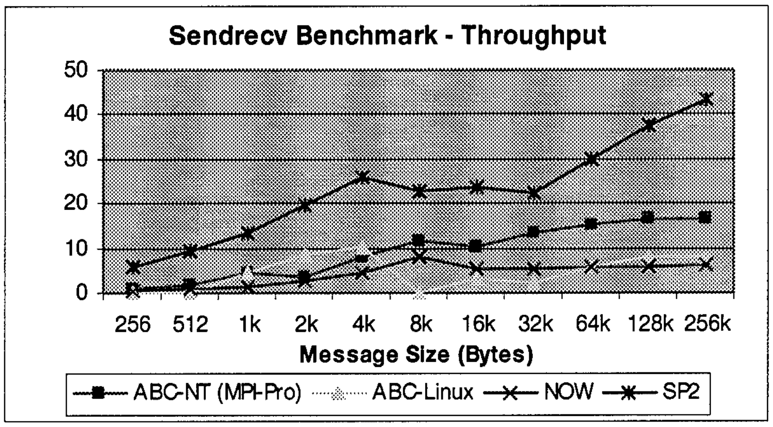


Figure 19: Measured throughput for Sendrecv Benchmark

5.2.3.3 Bcast

This benchmark measures the performance of the MPI_Bcast function. This function is used in cGRaCCE to distribute all of the data read in from the data files to each of the participating processors. The Bcast benchmark does not return a throughput value for MPI_Bcast – only bare timings are reported. A root process, which is changed cyclically, broadcasts an X byte message to each of the other processes. The results for this benchmark are shown in Figure 20.

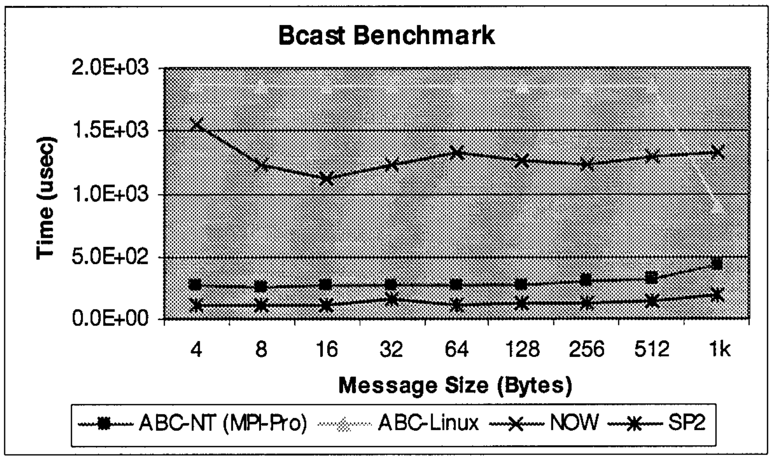


Figure 20: Timing for Bcast benchmark

Again, it is no surprise that the SP2 outperforms the other platforms; however, the performance of ABC-NT to that of ABC-Linux and the AFIT NOW is surprising. It appears from these results that the negative effects of both -- the TCP bug in Linux and the high TCP/IP latency on the AFIT NOW's Myrinet network -- are compounded by message broadcasts. In the next section, we show how these results and those previously discussed affect the performance of a real world application on these platforms.

5.2.4 Run Time Performance of cGRaCCE

As was mentioned in Section 4.1, the run time of a parallel algorithm is the time that elapses from the moment that a parallel program begins execution to the last processor finishes executing. For comparing computing platforms, run time is the best metric, since the time required to finish executing a program is normally the most visible and important parameter to the end user. In this section, we analyze the run time of cGRaCCE on each of the parallel systems using the TH513 and checker data sets. These results are representative of those observed with the other three data sets.

Before presenting the results, it is necessary to provide more info on the TH513 and checker data sets and a brief summary of what we've found thus far with the PMB suite. This information, which is presented in Table 8 and Table 9, should be very useful in our analysis of cGRaCCE's run time performance.

Data Set Name	*Integer Ops*	*FP Ops*	*Max Msg Size (KB)*	*Total Shared Data (KB)*	*Max % Comm*	*Ave Run Time (sec)*
Checker	6,563,800	42,564,100	6.77	23.28	0.34	517 ± 41.7
TH513	285,300	2,442,800	3.12	13.23	7.80	38 ± 0.67

Table 8: Measured statistics for Checker and TH513 data sets

			Throughput [4]		*Average*[5]	*Bcast*[6]	*Level 1*
Parallel System	*SPECfp95*	*SPECint95*	*PingPong (MBps)*	*Sendrecv (MBps)*	*Startup Time (usec)*	*Time (usec)*	*Cache (KB)*
ABC-NT	12.4	15.3	7.62	11.77	568.91	326.01	16 / 16
ABC-Linux	12.4[7]	15.3	7.18	10.19	544.34	1858.2	16 / 16
AFIT NOW	9.06	6.26	7.40	8.10	738.49	1290.31	16 / 16
MSRC SP2	17.6	6.17	16.58	26.07	235.58	148.12	32 / 128

Table 9: Performance data for test systems

Based on the information in these tables, we can make the following observations:

- *The SP2 has the highest floating point performance – approximately 40% greater than ABC and 95% greater than the AFIT NOW.*
- *The ABC has the highest integer performance – approximately 144% greater than the AFIT NOW and 148% greater than the SP2.*

[4] Measured for 8k message sizes or smaller – all messages sent out by cGRaCCE were smaller than 8k.

[5] Measured for 4k message size – average size for messages sent out by cGRaCCE.

[6] Measured at 512 byte message size – this is the largest message size within the range used by cGRaCCE for which Linux had a stable value (i.e. unaffected by TCP bug)

[7] No SPEC benchmark data were available for the 400 MHz PII running Linux; however, results for this processor with other UNIX operating systems indicated that the processor performance should be very close to the results for NT.

- *The SP2 had the highest throughput for the PingPong benchmark – approximately 125% greater than the other platforms, which were all roughly equivalent.*
- *The SP2 had the highest throughput for the Sendrecv benchmark – approximately 120% greater than ABC-NT, 155% greater than ABC-Linux, and 220% greater than the AFIT NOW.*
- *The SP2 had the lowest startup time – approximately 43% of ABC-Linux's, 41% of ABC-NT's, and 32% of the AFIT NOW's startup times.*
- *The SP2 had the lowest Bcast time – 45% of ABC-NT's, 11% of the AFIT NOW's, and 8% of ABC-Linux's Bcast time.*
- *The communication overhead of the TH513 data set is approximately 23 times that of the Checker data set.*
- *The total data size for the Checker data set (23.28KB) is too large to fit in the L1 cache of all of the parallel systems tested, except the SP2.*

With these observations in mind, let us now look at the run time results for cGRaCCE shown in Figure 21. Despite the SP2's superiority in network throughput, latency, startup time, broadcast performance, and floating point operations, it had the worst performance for all trials, with the exception of the one-processor run of the checker data set, where it barely outperformed the AFIT NOW. This exception was apparently due to the SP2's larger L1 data cache, which could accommodate the entire checker data set. It can only be deduced that the slower processor (135MHz) in the SP2 was no match for the faster processors in the AFIT NOW and ABC clusters. The lower communication overhead of the SP2, however, was very noticeable, especially for the TH513 experiment. In this experiment, the performance gap between the SP2 and the other platforms steadily decreases as the number of processors is increased from one to five. This is, of course, due to the lower rate of increase in the communication overhead for the SP2, as compared to the other systems.

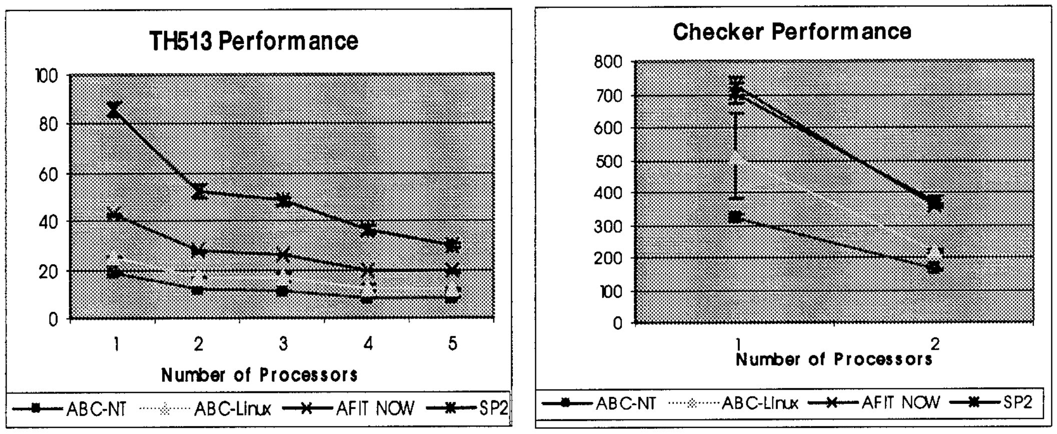


Figure 21: Run Time performance of cGRaCCE

The ABC-NT cluster outperformed all of the other systems for all data sets. The large difference in run times between ABC-NT and ABC-Linux with one processor using the checker data set indicate that the compiler optimizations of g++ on Linux were not as effective as those of MS Visual C++ on NT. Information discovered after these experiments were completed revealed that the particular version of g++ (2.7.2.3) used to compile cGRaCCE on Linux has not been optimized for the Pentium II processor. This may account for some of the difference in performance. An optimized version of GNU gcc/g++, known as Pentium GCC (PGCC) is available for free download from the Internet at [PGCC] and is recommended for use in future experiments with C++ code on Linux.

5.2.5 Price-Performance Evaluation

In this section, the estimated costs-per-node of the AFIT NOW, MSRC's IBM SP2, and ABC cluster are used for a rough price-performance analysis. A distinction is not made between the ABC-NT and ABC-Linux cluster in this section because they use the same hardware and there is essentially no cost difference. One might argue that the Linux kernel is free and therefore the ABC-Linux cluster must be less expensive than the ABC-NT cluster. This, however, is not the case, as the computers used in this cluster, like most PCs sold today, came with a choice of Windows NT or Windows 95/98 preloaded. These costs are then applicable to both systems, regardless of whether the system is booted under Linux or NT. Furthermore, recent announcements by Compaq, Gateway, and other PC manufacturers to offer Linux as a preinstalled O/S on their computers don't indicate any price savings for choosing this option.

The purpose of this section is to determine the price-performance ratio of the ABC, SP2, and AFIT NOW. This comparison does not attempt to determine the maximum cost-performance capabilities of each of these systems. This issue has been addressed by other researchers such as [Sterli98], where the author describes Beowulf systems that have achieved $30/Mflop sustained price-performance rates. [Anders95] presented some theoretical price-performance values for a NOW and comparable MPP of approximately $2900/Mflop and $12,700/Mflop, respectively, but these figures are somewhat dated. Algorithms such as cGRaCCE are far too complex to be used for this type of benchmarking. Therefore, in order to avoid any confusion over the data presented here, the cost-performance figures are only presented as ratios with ABC serving as the benchmark system with a rating of one. Ratings lower than one indicate that a system had a lower price-performance value than ABC and higher ratings indicate the price-performance was greater. These ratings are presented in Table 10.

Parallel System	*Original System Cost*	*Number of Nodes*	*Depreciated Cost/Node*[8]	*Price-Performance (th513)*	*Price-Performance (checker)*
ABC	$27,700	12	$1538.89	1	1
AFIT NOW	$104,000	6	$5135.80	0.133	0.135
IBM SP2	$10M[9]	256	$7716.05	0.057	0.086

Table 10: Price-performance for ABC, AFIT NOW, & SP2

In this table, two separate ratings are presented based on the two data sets evaluated. In both cases, the ABC has a clear advantage in the price-performance comparison by a factor of seven or higher. In [Bakerm98], the author also experienced similar results when comparing the performance of two dual processor 200 MHz Solaris workstations with that of two dual processor 200 MHz Pentium NT workstations.

5.2.6 Conclusions – Linux vs. NT Cluster

In the five previous sections, we compared the performance of an NT cluster, a Linux cluster, a NOW, and an MPP. The following general conclusions about the differences between PoPCs, NOWs, and MPPs can be drawn from this comparison:

[8] This value is based on average annual U.S. inflation rate of 3.6% and an expected effective life span of four years for computer systems due to technological advances, yielding a total depreciation of approximately 33%/yr.

[9] Although requested multiple times, costs for the MSRC's SP2 were not provided. This figure is based on the cost of a similarly equipped Paragon sold the same year as listed by [Anders95].

