Cluster Computing and Performance Measurement

by

Mohd Azzart Moideen

Dissertation submitted in partial fulfillment of
The requirements for the
Bachelor of Engineering (Hons)
(Information Technology)

JULY 2004

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

t
QA
76.58
.M697
2004

1) Electronic data processing -- Distributed processing
2) Parallel processing (Electronic computers)

CERTIFICATION OF APPROVAL

Cluster Computing and Performance Measurement

by

Mohd Azzart Moideen

Final Draft submitted to the
Information Technology Programme
Universiti Teknologi PETRONAS
In partial fulfillment of the requirement for the
BACHELOR OF TECHNOLOGY (Hons)
(INFORMATION TECHNOLOGY)

Approved by,

____

(Mr. Suhaimi Abdul Rahman)

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK
November 2004

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

[illegible]

MOHD AZZART MOIDEEN

ABSTRACT

There is a continual demand for greater computational power from computer systems than is currently possible. Areas requiring great computational speed include numerical simulation of scientific and engineering problems. Such problems often need huge quantities of repetitive calculations on large amount of data to give valid results. Cluster computing offers many advantages as a highly cost-effective and often scalable approach for high-performance computing in general. To achieve the full potential of high performance computing systems, centralized configuration services are the starting point. For a large scale of projects, cluster computing is required where it is supposed to be optimized for the system topology and management of the project. This paper presents the consequences of using cluster computing and performance management and the consequences without this technology. The experimental results of this paper highlight the affects of the design of this service and provide a comprehensive performance analysis of the project.

ACKNOWLEDGEMENT

Bismillah ar-Rahmani Ar-Raheem

In the Name of Allah, The Most Compassionate, the Most Merciful

In order to complete this report, I have been doing researches over internet and books over cluster computing and distributed computing. In this Cluster Computing and Performance Measurement project, I would like to thank:

1) Mr Suhaimi Abdul Rahman, my **supervisor** (for giving me the guidelines and ways in producing a good output and full support in terms of knowledge input along this internship)

2) **The *Backbone* Of This FYP Committee** – Mr. Mohd Nor Ibrahim and Ms Vivien, and all IT/IS lecturers, (for giving full commitment in term of providing info about the final year project)

3) **Universiti Teknologi PETRONAS** – all UTP staff (for the full cooperation and providing me very convenient places to complete the project with the provided utilities)

4) **My parents**, Tn Hj. Moideen Ahmad and Pn Hjh Harizon Mohd Isa and **family** who supports me financially and mentally

5) **Friends**, for their support and informations over my FYP project

6) Those who are involved directly and indirectly towards the project.

ABBREVIATIONS AND NOMENCLATURES

MPI	:	Message Passing Interface
HASP	:	Houston Automatic Spooling Priority
PC	:	Personal Computer
COTS	:	Commodity-off-the-shelf
OS	:	Operating Systems
OTS	:	Off –the-shelf
HTML	:	Hyper Text Markup Language
NASA	:	National Aerospace Association
MPP	:	Massively Parallel Processors
SMP	:	Symmetric Multiprocessors
CC-NUMA	:	Cache-Coherent Nonuniform Memory Access
NIC	:	Network Interface Card
LSF	:	Load Sharing Facility
CODINE	:	COmputing In DIstributed Networked Environments
USB	:	Universal Serial Bus

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1. INTRODUCTION

Chapter 1 explains the fundamental information of the project, which consists of background of study, problem statement, objective and scope of the project. A brief explanation of 'Cluster Computing' is also included in this section.

1.1 Background of the project

This report is written as a pre-requisite for undergraduate students to complete the studies. 'Cluster Computing and Performance Measurement' is chosen the title for the author's final year project as it could help the author to enhance and in-depth with all the theories the author have learnt during his years of studies.