- *The performance of PoPCs is very competitive when compared to NOWs and MPPs.*
- *The rapid technological advances in PC technology and lower commercial costs give PoPCs a clear price-performance advantage over NOWs and MPPs.*

Due to the aforementioned TCP bug in the Linux kernel, it is difficult to make general conclusions about the performance differences between NT and Linux clusters. The original hypothesis was that the performance of the NT cluster would be slightly lower than that of the Linux cluster. This hypothesis was based on the assumption that a Graphical User Interface (GUI) based O/S, such as NT, would naturally have a higher overhead, and thus produce a lower performance than a leaner O/S, such as Linux. This, however, did not prove to be the case. In fact, the NT cluster outperformed the Linux cluster for all tests with the exception of the message startup time for a small range of message sizes and the PingPong throughput, also for a small range of values. These results, as previously mentioned, may be due in part to the Linux TCP bug and more effective code optimizations on the NT system. In general, however, it is safe to conclude that:

- *Clusters of NT workstations are viable alternatives to Linux clusters for parallel and distributed computation.*
- *NT clusters can perform as well or better than Linux clusters for computationally intensive algorithms.*
- *Differences in communication overhead for Linux and NT clusters are small and for the most part insignificant.*

5.3 Performance of MPI Tools on NT

In this section the data collected from the Pallas benchmarks and cGRaCCE trials, are again used to make performance comparisons. This time the comparison is between the different MPI implementations for NT: MPI/Pro 1.2.3 and PaTENT MPI 4.09. As mentioned in Section 2.3.3.3, HPVM which is the other major MPI tool for NT, is not evaluated in this research, primarily due to the expense of the queuing software required to use the Java-based front-end and the complexity of using this software without the front-end. According to [Baker98], the performance of HPVM on Ethernet is very poor compared to the other MPI tools for NT. This is primarily due to the fact that HPVM was designed for Myrinet networks. [Chien97]

5.3.1 Pallas Benchmarks

As with the comparison of the different parallel platforms in Section 5.2, the PingPong, Sendrecv, and Bcast benchmarks are used to compare these tools. These benchmarks had to be run multiple times with different versions of both MPI/Pro and PaTENT because of bugs in the software that surfaced under the heavy communication loads of the Pallas benchmarks. The technical support teams at MSTI and Genias used data from the experiments in this research to resolve some of these problems. Even so, a complete error-free run of the PMB suite with more than four processors was not possible. Fortunately, complete runs of the three benchmarks used for this evaluation with four processors were successful.

Information discovered after the experiments were complete indicates that at least part of the problem experienced in running these benchmarks may have been caused by the limitations of the switch used on the ABC system. According to the User's Manual for the Intel Express 510T switch, the maximum aggregate network bandwidth of this switch is 800 Mbps. When running the PMB suite with four processors, the maximum channel throughput produced was approximately 142.3 Mbps. Assuming an equal throughput on all channels, this is a total network bandwidth of approximately 569 MBps. Using six processors and assuming the same throughput, the total network bandwidth would reach 854 Mbps, exceeding the maximum capacity of the switch.

5.3.1.1 PingPong

5.3.1.1.1 Startup Time

As can be seen in Figure 22, MPI/Pro has a lower *startup time* for small message sizes. The difference is approximately 40% ± 2% for messages sizes up to 128 bytes and then decreases rapidly until it is completely negligible at 256k and above.

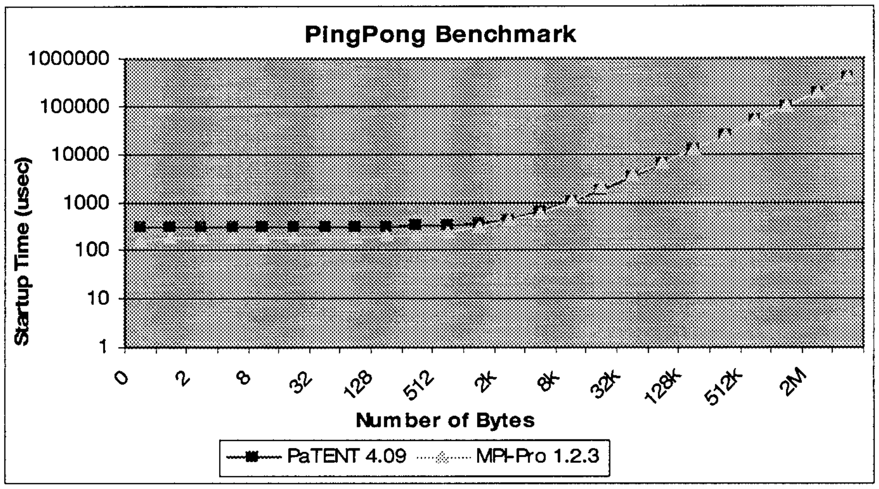


Figure 22: Startup Time for PingPong Benchmark

5.3.1.1.2 Channel Throughput

As shown in Figure 23, the results of the throughput measurements for the PingPong benchmark were somewhat unstable for MPI/Pro and failed to produce a clear indication of which tool was superior in channel throughput performance. These results are most likely caused by unresolved bugs in the MPI/Pro code. Even so, the results do indicate that the throughput of both packages is very similar.

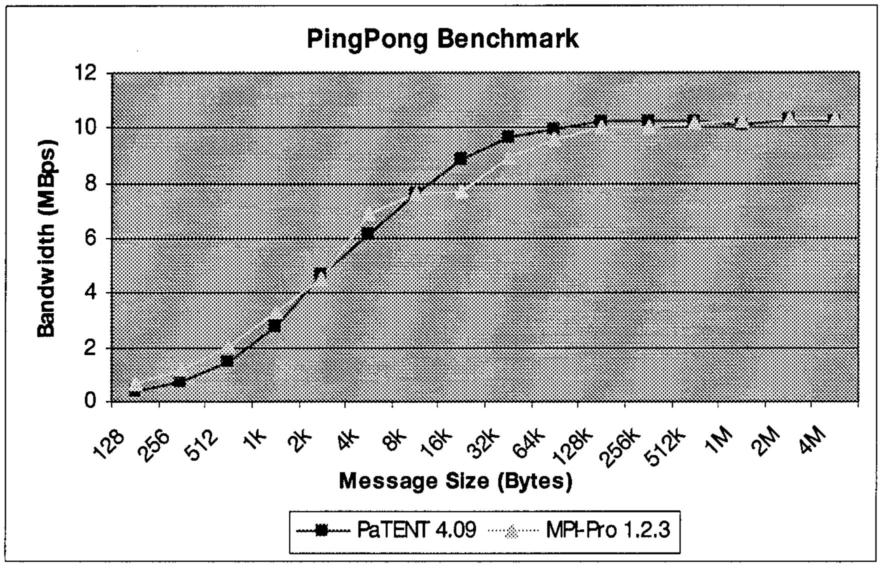


Figure 23: Channel Throughput for PingPong Benchmark

5.3.1.2 Sendrecv

As with the PingPong throughput results, the Sendrecv results, which are presented in Figure 24, showed very erratic behavior for MPI/Pro. Unlike PingPong, however, the PaTENT performance for Sendrecv began to drop significantly for message sizes larger than 64k, while MPI/Pro continued to experience increases in throughput up to a 1M message size. Again, we are unable to make conclusions about the performance difference between these two packages based upon the observed results.

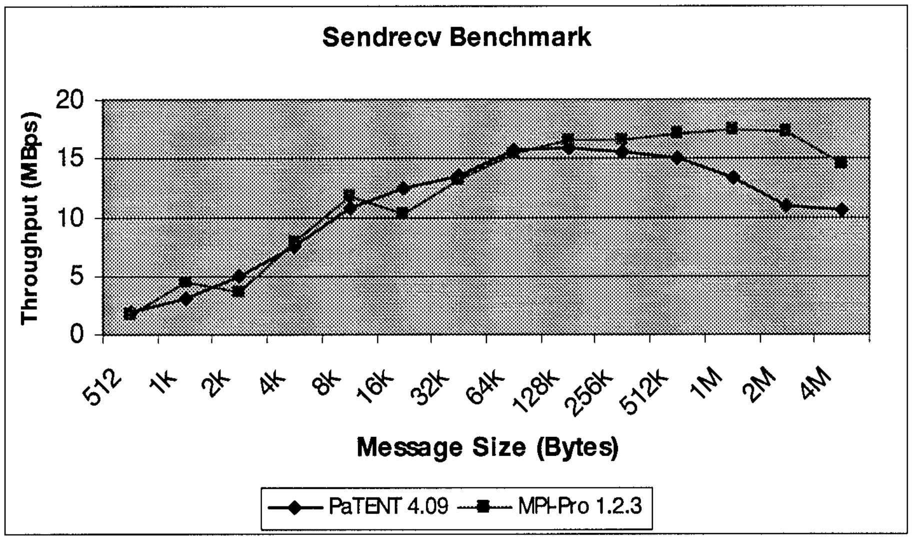


Figure 24: Channel Throughput for Sendrecv Benchmark

5.3.1.3 Bcast

The results for the Bcast benchmark, shown in Figure 26, are more stable than the results of the previous two sections. Once again, MPI/Pro starts out with a slight advantage of about 40%. This advantage continues up to a 512 byte message size and then rapidly decreases to an insignificant amount, except for a jump at 128k.

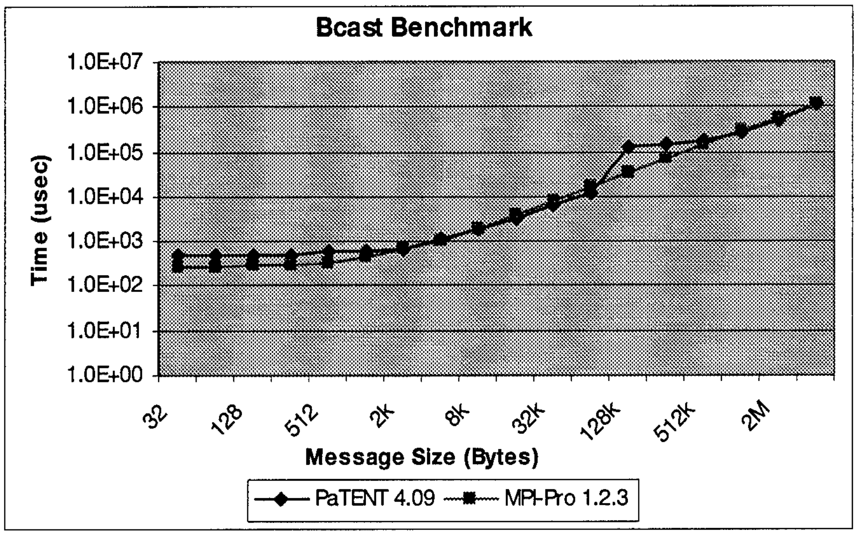


Figure 25: Timing for Bcast Benchmark

5.3.2 Run Time Performance of cGRaCCE

Because of its low communication overhead, the cGRaCCE algorithm was not effective in measuring the performance differences between the PaTENT and MPI/Pro parallel communication libraries. The results of trials with the cGRaCCE algorithm for both of these tools are shown in Figure 26.

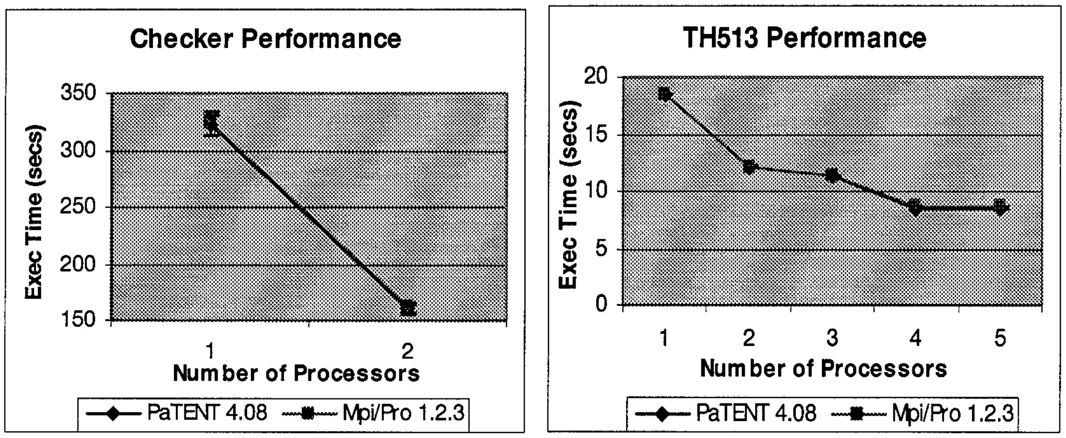


Figure 26: Run Time Performance of cGRaCCE with MPI/Pro and PaTENT

As shown in Figure 26, the performance differences appear to be negligible. A "test of means", using the ANOVA method, confirms this assumption for the checker data set and for the TH513 data set with up to three processors. The results of these tests are presented in Table 11 and Table 12. In both of these tables, the *test statistic* for the samples (i.e. MPI tools) does not fall in the *critical region* (F < F crit). Therefore, the *null hypothesis*, H_0, that the means are equal, must be accepted.