The initial idea leading to 'Cluster Computing' was developed in the 1960s by IBM as a way of linking large mainframes to provide a cost-effective form of commercial parallelism. During those days, IBM's HASP (Houston Automatic Spooling Priority) system and its successor, JES (Job Entry System), provided a way of distributing work to a user-constructed mainframe cluster. IBM still supports clustering operating system, middleware, and system management software to provide dramatic performance and cost improvements while permitting large mainframe users to continue to run their existing applications. The recent advances in these technologies and their availability as cheap and commodity components are making clusters or networks of computers (PCs, workstations, and SMPs) an appealing vehicle for cost-effective parallel computing.Clusters, built using commodity-off-the-shelf (COTS) hardware components as well as free, or commonly used, software, are playing major role in redefining the concept of super computing.

Clusters computing have emerged as a low cost solution providing a high performance and high availability in creating a super machine system. There are so many reasons why cluster computing is chosen instead of using distributed computing. Cluster supports both parallel and sequential programs. E.g. Emerging OSs like Solaris MC and Unixware. Cluster nodes have a strong nodes sense of membership (unlike distributed systems) Single point of entry : A user can connect to the cluster as a single system (like telnet Beowulf.myinstitute.edu), instead of connecting to individual nodes as in the case of distributed systems (like node1.beowulf.myinstitue.edu). The above property is exhibited by only CLUSTER COMPUTING. Today, a wide range of applications are hungry for higher computing power, even though single processor PCs and workstations now can provide extremely fast processing, the even faster execution that multiple processors can achieve by working concurrently is still needed.

Now, finally, costs are falling as well. Networked clusters of commodity PCs and workstations using off-the-shelf processors and communication platforms such as Myrinet, Fast Ethernet and etc are becoming increasingly cost effective and popular. This concept, known as cluster computing, will continue to flourish; clusters can provide enormous computing power that a pool of users can share or that can be collectively used to solve a single application.

1.2 Problem statement

Very often applications need more computing power than a sequential computer can provide. One way of overcoming this limitation is to improve the operating speed of processors and other components so that they can offer the power required by computationally intensive applications. Even though this is currently possible to a certain extent, future improvements are constrained by the speed of light, thermodynamic laws, and the high financial costs for processor fabrication. A viable and cost-effective alternative solution is to connect multiple processors together and coordinate their computational efforts. The resulting systems are popularly known as parallel computers, and they allow the sharing of a computational task among multiple processors.

In this cyber era, speed has been increasing rapidly. People are chasing towards this in order to reduce costs and time. Computing is an evolutionary process. Five generations of development history with each generation improving on the previous one in terms of technology, architecture, software, applications, and representative systems. As part of this evolution, computing requirements driven by applications have always outpaced the available technology. So system designers have always needed to seek faster, more cost effective computing systems and environment. The problem with coping the speed in processing, programming and etc can be done by using cluster computing. Cluster computing provides the best solution, by offering computing power that greatly exceeds the technological limitations of single processor systems.

1.3 Objective

This project is done in two phases. In the first phase, the research has been done to understand the concept of cluster computing and the performance measurement. In the second phase, it concentrates on the design and development of the cluster computing and how to perform the measurement. Objectives of the project to be achieved are as follow:

1) To make a research on understanding on how cluster computing works and how it will be implemented
2) To setup the network which will consist of several PCs for cluster computing
3) To make a measurement **using time between sequential and parallel processing.**

1.4 Scope of study

Clusters offer the following features at a relatively low :

- High performance
- Expandability and Scalability
- High Throughput
- High Availability

The research area for this project will cover cluster computing and the performance measurement. This project will be implemented in Data Communication lab. The cluster will be able to prove extra speed and performance in processing data. A program of calculating numerical method (engineering mathematics) will be created where it will be tested on both cluster PCs and PC with a single processor. Basically, the project will show how fast and reliable cluster PCs are in processing data.

CHAPTER 2

LITERATURE REVIEW AND THEORY

2.1 Literature Review

This chapter contains the acknowledged findings on this field, consisting of relevant theories, hypothesis, facts and data which are relevant to the objective and the research of this project.

Researcher [1] David Abramson of Monash University during a talk "From PC Cluster to a global Computational Grid," indicates that by taking a global view, the scheduling system uses various kinds of parameters to determine a scheduling policy for optimally completing an application execution. These parameters include resource architecture and configuration, resource capability, resource requirements, priority, network latency and bandwidth, reliability of resources, contention, user preference, application deadline, and user willingness to pay for resource usage.