Anova: Two-Factor With Replication						
SUMMARY	1	2	Total			
PaTENT						
Count	30	30	60			
Sum	9664.8	4800.7	14465.5			
Average	322.16	160.0233	241.0917			
Variance	139.1308	31.14392	6767.16			
MPI/Pro						
Count	30	30	60			
Sum	9687.5	4846.3	14533.8			
Average	322.9167	161.5433	242.23			
Variance	188.5607	33.34254	6729.754			
Total						
Count	60	60				
Sum	19352.3	9647				
Average	322.5383	160.7833				
Variance	161.2143	32.28412				
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Sample	38.87408	1	38.87408	0.396494	0.530144	3.922878
Columns	784940.4	1	784940.4	8005.961	7.1E-109	3.922878
Interaction	4.370083	1	4.370083	0.044572	0.833163	3.922878
Within	11373.16	116	98.04449			
Total	796356.8	119				

Table 11: ANOVA values for ABC-NT with checker data set

Anova: Two-Factor With Replication			3 Processors	
SUMMARY	1	2	3	Total
PaTENT				
Count	30	30	30	90
Sum	553.32	364.4	337.53	1255.25
Average	18.444	12.14667	11.251	13.94722
Variance	0.000928	0.00174	0.001071	10.36053
MPI/Pro				
Count	30	30	30	90
Sum	553.27	365.05	338.04	1256.36
Average	18.44233	12.16833	11.268	13.95956
Variance	0.001225	0.007263	0.001113	10.30029
Total				
Count	60	60	60	
Sum	1106.59	729.45	675.57	
Average	18.44317	12.1575	11.2595	
Variance	0.001059	0.004544	0.001147	

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Sample	0.006845	1	0.006845	3.078472	0.081095	3.895451
Columns	1838.422	2	919.2108	413406.1	0	3.047901
Interaction	0.004573	2	0.002287	1.028406	0.359739	3.047901
Within	0.38689	174	0.002224			
Total	1838.82	179				

Table 12: ANOVA values for ABC-NT, TH513 data set, 3-processors

5.3.3 Conclusions – Performance of MPI Tools for NT

The results of this set of experiments produced no discernable performance difference between the two MPI tools for NT, other than a slight advantage in message startup times and broadcast performance by MPI/Pro for small message sizes. Errors encountered while running the PMB suite however indicate that there may still be unresolved problems with both packages, especially with MPI/Pro.

5.4 Parallel GRaCCE (cGRaCCE) Performance

In this section, the results of multiple trials of the cGRaCCE algorithm are used to analyze its speedup, efficiency, and isoefficiency. Discussed are the various factors contributing to the observed sublinear speedup and low efficiency of this algorithm in the majority of these trials. These factors include communication and I/O overhead, load balancing, and the effects of instruction and data caching. This section ends with a brief description of the results of the ANOVA tests used to validate the experimental results and a summary of the conclusions made from this analysis.

5.4.1 cGRaCCE Complexity

A good way of estimating the performance and scalability of an algorithm is by calculating the algorithm's complexity. This can be very difficult with stochastic algorithms. For instance, the number of computations performed by the cGRaCCE algorithm depends on the following factors:

1. Number of classes (c)
2. Number of boundary points evaluated per class (b)
3. Number of vectors (data points) in data set (d)
4. Number of features/dimensions (f)
5. Size of GA population (ρ)
6. Length of GA chromosome (L)
7. Number of generations per GA search (g)

The number of classes, data points, and features are always known and the size of the GA population is generally kept at a fixed value (100 for this research). The length of the GA chromosome varies but is bounded by the total number of partitions. The number of boundary points evaluated per class also varies and can not be determined based solely on the number of boundary points in the data set. It is dependent on the number of boundary points per class, the dispersion of the data in that class, and the probability that a boundary point will be enclosed in a region by a preceding boundary point evaluation. The equation for this number would be very difficult to derive, if at all possible. An overly conservative estimate of this number would be to use the total number of boundary points. The number of generations per GA search is also not constant, but is limited to twice the minimum number of generations. (See Section 3.3) This parameter (*mingen*) is provided by the user at run time. For this research, *mingen* is used to control the problem size for each data set and is set to 10, 100, and 1000. Therefore, with this information in mind, our complexity equation might look as follows:

$$Complexity = O(cB\rho gPf)$$

Equation 7: cGRaCCE Complexity

In this equation, B is the total number of boundary points and P is the total number of partitions/boundaries. Although the number of data points does affect the complexity, it is not used in this formula because the overall effect is somewhat complicated to determine and trivial in comparison to other factors. Using this equation, the anticipated total number of computations for each of the five data sets used in our experiments are estimated in Table 13.

Data Set Name	*Classes*	*Boundary Points*	*GA Pop*	*Max Gens*	*Partitions*	*Dims*	*Total Calcs*
Checker	2	105	100	20	66	2	55,440,000
TH513	5	61	100	20	48	2	58,560,000
Glass	4	23	100	20	25	4	18,400,000
Wine	3	21	100	20	17	3	6,426,000
Cancer	2	17	100	20	11	9	6,732,000

Table 13: Estimated number of computations performed by cGRaCCE

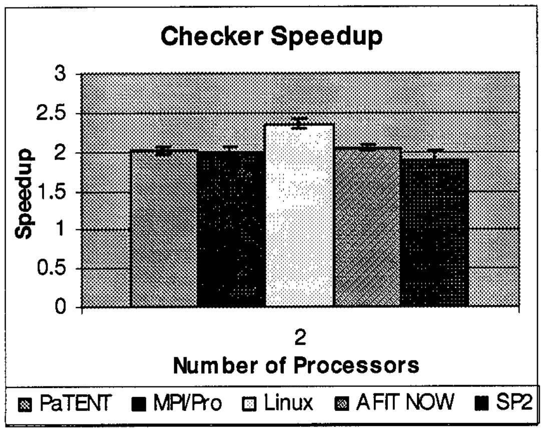
Table 14 provides a look at some rough measurements taken during execution of cGRaCCE with the five data sets and unfortunately shows that our complexity algorithm does not provide a very tight upper bound for all data sets. Although this equation provides a reasonable upper bound for the checker data set, the estimated number of operations for the other data sets are between six and twenty-two times larger than the measured values. This, again, has to do with the stochastic nature of the algorithm. With one data set the algorithm may evaluate ten percent of the boundary points in all classes, whereas, with a different data set, ninety percent of the boundary points may be evaluated. A detailed analysis of the complexity of the original GRaCCE algorithm can be found in [Marmel99].

Data Set Name	*Integer Ops*	*FP Ops*	*Total Ops*
Checker	6,563,800	42,564,100	49,127,900
TH513	285,300	2,442,800	2,728,100
Glass	231,900	1,307,400	1,539,300
Wine	161,700	858,200	1,019,900
Cancer	116,300	537,300	653,600

Table 14: Actual number of computations performed by cGRaCCE

5.4.2 Speedup

As was pointed out in Section 4.1.1, speedup is the ratio of the parallel and serial run times of an algorithm. Figure 27 below show the speedup of cGRaCCE for the checker and TH513 data sets. These data sets were chosen for this discussion because the first produced a slight superlinear speedup, whereas the other showed very little speedup. The speedup of the other three data sets used in these experiments fell somewhere in between. The primary factors contributing to both of these speedups are discussed in the sections that follow.



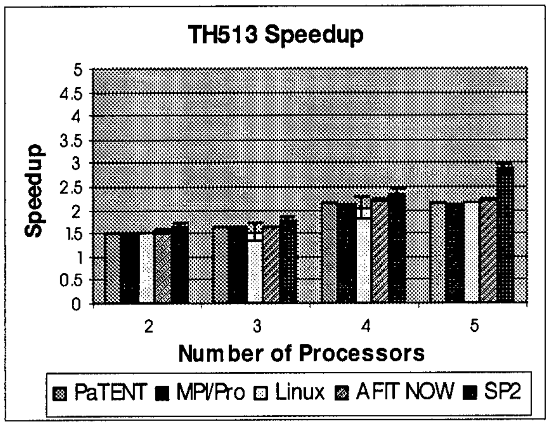


Figure 27: Speedup for Checker and TH513 data sets

5.4.2.1 Load Balance

To gain the maximum speedup, an algorithm must distribute the workload evenly among all of the computing nodes. This is known as load balancing. As was mentioned in Section 3.3.4, the two major categories of load balancing schemes are static and dynamic. It was decided that a static load balancing scheme would be used for cGRaCCE. This involved distributing out each class evaluation to a separate processor. This scheme required that data only be passed at the beginning (distribution of the data to each process) and at the end (gathering of results from each process) of program execution. Thus, no interprocess communication is required during the parallel execution of cGRaCCE and the communication overhead is very low for average or larger problem sizes.

This method, although successful in reducing communication overhead, resulted in large load imbalances for certain data sets. In these data sets, the number of boundary points and/or partitions evaluated for some classes is much larger than for others. The load balance for the checker and TH513 data sets are shown in Figure 28. As expected from the speed up results shown earlier, the load was almost evenly balanced with the checker data set. The TH513 data set, on the other hand, produced a very unbalanced workload with process one completing its work in almost a third of the time required for process four. Table 15 shows the division of boundary points and partitions evaluated for each class in the checker and TH513 data sets. As expected the distribution of work for checker is balanced, but, not so for TH513. One interesting observation from the charts and table is the fact that although processor five has the smallest work load for the TH513 data set, it took twice as long to execute as processor one, which had a slightly larger workload. This observation hints at other contributing factors, one of which is discussed in the next section.

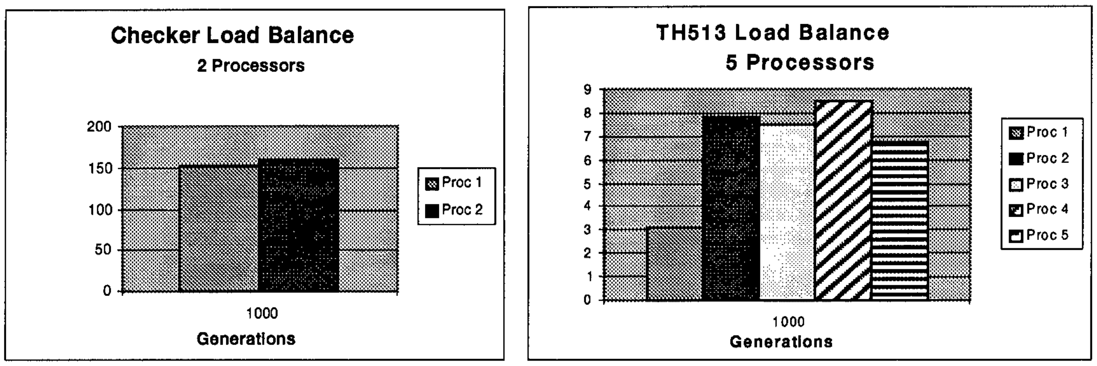


Figure 28: Load balance of cGRaCCE algorithm

Data Set Name	*Class Number*	*Bpts/Class*	*Bnds/Class*
Checker	1	25	52
	2	25	52
TH513	1	1	15
	2	1	23
	3	1	21
	4	1	23
	5	1	14

Table 15: Number of bpts and bnds evaluated per class

5.4.2.2 Cache Advantage

The load imbalance discussed in the previous section provided some insight into the sublinear speedup of cGRaCCE for the TH513 data set, but a more in depth analysis is necessary to determine other contributing factors. One such factor discovered by this analysis was the cache advantage experienced by running multiple iterations of the algorithm for the serial execution and for parallel executions in which the number of processors was less than the number of classes in the data set. For example, the Dell PCs used in these experiments each have on-chip Level 1 (L1) instruction and data caches. Whenever, the TH513 data set is evaluated using one processor, the five classes are evaluated in five iterations of the outer loop of the program. After the first iteration, the instructions have all been loaded in the instruction cache and the data have been loaded in the data cache. Hence, subsequent accesses to either are very fast. The effects of this data and instruction caching are shown in Figure 29.