Researcher [2] Thomas Sterling of the California Institute of Technology and NASA's Jet Propulsion Laboratory presented the cluster system used at JPL. He gave a broader perspective on cluster computing and talked about current trends leading to very large-scale clusters that can deliver teraflop or even petaflop performance. Building such systems will require SOCs (systems-on-a-chip), gigahertz processor clock rates, VLIW (very long instruction word) architecture, Gbit DRAMs, On board microdisks, optical fiber and wave division multiplexing communication platform.

Researcher [3] Pfister points out, there are three ways to improve performance which are work harder, work smarter and get help. In terms of computing technologies, the analogy to this mantra is that working harder is like using faster software (high performance processors or peripheral devices). Working smarter concerns doing things more efficiently and this revolves around the algorithms and techniques used to solve computational tasks.Finally, getting help refers to using multiple computers to solve a particular task.

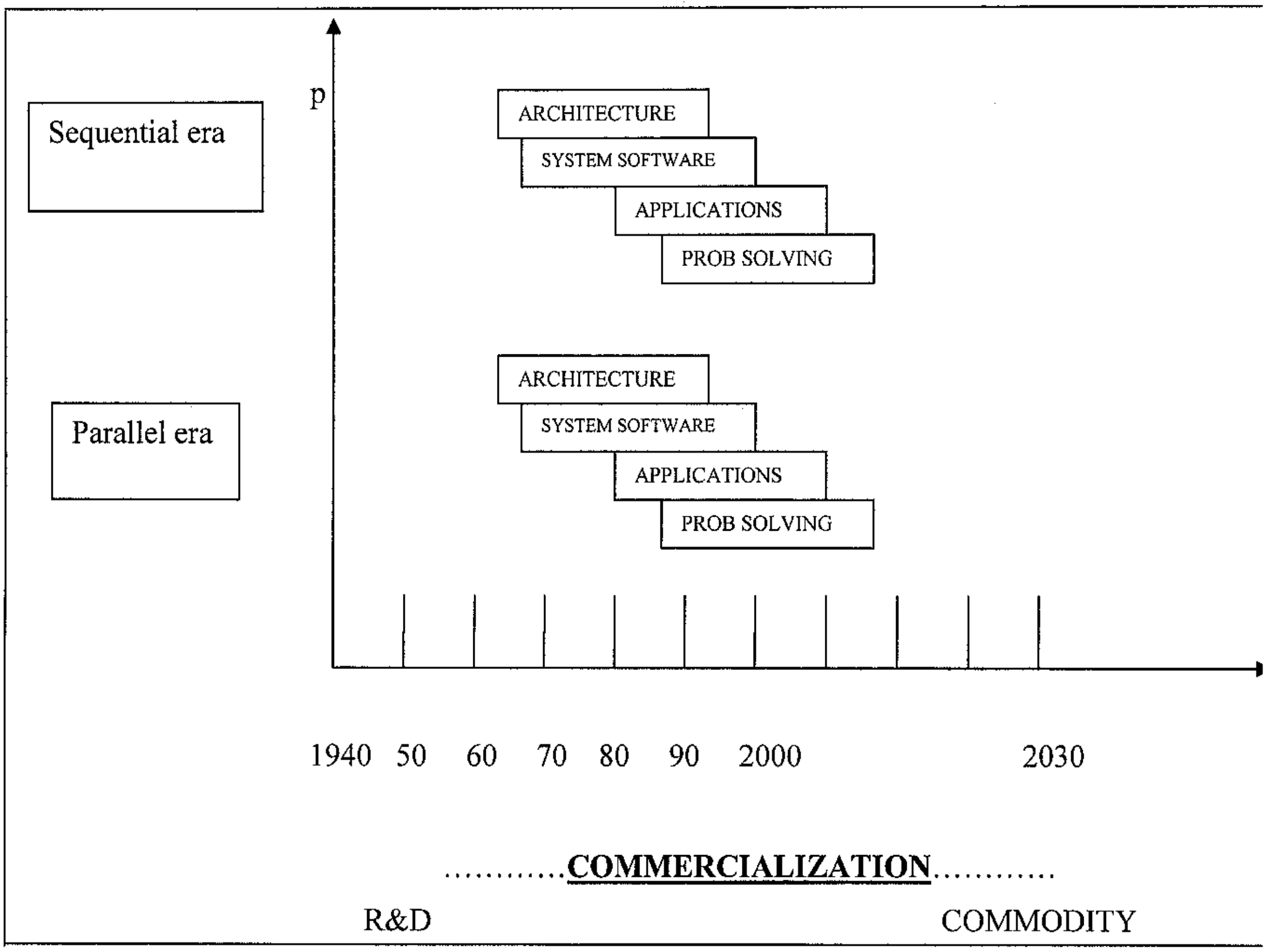


FIGURE 2.1 Two eras of computing.

Table 2.1 shows a modified version comparing the architectural and functional characteristics of these machines originally given in [4] by Hwang and Xu.During the past decade many different computer systems supporting high performance computing have emerged. Their taxonomy is based on how their processors, memory, and interconnect are laid out. The most common systems are :

- Massively Parallel Processors (MPP)
- Symmetric Multiprocessors (SMP)
- Cache-Coherent Non-uniform Memory Access (CC-NUMA)
- Distributed Systems
- Clusters

Researcher [5] Rajkumar from Monash University says a cluster is a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers working together as a single, integrated computing resource. A computer node can be single or multiprocessor system (PCs, workstations, or SMPs) with memory, I/O facilities, and an operating system. A cluster generally to two or more computers (nodes) connected together.The nodes can exist in a single cabinet or be physically separated and connected via a LAN. An interconnected (LAN-based) cluster of computers can appear as a single system to users and applicatios. Such a system can provide a cost-effective way to gain features and benefits (fast and reliable services) that have historically only on more expensive proprietary shared memory systems.

Researcher [6] Prof Michael Allen, a full professor in the Department of Computer Science at the University of North Carolina at Charlotte, indicates that there is a continual demand for greater computational power from computer systems than is currently possible. Areas requiring great computational speed include numerical simulation of scientific and engineering problems. Such problems often need huge quantities of repetitive calculations on large amounts of data to give valid results. Calculations must be completed within a 'reasonable' time period. In manufacturing realm, engineering calculations and simulations must be achieved within seconds or minutes if possible. A simulation that takes two weeks to reach a solution is usually unacceptable in a design environment, because the time has to be short enough for the designer to work effectively. As systems become more complex, it takes more time to simulate them. There are some problems that have specific deadline for the computations, for example, weather forecasting. Taking two days to forecast the local weather accurately for the next day would make the prediction useless.Some areas, such as modeling large DNA structures and global weather forecasting, are grand challenge problems. A grand challenge problem is one that cannot be solved in a reasonable amount of time with today's computer.

Researcher [7] Professor Barry Wilkinson of Western Carolina University indicates that apart from obtaining the potential for increased speed on an existing problem, the use of multiple computers/processors often allows a larger problem or a more precise solution of a problem to be solved in a reasonable amount of time. For example, computing many physical phenomena involves dividing the problem into discrete solution points. As we have mentioned, forecasting the weather involves dividing the air into a three- dimensional grid of solution points occur in many other applications. A multiple computer or multiprocessor solution will often allow more solution points to be computed in a given time, and hence more precise solution. A related factor is that multiple computers very often have more total main memory than a single computer, enabling problems that require larger amounts of main memory to be tackled.

CHAPTER 3

3.0 METHODOLOGY AND PROJECT WORK

Chapter 3 features the detailed description of methodology and procedure of completing this project. This methodology is implemented in order to ensure that the project is running as required. An overview of the network is also described in this chapter.