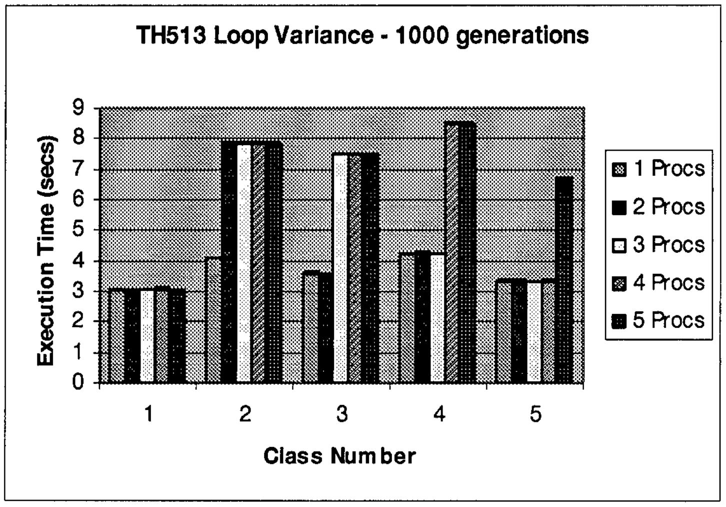


Figure 29: Cache advantage for TH513 data set

In this figure, the execution time for the evaluation of class 1 remains the same for one to five processors. This is because class 1 is always evaluated first and hence can not take advantage of cached instructions or data. Class 5, on the other hand, is evaluated last for one to four processors and uses cached instructions and data each time. When the number of processors is increased to five, this cache advantage is no longer available for class 5 and the execution time is approximately doubled.

Because of the negative effect caching has on the speedup of parallel programs in which multiple iterations of the main code are run by individual processors, it is a common practice, to discard the first iteration of each loop. This is not a problem in programs where the loop consists of hundreds or thousands of iterations and comprises only a small part of the overall solution. This unfortunately is not the case with cGRaCCE. Each loop is an evaluation of an entire class and a major part of the overall solution. However, the effects of caching become less noticeable as the overall processing time increases. This is the reason that execution of cGRaCCE with the checker data set is not significantly effected by caching. Evaluation of the checker data set takes approximately twenty times longer than that of TH513 for the same number of processors and generations. Figure 30 shows the predicted speedup for TH513 if the cache advantage were ignored. When compared to Figure 27 shown earlier, it can be seen that the cache advantage is very significant, much more than load balancing, for small data sets.

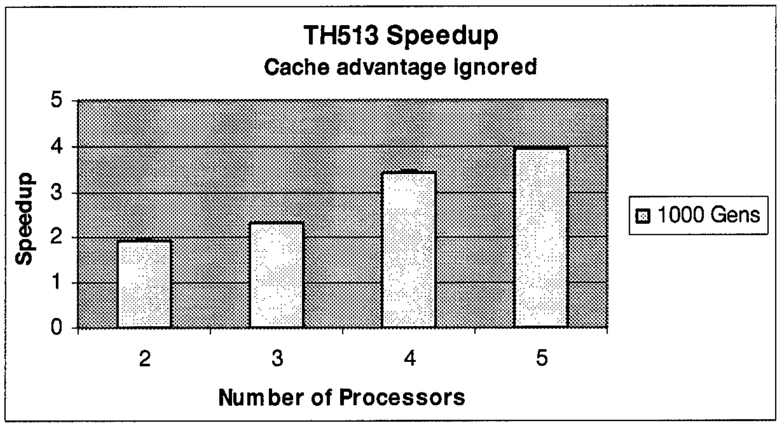


Figure 30: Cache advantage ignored for TH513 data set

We've looked at the primary factors affecting the sublinear speedup of cGRaCCE for the TH513 data set (coincidentally the speedup was also sublinear for the other three data sets tested), but what about the super linear speedup experienced with the checker data set. As was mentioned in Section 4.1.1, super linear speedup can usually be attributed to one of three primary factors as follows:

1) A sub-optimal serial algorithm
2) The stochastic nature of the algorithm such as a tree search in which multiple paths can be evaluated simultaneously to find the solution more rapidly
3) Division of the program data into smaller units such that it fits into memory that was previously too small.

In this particular case, the third factor applies. Table 16 shows the total size of the data that is passed to each processor for a particular data set. Table 17 shows the size of the Level 1 cache for each of the three systems used in this research. The data for all of the data sets, except checker, are small enough to fit into the L1 data cache on all of the systems. Since the L1 data cache on the SP2 is large enough to house the entire data set for checker, superlinear speedup should not be observed. This is indeed the case as is shown in Figure 27. As for the other systems, although the entire data set is passed to each processor, only a portion of these data is used for the majority of the program execution. This portion apparently fits into the L1 cache on the ABC and NOW. This allows faster access to the data and the evaluation of each class takes less time than on a single processor thus producing a super linear speedup.

Data Set Name	*Total Message Size (KB)*
Checker	23.28
TH513	13.23
Glass	3.13
Wine	2.52
Cancer	12.46

Table 16: Size of data passed for each data set

Processor	*Instruction/Data Cache Size (KB)*
Dell 400 MHz Pentium II (ABC)	16 / 16
Sun 167 MHz UltraSPARC (NOW)	16 / 16
IBM 135 MHz P2SC (SP2)	32 / 128

Table 17: Size of Level 1 cache

5.4.2.3 Communication Overhead

Although it has been noted that the chosen parallelization method for cGRaCCE significantly reduces the communication overhead, it can not be completely ignored. As with the majority of algorithms, the communication overhead for cGRaCCE increases as the number of processors are increased. This increase eventually leads to reduced performance and lower efficiency. This overhead is especially noticeable for smaller problem sizes, where the communication-to-computation ratio is high. In Figure 31 below, the speedup for the TH513 data set levels out at four processors for each of the systems tested, with the exception of the SP2. This is understandable since the throughput, startup time, and latency on the SP2, according to the Pallas benchmark results discussed in Section 5.2, is better than that of the AFIT NOW or ABC clusters.

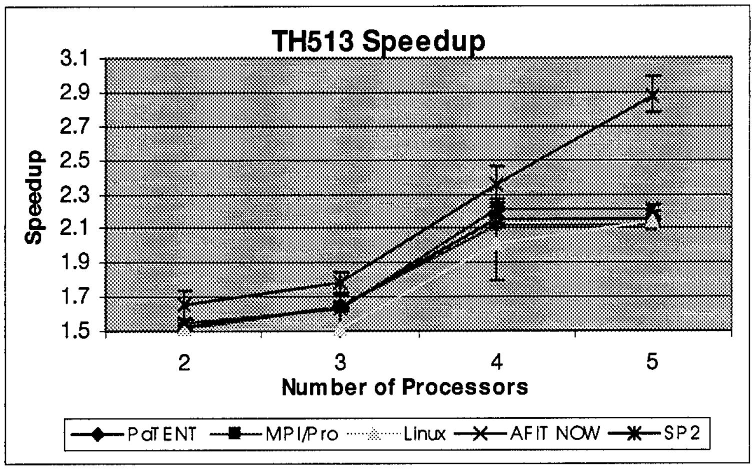


Figure 31: Effects of communication overhead on speedup

5.4.3 *Efficiency*

As was mentioned in Section 4.1.2, the *efficiency* of a parallel algorithm is a measure of the fraction of time for which a processor is busy doing useful work and is defined as the ratio of the speedup to the number of processors. Thus, as one would expect a small speedup indicates a low efficiency and a large speedup a high efficiency. With this in mind, the results of Figure 32 are no surprise.

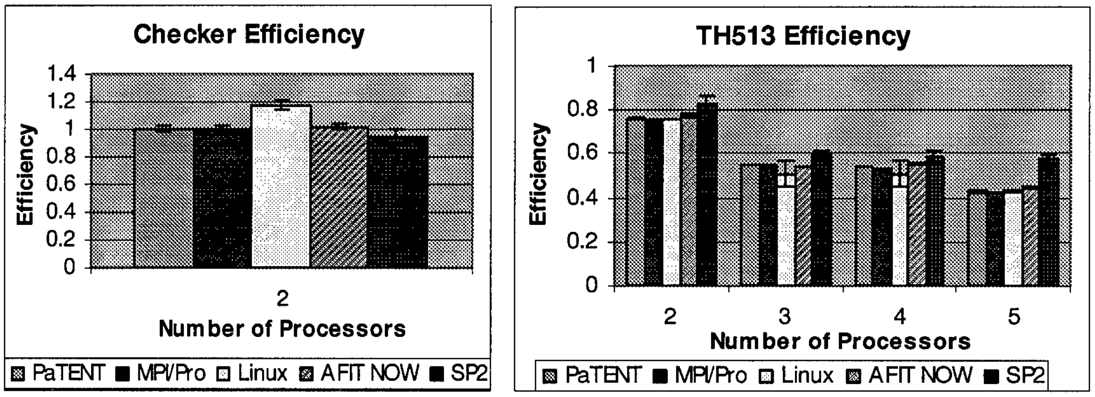


Figure 32: cGRaCCE Efficiency with Checker and TH513 data sets

In this figure, results of cGRaCCE with the checker data set, which exhibited a superlinear speedup for all platforms except the SP2, show an efficiency of one or greater. With the TH513 data set, the efficiency is 20% to 50% lower than with the checker data set, again showing the side effects of the cache advantage, load imbalance, and higher communication overhead discussed in the last section. Figure 33 displays the efficiency that is expected if the cache advantages of loop iterations greater than one are ignored. This gives a good picture of the scalability of the cGRaCCE algorithm, which as shown in the chart below, is capable of efficiencies, ranging from 77% to 97% even with the rather heavy load imbalance of the TH513 data set.

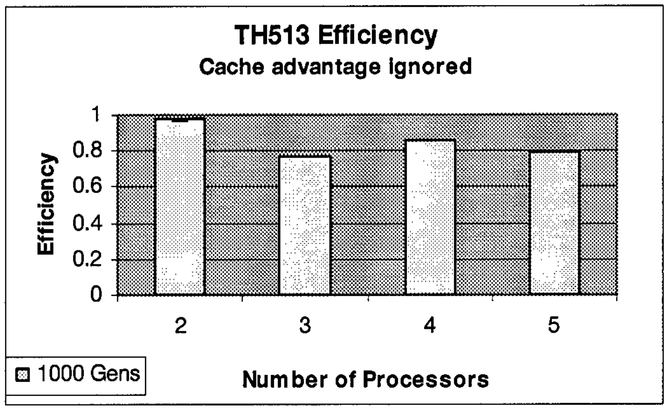


Figure 33: Predicted Efficiency with cache advantage ignored

5.4.4 Isoefficiency

In Section 4.1.3, we defined *isoefficiency* as "the rate at which the problem size must to increased to maintain a fixed efficiency." This value gives us a good indication of the scalability of an algorithm. Unfortunately, it is not always possible to determine the isoefficiency of an algorithm that is stochastic. That is, in a single trial, a particular problem size increase may maintain the same efficiency as the number of processors is increased by one. In a subsequent trial, the stochastic algorithm may converge to a solution faster even with the same problem size resulting in a higher efficiency.

Another problem that is more specific to the cGRaCCE algorithm is the definition of the *problem size*. In the experiments with cGRaCCE, we varied the problem size by using data sets with varying numbers of data points, boundary points, partitions, features, and classes. We also varied the problem size by increasing the minimum number of generations for the GA search from ten to one thousand. We could have also changed the problem size by varying the size of the GA population, the mutation and crossover probabilities, the cluster purity level, the maximum number of partitions evaluated per boundary point, and many other parameters. Therefore, defining the problem size for cGRaCCE is within itself quite complex.

Furthermore, to derive an effective isoefficiency function it is necessary to start with a valid complexity equation for the algorithm and as is explained in Section 5.4.1, this is very difficult, if not impossible, to derive for the cGRaCCE algorithm. Thus, with all of these factors in mind, a derivation of the isoefficiency function for cGRaCCE is not included.

5.4.5 Statistical Validation

As was described in Section 4.3, the ANOVA test was applied to each sample population with a 0.05 level of significance (95% C.I.). This test served two primary functions. First, it served to show that the data collected from the experiments was valid for comparison. Second, it was used as a "test of means" to determine if the difference between two sample populations was of statistical significance. Some examples of the *analysis of variance tables* produced by these tests are shown in Appendix C.

5.4.6 Conclusions - cGRaCCE Performance

In this section, we have looked in detail at the results produced by several trials of the cGRaCCE algorithm with different sets of test data. For one of these data sets (checker), we experienced superlinear speedup, which was determined to be caused by the decomposition of the data into smaller segments, which fit into the L1 cache of remote processors. For the remaining data sets, cGRaCCE exhibited low speedup and efficiency. A careful analysis of the results revealed the following contributing factors: 1) *unequal load balancing*, 2) *first-iteration caching*, and 3) *communication overhead.*

Because there is no interprocessor communication with cGRaCCE, other than the initial data distribution and the final collection of results, the effects of the communication overhead proved to be marginal for all but the smallest problem sizes. Also, the effect of load imbalances observed for most data sets, although significant was determined to only be a minor contributor to the low speedup and efficiency. The major factor of the three was the first-iteration caching. It was pointed out that ignoring initial loop iterations in parallel programs is a common practice for more accurately determining the scalability of an algorithm. Since the amount of time to load data and instructions in the cache during the first iteration of a loop is marginal with large data sets where the number of loops is much greater than those normally found in test data, this practice is completely valid. Therefore, by estimating the performance of cGRaCCE minus the effects of caching, we were able to realize efficiencies of greater than 75% for all tested data sets. Although these efficiencies fall short of the ideal, they show that even with unequal load balancing the cGRaCCE algorithm is capable of relatively good performance. The recommended next step in improving the cGRaCCE algorithm is to devise a dynamic load balancing scheme that more evenly distributes the work load among all of the processors without significantly increasing the communication overhead.