3.1 Project Methodology

Methodology plays a vital role in completing any project. This methodology consists of 5 important phases that are:

a. Planning
b. Analysis
c. Design
d. Implementation &testing
e. Operation and support

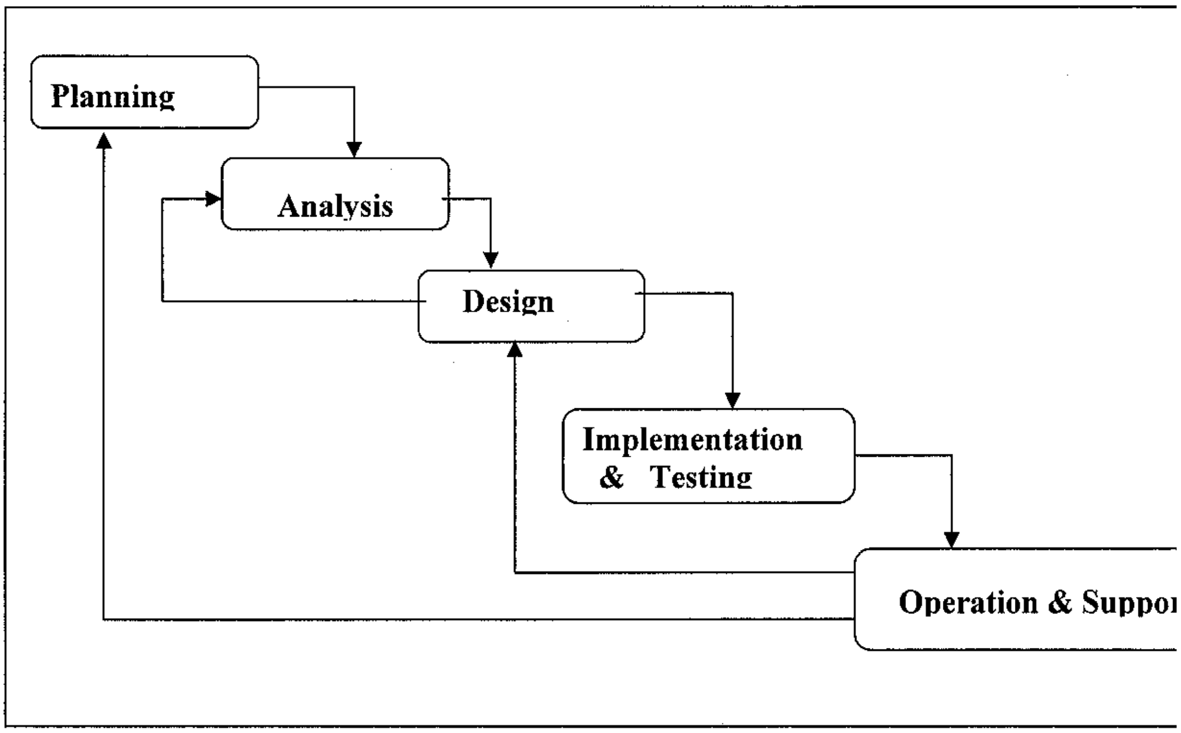


Figure 3.1 shows the cluster computing development life cycle

3.1.1 Planning

System planning begins with a formal proposal or request for the project. In this phase, the purpose is to identify clearly the nature and scope of the business opportunity or problem by performing preliminary investigation or also called as feasibility study. The outcome from this study is project scope. This preliminary investigation is a critical step since the outcome will affect the entire of our development process.

At this phase, the project started with the request from the lecturers to submit the project proposal. As discussed with the author's supervisor, this topic was selected since it is an interesting topic to discover. During this stage, a proposal was sent to the FYP committee for approval. Scope of studies was also established during this period.

Note: Please view appendix for Gantt Chart

3.1.2 Analysis

The purpose of this phase is to understand the requirements and build a logical model for the cluster computing environment. As implementing the project, this is the phase of doing research and analysis. Information, data and findings were collected as much as possible during this stage.

The software and hardware for the project are identified during this phase.

Hardware

For each PC:

- Pentium 4 processor 2.0 GHz or AMD Athlon 2.0 Ghz processor
- 512 Mb RAM
- CDROM 52X
- Hard disk with at least 20 G of space
- Minimum 32 bit Graphic Card
- USB port
- Portable USB drive minimum of 128 Mb of space

Software

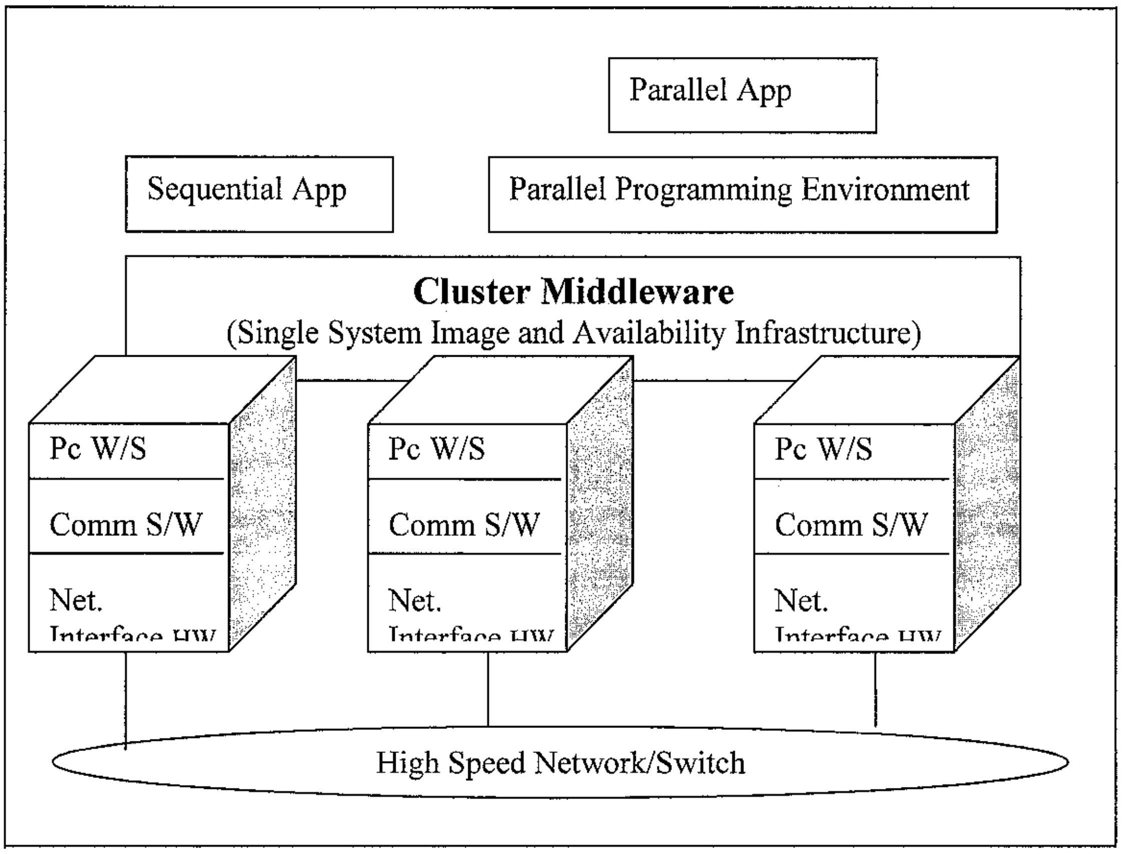
Using Linux, for each PC :

- Red Hat Linux 9.0 (shrike)
- GNU++ compiler
- PICO or LV for script editing
- Message Passing Interface software

Using windows,for each PC:

- Microsoft Windows 2000 or XP Professional SP1
- MPICH installer
- ADOBE acrobat reader
- Microsoft Visual C++ compiler