6 Conclusions/Recommendations

6.1 Review

Chapter 1 defines the objectives and goals of this research effort. Chapter 2 provides appropriate background on parallel computing and data mining, including some of the general principles and current research. In Chapter 3, the problem domain and algorithm domain is discussed. Chapter 4 presents the experiment design and general methodology for completing this research. Using the results of the experiments described in Chapter 4, a detailed data analysis is presented in Chapter 5. This Chapter culminates the thesis research with a summary of the conclusions presented in Chapter 5, a discussion of the contributions of this effort, and recommendations for future research.

6.2 Summary

The conclusions for each of the three main objectives of this research are presented in Chapter 5. Those conclusions are summarized as:

- *The efficiency/performance of Linux and NT PoPCs is very competitive to that demonstrated by NOWs and MPPs.*
- *The rapid technological advances in PC technology and lower commercial costs give PoPCs a clear price-performance advantage over NOWs and MPPs.*
- *Clusters of NT workstations are viable alternatives to Linux clusters for parallel and distributed computation.*
- *NT clusters can perform as well or better than Linux clusters for computationally intensive algorithms.*
- *Differences in communication overhead for Linux and NT clusters are small and for the most part insignificant.*

- *The MPI/Pro and PaTENT MPI communication libraries for Windows NT demonstrate no significant difference in performance in trials with up to six PCs. These tools are still relatively new and some bugs are still being worked out.*
- *The parallel C++ version of the GRaCCE algorithm, known as cGRaCCE, achieves a significant efficiency/performance advantage over the original MatLab code. This advantage, although not precisely measured, is at least an order of magnitude based on results from runs with the same data sets using both algorithms.*
- *The static load balancing scheme used by cGRaCCE results in a significant load imbalance for the majority of tested data sets.*
- *Ignoring the effects of data and instructions caching, the cGRaCCE algorithm is capable of relatively high efficiencies (77%-97%) even with a significant load imbalance.*
- *The communication overhead of the cGRaCCE algorithm is relatively trivial, averaging less than 2.3% of total execution time for all trials with a moderate workload of 1000 generations .*

6.3 Contributions

This research has made several contributions to the field of Computer Science/Computer Engineering, particularly in the areas of parallel computing and data mining. Some of the specific contributions are outlined below:

- *Provides one of the first detailed comparisons of the performance of NT versus Linux clusters.* By showing that NT clusters are viable alternatives to "free" UNIX-type clusters for parallel and distributed processing, a whole new door of opportunity is opening up to both academia and the commercial world where Windows NT has gained a strong foothold. This is especially significant to the Air Force in light of recent policies that define NT as the O/S of choice for all Air Force LAN servers and desktops.

- *Demonstrated that PoPCs are viable alternatives to Networks of Workstations and Massively Parallel Processors.* This effort builds on the current PoPC research, such as NASA's Beowulf project, with the added twist of evaluating NT, as well as Linux clusters. Price-performance comparisons show that relatively high performance parallel computing can be attained at commodity prices, allowing even organizations with comparably small IT budgets to take advantage of the benefits of distributed processing.

- *Showed that MPI implementations for NT, although still somewhat immature, are capable of competitive performance when compared with similar MPI implementations for UNIX.* This research also showed that the two primary MPI tools (MPI/Pro and PaTENT MPI) for NT are relatively equal in performance, allowing consumers some choice of which tool to use.

- *Provided the sponsor with parallel C++ code implementing the major portion of the original GRaCCE algorithm. Although this algorithm as demonstrated is not the most efficient version possible, it is at least an order of magnitude faster that the original MatLab code.* This algorithm provides the data mining community with an alternative to the traditional decision tree algorithms, which significantly reduces the complexity of the rule set for about the same accuracy and performance.

6.4 Proposals for Future Research

This research provided useful insight into the parallel tools for NT clusters and potential of the GRaCCE algorithm on a parallel system. Nevertheless, there are several potential areas left unexplored which could contribute to this research effort. The following areas are recommended for future research:

- *Explore the use of an efficient dynamic load balancing algorithm for the parallel GRaCCE algorithm.* The schemes outlined in [El-Rew94] are probable candidates if the algorithm is rewritten to run the boundary point searches in parallel rather than only the class searches.

- *Optimize the memory handling functions in the current parallel GRaCCE code to reduce communication time and memory usage.* The converted C++ code for GRaCCE uses static memory allocation for the majority of the data structures used in this algorithm. Although this design decision proved sufficient for the test data used for this research, it limits the scalability of the algorithm. This constraint could be at least partially removed by using dynamic memory allocation for all data structures and freeing any memory when the data in it are no longer needed.

- *Investigate possible optimizations to reduce the overhead imposed by the Windows NT operating system.* Although the comparison of NT and Linux clusters showed no significant disadvantage caused by NT's much larger O/S code, the possibility of gaining performance by reducing overhead still exists. Possible optimizations include: 1) running parallel programs without the graphical user interface (Explorer) on NT, 2) optimizing the TCP/IP stack or installing one that is more efficient, and 3) shutting down or removing any non-critical services from NT. Optimizations dealing with real-time NT performance are discussed in [Timmer97].

- *Investigate possible optimizations for the Linux environment.* Two such optimizations are discussed in Chapter 5. First, the TCP bug needs to be corrected by either implementing the recommended fix [NIST] and recompiling the kernel or by upgrading the kernel to a version that is unaffected by this bug. Second, a C++ compiler that is optimized for the Pentium II environment, such as PGCC, needs to be loaded on the Linux system and used for all future C++ code. A third possible optimization would be to investigate the use of "leaner" faster messaging layers in lieu of TCP/IP. One such messaging layer that is currently available for Linux is the Illinois Fast Messages (FM). [Pakin95]

- *Investigate the use of threads for increased parallel performance with both NT and Linux.* Since both of these operating systems support the use of threads, it may be possible to increase performance by distributing the workload assigned to a processor among multiple threads. This would be used in conjunction with message passing.

Appendix A: General Overview of Parallel Processing

A.1 Introduction

Whether it be transportation or communication, mankind has always bemoaned his "need for speed" in all that he has attempted to do. This is no different in the world of computing. Even as processors become faster and faster, the gap between the available and "needed" computing power is stretched by mankind's attempt to tackle increasingly complex problems. Parallel processing has made progress in narrowing this gap by utilizing the power of multiple processors to solve a single problem in parallel.

One can find many illustrations of parallel processing in the real world. For instance, race car drivers in an attempt to minimize the time required for pit stops, utilize the services of a crew of personnel to simultaneously perform all necessary tasks such as changing tires, refueling, cleaning windshields, etc. If these tasks were performed serially by a single crewmember, there would be no need for the driver to re-enter the race, as he would be much too far behind to successfully compete. Likewise, in the world of computing, a particular problem often consists of a collection of independent tasks, which can be completed concurrently. This inherent "parallelism", as it is called, provides the driving force behind parallel computing.

As was briefly mentioned in Chapter 1, two advances that have made parallel processing possible have been the development of faster, less expensive interconnection networks for connecting processors and memory and the development of standardized tools for programming parallel applications. These advancements are discussed in general in the sections that follow. However, it is necessary to first provide a brief overview of the general principles of parallel processing, beginning with the classification method for parallel computers devised by Flynn. [Flynn66]

A.1.1 Flynn's Taxonomy

Traditional serial computers are based on the Von Neumann model. This model consists of a central processing unit (CPU) and memory, which takes a single sequence of instructions and operates on a single sequence of data. In 1966, Michael Flynn introduced a classification of parallel computers based on their control mechanism and memory configuration. [Flynn66] Under this classification, the Von Neumann model is referred to as a single instruction stream, single data stream (SISD) computer. A computer in which a single control unit issues instructions to each processing element is referred to as a single instruction stream, multiple data stream (SIMD) computer. Multiple instruction stream, multiple data stream (MIMD) computers, such as the Intel Paragon XP/S, allow each processor to execute a different program independent of the other processors. Flynn's model also includes multiple instruction stream, single data stream (MISD) computers although in reality there are no commercial examples of these computers. MIMD computers are the most popular for modern MPPs and provide the greatest flexibility. An illustration of a typical SIMD and MIMD architecture is shown in Figure 34. [Kumar94] The two primary communication architectures for MIMD computers are discussed in the next section.

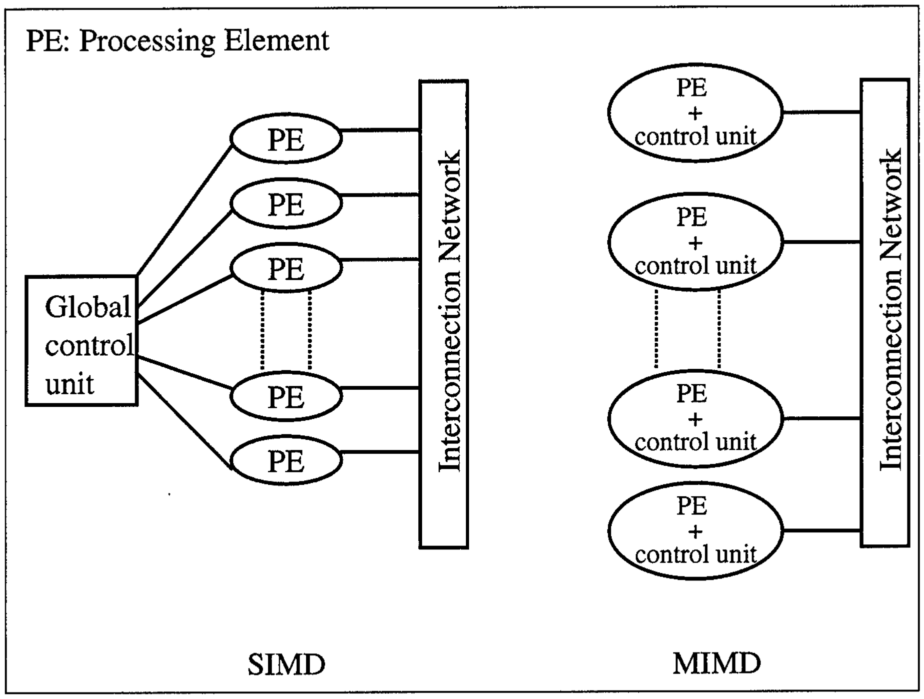


Figure 34: Layout of a typical SIMD and MIMD architecture

A.1.2 Shared Memory vs. Message-Passing Architectures

In order to run a parallel application, there must be some way of communicating between processors. MIMD computer architectures are divided into two main groups based on their method of interprocessor communication - *message-passing* and *shared-memory*. Both of these architectures are illustrated in Figure 35. [Kumar94] [Ragsda91] In a message-passing architecture, each processor has its own memory and communicates via an interconnection network (ICN). The processors can only communicate with other processors by passing messages. Shared-memory computers, as the name implies, provide a common memory for all processors. These processors communicate by changing data values in this address space. This research is limited to message-passing MIMD systems, which include PoPCs, NOWs, and most modern MPPs.

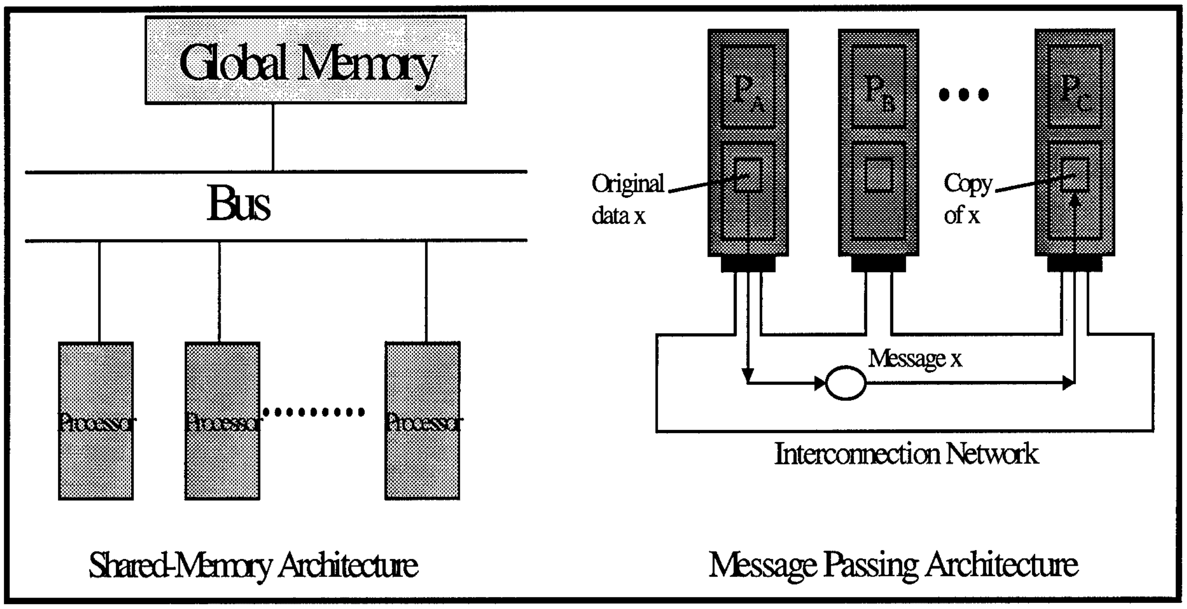


Figure 35: Two Primary Architectures for MIMD Computers

A.1.3 Interconnection Networks

A variety of interconnection networks may be used to connect processors and memory banks in shared-memory and message-passing computers. These networks can be classified into two major categories: *static* and *dynamic*. In dynamic networks, the path between processors and memory banks is dynamically determined by the use of switches and communication links. This type of network is commonly used in shared-memory computers. Static networks, on the other hand, are primarily used in message-passing architectures. These networks employ direct connections between processors.

A.1.3.1Dynamic Interconnection Networks

In order to reduce the number of switches required to connect processors to global memory, the global memory is divided into memory banks. In dynamic interconnection networks, these memory banks are connected to the processors in one of three primary configurations: crossbar switching networks, bus-based networks, and multistage interconnection networks. An example of each of these is illustrated in Figure 36. A crossbar switch is simply a grid of switching elements, which connects *p* processors to *b* memory banks. This switch requires *p* x *b* switching elements. Since it is unfeasible to have fewer memory banks than processors, the complexity of this system increases as the number of processors increase as $\Omega(p^2)$. Hence, crossbar switches are unscalable in terms of cost. Examples of crossbar switched networks include the Cray Y-MP and Fujitsu VPP 500. [Kumar94]

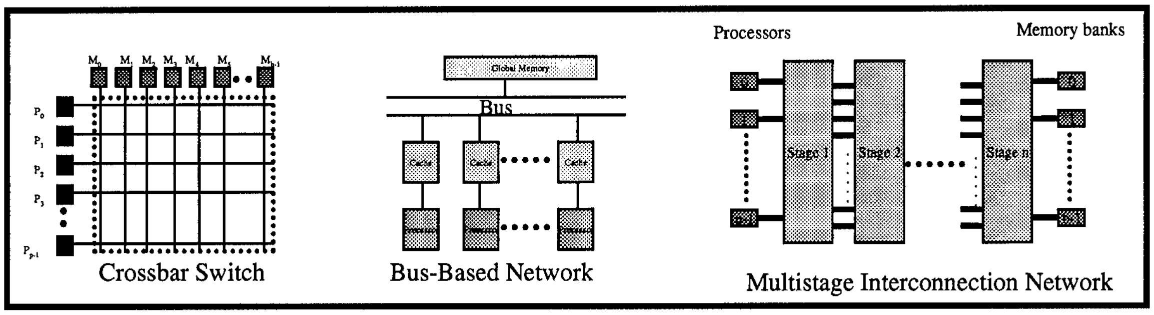


Figure 36: Dynamic Interconnection Networks

As with crossbar switches, bus-based networks are simple to construct. Processors and global memory are connected by means of a common data bus. Data requests and fetches are accomplished over the same bus. Since buses can carry only a limited amount of data, the processors may have to wait for memory accesses. This can lead to bottlenecks as the number of processors is increased. This problem can be partially alleviated by providing each processor with its own local cache memory, thus taking advantage of locality of reference. However, this technique may lead to *cache coherency* problems. That is, an outdated value may be read from the global memory by one processor before it can be updated from the local cache of a second processor. Even with local cache memories, the bus bottleneck can become a problem as the number of processors is increased beyond a certain level. Thus, it is uncommon to see these systems with more than 64 processors. [Tanenb95]

Multistage interconnection networks provide a middle ground between the high cost of crossbar switches and low performance of bus-based networks. These networks consist of multiple stages of interconnection patterns between processors and memory banks. At each stage, p inputs are connected to p outputs. The basic switching elements allow pass-through and crossover connections as a means of providing paths between all processors and memory banks. Using these stages, the required number of switches is significantly less than that needed for a crossbar switch, thus reducing costs. However, unlike the crossbar switch, multistage networks are blocking networks. That is, access to a specific memory bank by one processor may disallow access to another memory bank by another processor. An example of one commonly used multistage network, shown in Figure 37, is the Omega network, which is used in the IBM SP2. [Kumar94]

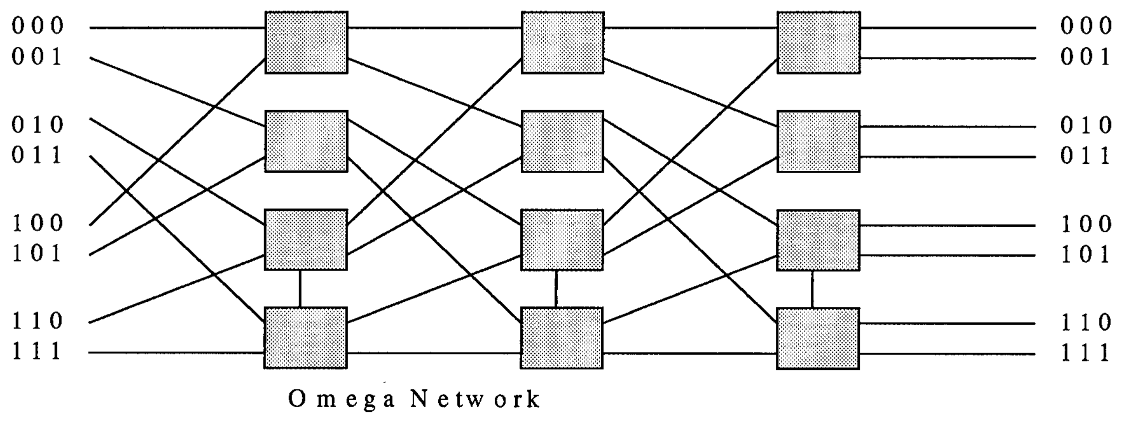


Figure 37: Omega Interconnection Network (used in IBM SP2)

A.1.3.2Static Interconnection Networks

As was mentioned earlier, static networks are commonly used to connect message-passing computers. One of the faster, as far as communication speeds, is the *completely-connected* network. As the name implies, each processor is directly connected to every other processor in the network. This is the fastest static network because each message has to traverse only one communication link between any two processors. However, it is also the most expensive in terms of communication links. This network is the static equivalent of the dynamic crossbar switching network; however, unlike the crossbar, the completely-connected network supports concurrent multi-channel communication from a single processor. The completely-connected network, as well as, the star, linear array, and ring networks, is illustrated in Figure 38. [Kumar94]

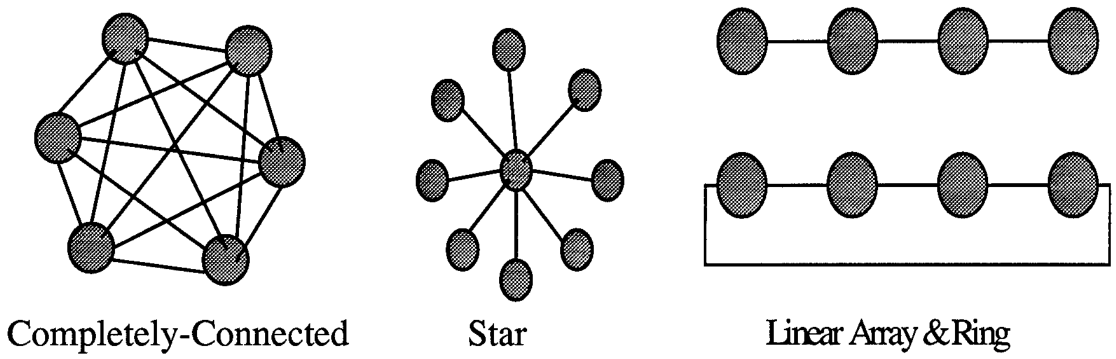


Figure 38: Static ICNs – Completely-connected, star, linear array, & ring

In a *star-connected* network, communication between any two processors must be routed through a central processor. This processor can become a bottleneck as communication increases. The *linear array* is the simplest static network. All processors are connected to two other processors, except at the ends. When a wraparound is added to the end processors, the linear array becomes a *ring*. Both the linear array and ring networks are special cases of a *tree* network. In a tree network, there is only one path between any pair of processors. A tree network may be static or dynamic. A static tree has processors at every node, whereas, a dynamic tree has processors only at the leaf nodes and switches at all intermediate nodes. Because all communication between processors must travel up the tree, communication bottlenecks can occur at the higher levels. This can be partially alleviated by increasing the number of communication links at higher levels. This type of tree is known as a *fat tree*. The tree and 2D Mesh networks are illustrated in Figure 39. [Kumar94]

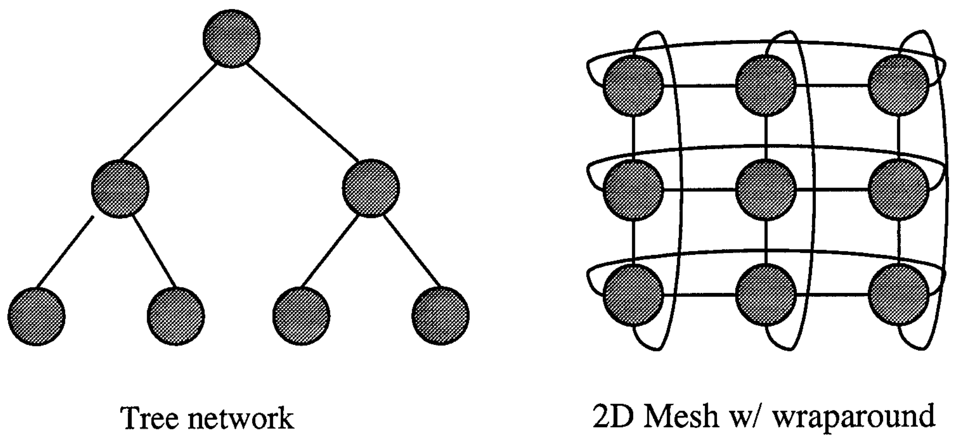


Figure 39: Static Interconnection Networks - Tree & 2D Mesh

A linear array, which has been extended into two dimensions, is known as a *two-dimensional mesh*. With the exception of the boundary processors, each processor in a mesh has a direct communication link with four other processors. If the boundary processors have wraparounds, then this is true of all processors in the mesh.

Communication in a mesh is accomplished by first sending a message along one dimension and then another dimension until it reaches the desired destination. Examples of mesh-based computers include the Paragon XP/S and the Cray T3D.

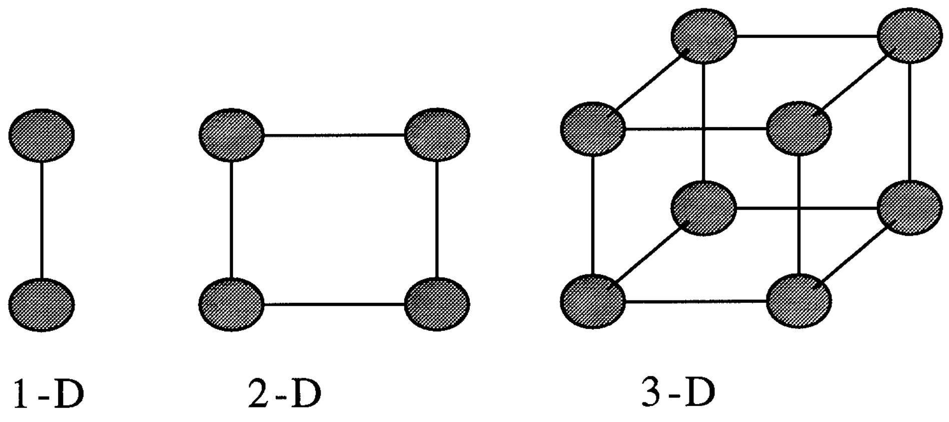


Figure 40: Static Interconnection Networks - 1-, 2-, & 3-dimensionsal hypercubes

One of the most versatile static networks is the *hypercube*, shown in Figure 40. It is a "multidimensional mesh of processors with exactly two processors in each dimension."[Kumar94] The number of processors in a hypercube is equal to 2^d, where d is the number of dimensions. A d+1-dimensional hypercube is constructed by linking the processors of two d-dimensional hypercubes. The binary representation of the labels of each pair of directly-connected processors differ by at most one bit position. This is important when distributing data to processors for parallel computation, as it can be used to reduce communication overhead. Each processor in a hypercube is connected to d other processors and the shortest path between any two processors cannot have more than d links. Traditionally a very popular choice for MPPs, hypercubes networks containing up to 16,384 CPUs are commercially available, but their popularity has waned in recent years. [Tanenb95] The nCUBE 2 and Cosmic Cube are examples of some hypercube network computers.

A.1.3.3Cost/Performance Metrics

It is important to evaluate the tradeoffs of each of the different static network types to determine the "best" choice for a particular application. Some of the criteria, which can be used, includes *network diameter*, *arc connectivity*, *bisection width*, and *cost*. [Kumar94] The network diameter is defined as the maximum distance between any two processors in a network. Shorter diameters are better, as this reduces communication times. As was mentioned earlier, the completely-connected network is the fastest in terms of communication speed, with a diameter of one. The linear array is the slowest with a diameter of p-1. Arc connectivity is the minimum number of arcs that must be removed to break a network into two disconnected networks. Higher connectivity is desirable, as it reduces the possibility of resource contention or downtime. The 2-D wraparound mesh and hypercube both have good connectivity. A measure of the minimum number of links that have to be removed to partition the network into equal halves defines the bisection width. Since a hypercube is constructed by connecting two sets of p/2 processors, its bisection width is p/2. Lastly, the cost which may be defined as the number of communication links required by a network is highest for the completely-connected network, p*(p-1)/2, and lowest for the ring, p. A summary of the metrics for each of the static networks discussed is presented in Table 18. [Kumar94]

Network	*Diameter*	*Bisection Width*	*Arc Connectivity*	*Cost (No. of links)*
Completely-connected	1	$p^2/4$	p-1	p(p-1)/2
Star	2	1	1	p-1
Linear array	p-1	1	1	p-1
Ring	$\lfloor p/2 \rfloor$	2	2	p
2-D mesh w/o wraparound	$2(p^{1/2}-1)$	$p^{1/2}$	2	$2(p - p^{1/2})$
2-D wraparound mesh	$\lfloor p^{1/2}/2 \rfloor$	$2p^{1/2}$	4	2p
Hypercube	log p	p/2	log p	(p log p)/2

Table 18 : Metrics for Static Interconnection Networks

Appendix B: AFIT Bimodal Cluster

The figure below is a diagram of the AFIT Bimodal Cluster (ABC) of NT/Linux PCs. The cluster as shown in this diagram has been in existence since Jan 5, 1999. The inaugural run of the original four-node cluster was on May 19, 1998.

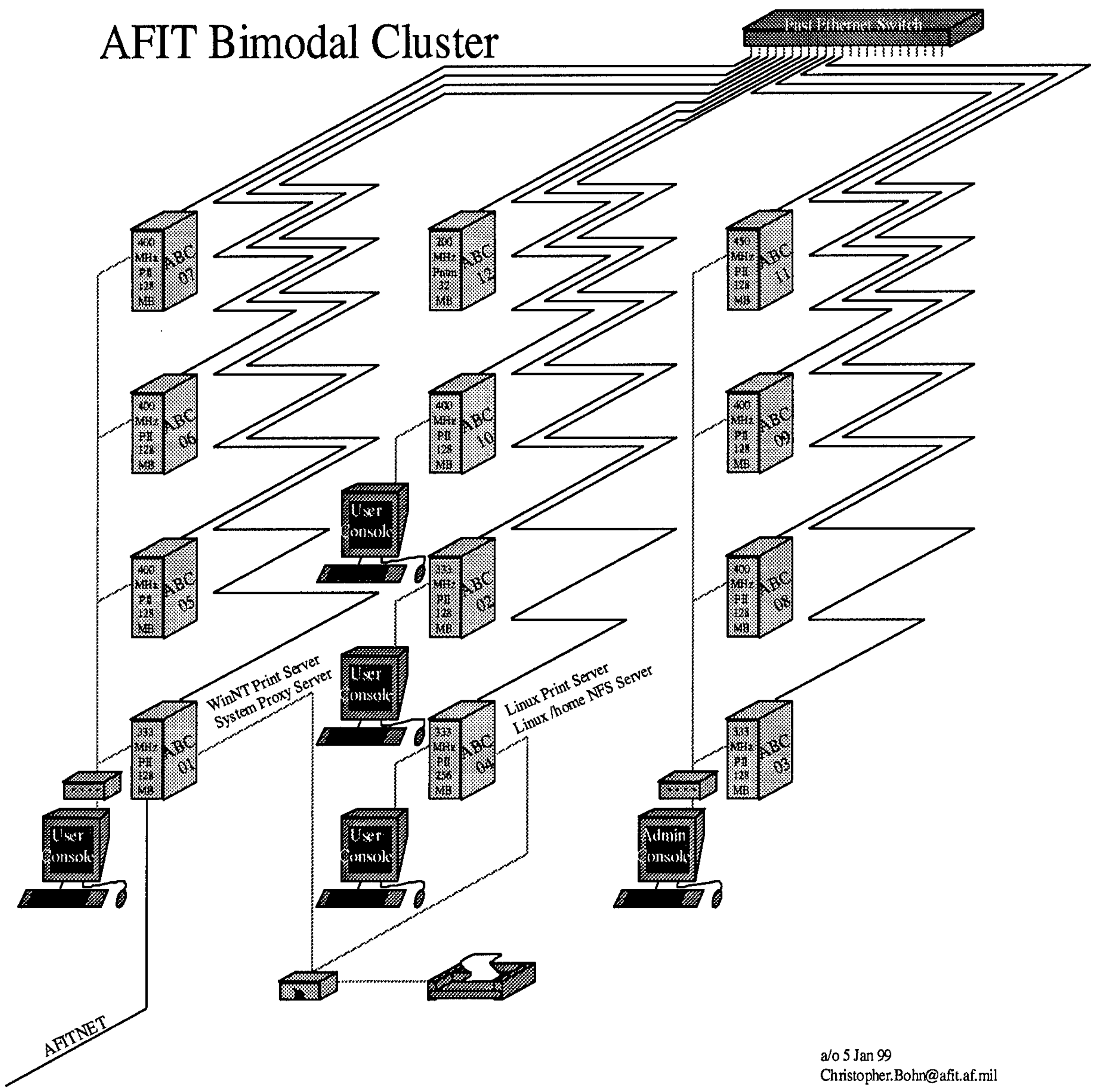

AFIT Bimodal Cluster
Fast Ethernet Switch
400 MHz PII 128 MB ABC 07
300 MHz Pntm 32 MB ABC 12
450 MHz PII 128 MB ABC 11
400 MHz PII 128 MB ABC 06
400 MHz PII 128 MB ABC 10
400 MHz PII 128 MB ABC 09
400 MHz PII 128 MB ABC 05
333 MHz PII 128 MB ABC 02
400 MHz PII 128 MB ABC 08
333 MHz PII 128 MB ABC 01
333 MHz PII 256 MB ABC 04
333 MHz PII 128 MB ABC 03
WinNT Print Server
System Proxy Server
Linux Print Server
Linux /home NFS Server
User Console
User Console
User Console
User Console
Admin Console
AFITNET
a/o 5 Jan 99
Christopher.Bohn@afit.af.mil

Figure 41: Diagram of AFIT Bimodal Cluster (ABC)

Appendix C: Analysis of Variance Tables

Since the ANOVA test requires that the samples tested be normally distributed, a "large" number of trials (i.e. 30) were run to ensure a normal distribution. The distribution was also checked using the SAS/JMP tool and Microsoft Excel. Results from two of these tests are shown in Figure 42.

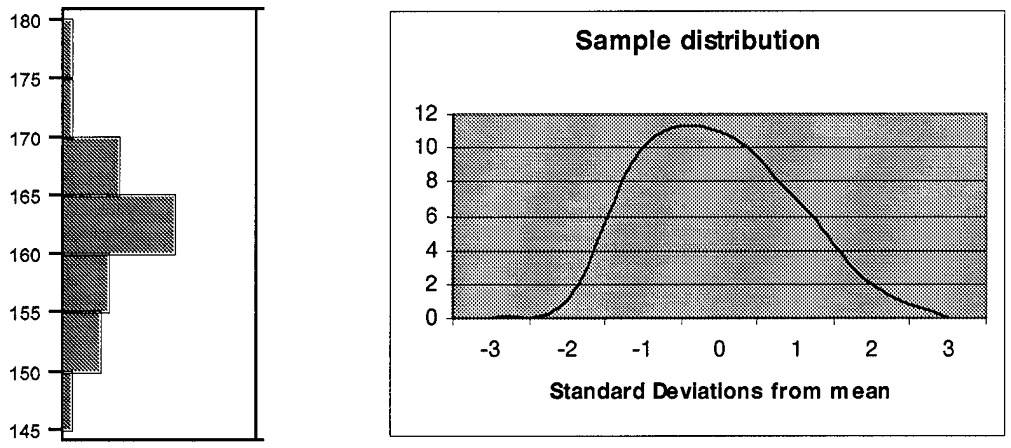


Figure 42: Plot of Sample distribution using SAS/JMP and Excel

The tables that follow were produced using the ANOVA tool in Microsoft Excel 97. A summary of the analysis is presented at the bottom of each table. The values shown in this summary are as follows:

- **Sample** – This is the amount of variation between each sample. In these tables, the samples are the different platforms. This was the largest source of variation for both tables. This tells us that the platform used had the greatest effect on the performance of the algorithm.

- **Columns** – This is the amount of variation between each column. In this case the column represents the number of processors and was the second largest source of variation as expected.
- **Interaction** – This is the amount of variance between a cross section of the rows and columns. In other words, this represents all of the other factors that cause the performance to vary nonlinearly on each system. These include such things as cache memory sizes, network bandwidth, and network latency. This value is lower than the "sample" and "columns" value and indicates that we can compare these two factors, but need to look at other factors also.
- **Within** – This represents the amount of variation within each sample and is the lowest source of variation. A high number here represents errors in the sample. These could be caused by such things as heavy CPU utilization by processes external to the executing program, network glitches, or failed nodes.

Table 19 presents the results of applying the ANOVA test to data collected from all trials of cGRaCCE with the TH513 data set for a problem size of 1000 generations. Several observations can be made from the data that validate conclusions made in Chapter 5. These observations are:

1) *The test statistic (F) falls in the critical region (i.e. > F crit) for all sources of variation indicating that:*
   a) *The mean is different for each platform (i.e. the performance varied from one platform to another).*

b) *The mean changes as the number of processors is increased (i.e. the performance varies with the number of processors).*

c) *The interaction means are not equal. This indicates that there are factors (e.g. load balance) other than platform and number of processors affecting the performance of cGRaCCE.*

2) *The largest variance/average ratio is produced by trials with Linux on three and four processors. This indicates a problem, which was determined to be caused by a TCP bug in the Linux kernel.*

Anova: Two-Factor With Replication

	Number of Processors					
SUMMARY	1	2	3	4	5	Total
ABC-NT						
Count	30	30	30	30	30	150
Sum	553.27	365.05	338.04	261.583	261.638	1779.581
Average	18.44233	12.16833	11.268	8.719433	8.721267	11.86387
Variance	0.001225	0.007263	0.001113	0.003776	0.001076	12.78551
ABC-Linux						
Count	30	30	30	30	30	150
Sum	751.89	496.35	496.45	375.11	351.05	2470.85
Average	25.063	16.545	16.54833	12.50367	11.70167	16.47233
Variance	0.006091	0.003233	15.2312	9.125045	0.001987	27.35748
AFIT NOW						
Count	30	30	30	30	30	150
Sum	1281.71	827.28	787.25	578.97	578.27	4053.48
Average	42.72367	27.576	26.24167	19.299	19.27567	27.0232
Variance	0.134038	0.222563	0.011373	0.07734	0.057963	74.01243
IBM SP2						
Count	30	30	30	30	30	150
Sum	2575.78	1561.15	1444.81	1096.72	894.8	7573.26
Average	85.85933	52.03833	48.16033	36.55733	29.82667	50.4884
Variance	14.51414	10.54788	3.479638	4.677124	2.112547	385.3813
Total						
Count	120	120	120	120	120	
Sum	5162.65	3249.83	3066.55	2312.383	2085.758	
Average	43.02208	27.08192	25.55458	19.26986	17.38132	
Variance	699.8266	243.7639	205.4143	118.3288	67.5205	

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Sample	133572.2	3	44524.08	14787.97	0	2.620268
Columns	49155.57	4	12288.89	4081.563	0	2.387296
Interaction	23529.12	12	1960.76	651.2356	0	1.768871
Within	1746.282	580	3.01083			
Total	208003.2	599				

Table 19: ANOVA values for TH513 data set on all platforms

Conclusions 1a, 1b, and 1c from the previous discussion of the ANOVA results in Table 19, also apply to the ANOVA results for cGRaCCE with the checker data set in Table 20. Additionally, the interaction value is much lower in proportion to the other sources of variance for this data set, indicating that the platform and number of processors has a greater effect on performance than other factors such as load balance and caching.

Anova: Two-Factor With Replication

	Number of Processors		
SUMMARY	1	2	Total
ABC-NT			
Count	30	30	60
Sum	9664.8	4800.7	14465.5
Average	322.16	160.0233	241.0917
Variance	139.1308	31.14392	6767.16
ABC-Linux			
Count	30	30	60
Sum	15334.1	6519.6	21853.7
Average	511.1367	217.32	364.2283
Variance	27482.21	48.06717	35479.69
AFIT NOW			
Count	30	30	60
Sum	21782.6	10663.6	32446.2
Average	726.0867	355.4533	540.77
Variance	1449.481	67.85568	35670.15
IBM SP2			
Count	30	30	60
Sum	21017.1	11093.8	32110.9
Average	700.57	369.7933	535.1817
Variance	1431.092	589.6331	28810.15
Total			
Count	120	120	
Sum	67798.6	33077.7	
Average	564.9883	275.6475	
Variance	34206.52	8247.63	

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Sample	3778229	3	1259410	322.5264	1.86E-82	2.64351
Columns	5023087	1	5023087	1286.379	1.29E-96	3.881851
Interaction	367895	3	122631.7	31.40515	4.48E-17	2.64351
Within	905919.8	232	3904.827			
Total	10075130	239				

Table 20: ANOVA values for Checker data set on all platforms

Bibliography

[Allen90] Allen A., *Probability, Statistics, and Queuing Theory with Computer Science Applications.* Academic Press, Inc., 1990.

[Anders95] Anderson T., Culler D., Patterson D., and the NOW Team. "A Case for NOW (Networks of Workstations)," *IEEE Micro* 15:1 (February 1995), pp54-64.

[Baker98] Baker M., "MPI on NT: The Current Status and Performance of the Available Environments", *EuroPVM/MPI98*, Liverpool, UK.

[Bakerm98] Baker M., Carpenter B., Fox G., Ko S., and Li X., "mpiJava: A Java MPI Interface," To be published in *Scientific Publishing*, 1999.

[Barr95] Barr S., Golden B., Kelly J., Resende M. and Stewart W., "Designing and Reporting on Computational Experiments with Heuristic Methods," *Journal of Heuristics*, 1: 9-32, 1995.

[Bohn98] Bohn C., "Asymmetrical Load Balancing on a Nonuniform Cluster of PCs," MSCE thesis, AFIT/GE/ENG/99M-02, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, March 1999.

[Briema84] Breiman L., Friedman J., Olsen R., and Stone C., *Classification and Regression Trees.* Wadsworth International Group, 1984.

[Chien97] Chien A., et al., "High Performance Virtual Machines (HPVM): Clusters with Supercomputing APIs and Performance," *Eighth SIAM Conference on Parallel Processing for Scientific Computing (PP97),* 1997.

[CH_P4] P4 communication library developed by Argonne National Labs and available at: ftp://info.mcs.anl.gov/pub/p4/

[El-Rew94] El-Rewini H., Lewis T., and Ali H., *Task Scheduling in Parallel and Distributed Systems.* Englewood Cliffs, New Jersey: PTR Prentice Hall, 1994.

[ERC] MSU/ERC, "MPI on Windows NT," http://www.erc.msstate.edu/mpi/mpiNT-download.html

[Festa98] Festa P., "Windows NT Server Market Grows," http://www.news.com/News/Item/0,4,18542,00.html

[Flynn66] Flynn, M., "Very High-Speed Computing Systems," *Proceedings of the IEEE* 54:12 (December 1966), pp1901-1909.

[GEA] Gigabit Ethernet Alliance, "Gigabit Ethernet Standard Formally Ratified; Gigabit Ethernet Poised for Widespread Deployment," http://www.gigabit-ethernet.org/news/releases/062998.html

[Gindha97] Gindhart D., "A Comparative Analysis of Networks of Workstations and Massively Parallel Processors for Signal Processing," MSCE thesis, AFIT/GCE/ENG/97D-01, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1997.

[GNUGCC] GNU GCC Home Page: http://www.gnu.org/software/gcc/gcc.html

[Goldbe89] Goldberg D., Genetic *Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley Publishing Company, Inc., 1989.

[Gropp96] Gropp W., Lusk E., Doss N. and Skjellum A., "A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard," *Parallel Computing,* Vol. 22, pp. 789-828, 1996.

[Hebert98] Hebert L., Seefeld W., Skjellum A., Taylor C., Dimitrov R., "MPI for Windows NT: Implementations and Experience with the Message Passing Interface for Clusters and SMP Environments," *Proceedings of the PDPTA '98 International Conference*, 1998.

[Henley97] Henley G., et al., "BDM: A multiprotocol Myrinet control program and host application programmer interface," Mississippi State University, Technical Report #MSSU-EIRS-ERC-97-3, May 1997.

[HPVM] University of Illinois' High Performance Virtual Machine (HPVM) Home Page: http://www-csag.cs.uiuc.edu/projects/hpvm.html

[IBMSP2] IBM SP System Home Page: http://www.rs6000.ibm.com/sp.html

[Joshi97] Joshi K., "Analysis of Data Mining Algorithms," http://www.gl.umbc.edu /~kjoshi1/data-mine/proj_rpt.htm, 1997.

[Kumar94] Kumar V. et. al, *Introduction to Parallel Computing*. Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc, 1994.

[Lauria97] Lauria M. and Chien A., "MPI-FM: High Performance MPI on Workstation Clusters," *Journal of Parallel and Distributed Computing*, January 1997.

[Linux] The Linux Documentation Project: http://sunsite.unc.edu/mdw/linux.html

[LUGR] Linux User Group Registry: http://www.linux.org/users/index. html

[Marmel98] Marmelstein R. and Lamont G., "GRaCCE: A Genetic Environment for Data Mining", *Intelligent Engineering Systems Through Artificial Neural Networks*, vol. 8, pp. 405-412, ASME Press, 1998.

[Marmel99] Marmelstein, R., "Evolving Compact Decision Rule Sets," Ph.D. dissertation, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, June 1999 (projected).

[Marml99] Marmelstein R., Hammack L., and Lamont G., "A Concurrent Approach for Evolving Compact Decision Rule Sets," *Data Mining and Knowledge Discovery: Theory, Tools, and Technology, part of SPIE's Aerosense '99,* Orlando, FL, April 5-9, 1999.

[MatLab] MatLab 5.x numerical computation, graphics, visualization, and programming language software by The MathWorks, Inc., Home Page: http://www.mathworks.com/products/matlab/

[Mobash96] Mobasher B., Jain N., Han E., and Srivastaba J, "Web Mining: Pattern Discovery from World Wide Web Transactions (1996)," available at http://www-users.cs.umn.edu/~mobasher/webminer.html.

[MPI] The Message Passing Interface (MPI) Standard Home Page: http://www-c.mcs.anl.gov/Projects/mpi/

[MSRC] ASC MSRC Home Page: http://www.asc.hpc.mil/

[MST] MPI Software Technology Inc., MPI/PRO^{TM}: http://www.mpich.com/products/mpi/mpipro/PDS-MPIProNT-Feb1998-1.html

[NIST] NIST Website – Code fix for TCP performance drop in Linux: http://www.multikron.nist.gov/scalable/misc_info/Linux_TCP.html

[Nupair94] Nupairoj N. and Ni L., "Performance Evaluation of Some MPI Implementations on Workstation Clusters," *Proceedings of the 1994 Scalable Parallel Libraries Conference (SPLC94)*, October 1994.

[Pachec97] Pacheco, P.: *Parallel Programming With MPI.* San Francisco, CA: Morgan Kaufmann, 1997.

[Pakin95] Pakin S., Lauria M., and Chien A., "High Performance messaging on workstations: Illinois Fast Messages (FM) for Myrinet," *Proceedings of Supercomputing '95*, San Diego, CA, December 1995.

[Pallas]	Pallas MPI Benchmarks: http://www.pallas.de/pages/pmbd.htm
[PaTENT]	Genias Software, PaTENT MPI 4.0 Home Page: http://gimli.genias.de/products/patent/index.html
[PC4.5]	Parallel C4.5 Home Page: http://merv.cs.nyu.edu:8001/~binli/ pc4.5/
[PGCC]	Pentium Compiler Group FAQ: http://goof.com/pcg/pgcc-faq.html
[Platform]	Platform Computing's commercial Load Sharing Facility Software for HPVM front-end available at http://www.platform.com/
[PMSI]	PMSI paper on "Data Mining," http://www.geocites.com/CapeCanaveral/Launchpad/7651/dminita.htm
[Provant]	Provantage Computer Products Home Page: http://www.provantage.com/
[Quinn94]	Quinn, M., *Parallel Computing: Theory and Practice.* San Francisco, CA: McGraw-Hill, Inc., 1994.
[Quinla93]	Quinlan J.: *C4.5 – Programs for Machine Learning.* San Francisco, CA: Morgan Kaufmann, 1993.
[Ragsda91]	Ragsdale S., *Parallel Programming.* New York, NY: McGraw-Hill, Inc., 1991.
[RedHat]	Red Hat Software, Inc. Home Page: http://www.redhat.com/
[Ridge97]	Ridge D., Becker D., Merkey P. and Sterling T., "Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs," *Proceedings, IEEE Aerospace*, 1997.
[Sharaz95]	Shirazi B. and Hurson A., *Parallelism Management Scheduling and Load Balancing.* San Francisco, CA: PTR Prentice Hall, 1995.
[SP2MPI]	Parallel Environment v2.3 (IBM SP2's proprietary MPI implementation) Home Page: http://www.austin.ibm.com/software/sp_products/pe.html
[SPEC]	The Standard Performance Evaluation Corporation (SPEC) Home Page: http://www.spec.org/
[Sterli98]	Sterling T., "Beowulf-class Clustered Computing: Harnessing the Power of Parallelism in a Pile of PCs," *Proceedings of the Genetic Programming 98 Conference*, April 1998.
[Tanenb95]	Tanenbaum A., *Distributed Operating Systems.* Upper Saddle River, NJ: Prentice Hall, Inc., 1995.

[Timmer97] Timmerman M., "Windows NT Real-Time Extensions: An Overview," *Real-Time Magazine,* Quarter 2, 1997. Available at http://www.realtime-info.be/encyc/magazine/97q2/winntext.htm

[UCLA] UCLA: "Classification and Regression Trees (CART)," http://neurosun.medsch.ucla.edu/BMML/Stitt/new.htdocs/cart.html

[USAFFS95] USAF Fact Sheet 95-20, "Information Warfare and its Importance," http://www.af.mil/news/factsheets/Information_Warfare.html

[VAST/f90] VAST/f90 – a high performance Fortran compiler from Pacific-Sierra Research available at: http://www.psrv.com/lnxf90.html

[vonEic92] von Eicken T., et al., "Active Messages: A mechanism for integrated communication and computation," *Proceedings of the 19th ISCA,* May 1992, pp. 256-266.

[Weiss91] Weiss S. and Kulikowxki C., *Computer Systems That Learn.* San Francisco, CA: Morgan Kaufmann Publishers, Inc, 1991.

[Weiss98] Weiss S. and Indurkhya N., *Predictive Data Mining.* San Francisco, CA: Morgan Kaufmann Publishers, Inc, 1998.

[White97] White R., "Object Classification in Astronomical Images," *Statistical Challenges in Modern Astronomy II,* pp. 135-148, 1997.

[Whitle94] Whitley D., "A Genetic Algorithm Tutorial," *Statistics and Computing,* vol. 4, pp. 65-85, 1994.

[Windows] Microsoft Windows Home Page: http://www.microsoft.com/windows/default.asp

[Yang90] Yang T. and Gerasoulis A., "A Fast Static Scheduling Algorithm for DAG's on an Unbounded Number of Processors," Tech report, Rutgers University, 1990.

[Zelkow98] Zelkowitz, M. and Wallace, D., "Experimental Models for Validating Technology," *Computer*, May 1998.

VITA

Captain Lonnie Pafford Hammack was born on June 6, 1962 in Cuthbert, Georgia. He graduated from Terrell County High School in 1980. In September 1983, he enlisted in the United States Air Force and was assigned to Beale AFB, CA, where he worked as an aircraft engine technician. In March 1988, he separated from the Air Force to continue his education. From August 1989 to April 1993, he served as an avionics specialist with the Air Force Reserves at Robins AFB, GA. He received a Bachelor of Science in Computer Science from Albany State College in 1992 and was commissioned a second lieutenant in the Air Force on July 28, 1993. Lonnie's first assignment as a commissioned officer was to the Air Force Pentagon Communications Agency in Washington, D.C., where he worked as a system administrator and network engineer. Following his tour at the Pentagon, he was assigned as a Master's student to the Air Force Institute of Technology.