This is the example of current planning of the architecture

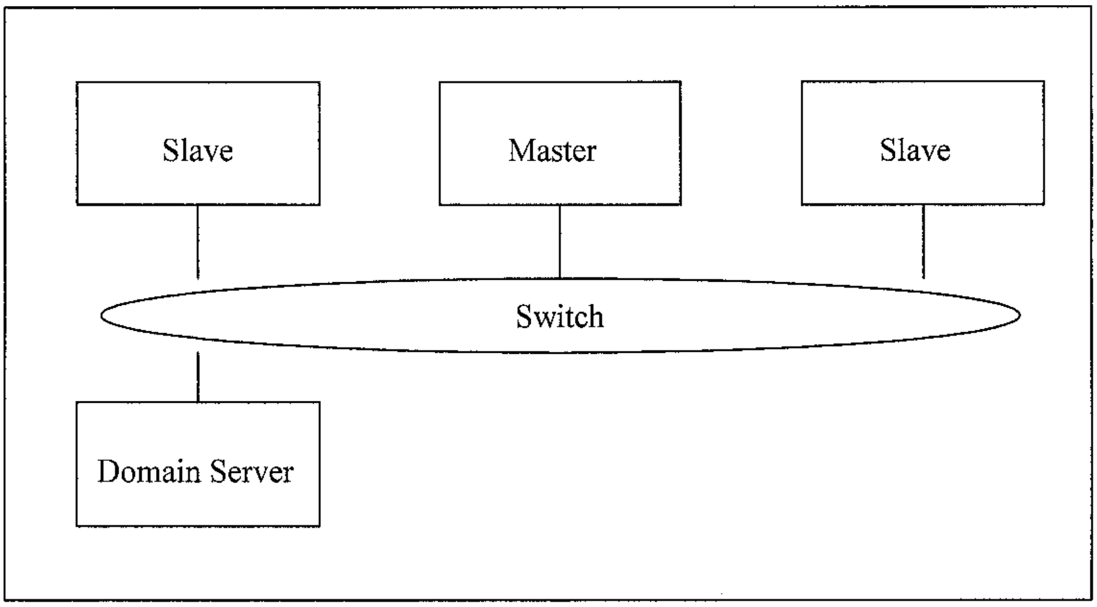


The following are some prominent components of cluster computers :

- Multiple High Performance Computers (PCs,Workstations,or SMPs)
- State-of-the-art Operating Systems (Layered or Micro-Kernel based)
- High Performance Networks/Switches
- Network Interface Cards (NICs)
- Fast Communication Protocols and Services
- Cluster Middleware
    - Hardware (such as DEC Memory Channel,hardware DSM, and SMP techniques)
    - Operating System Kernel or Gluing Layer (such as Solaris MC and GLU-nix)
    - Applications and subsystems
        - Applications (such as system management tools and electronic forms)
        - Runtime systems (such as software DSM and parallel file system)
        - Resource Management and scheduling software (such as LSF (Load Sharing Facility) and CODINE (COmputing In DIstributed Networked Environments))
    - Parallel Programming Environments and tools (such as PVM and MPI)
    - Applications
        - Sequential
        - Parallel or Distributed

3.1.3 Design

In this phase, all necessary inputs, outputs,, interfaces and processes were identified. The tools needed for the design phase were downloaded and installed. PCs will be setup as clusters and single processing PC as control experiment.
Below is the design for Cluster computing in Data Communication lab.



- For starting, 3 Pentiums with 4 2.0 Ghz will be used, with one domain server
- There will be one master and two slaves
- Each PCs will be interconnected using network switch with unique Internet Protocol address
- Next, the Operating System of each PC is installed.
- The software, compiler etc will be setup too.
- The programming stage will start after this stage.

3.1.4 Implementation

During the implementation phase, the system is constructed. The documents are written, tested and documented, and the cluster is setup and programmed. The implementation will be based on the design made earlier.

There will be two choices in implementing the cluster computing environment based on operating systems:

- Using Microsoft Windows 2000 or XP
    - The library will be Message Passing Interface (MPI)
    - The compiler and programming platform will be visual C++
    - The programming language used is C language

- Using Linux Redhat 9.0 (shrike)
    - The library will be Message Passing Interface (MPI)
    - The compiler will be GCC or GNU++
    - The programming platform will be pico or vi which available automatically with the operating systems.
    - The programming language used is C language

3.1.5 System Operation and Support

During system operation and support, the maintenance will maintain and enhance the cluster computing system. Maintenance changes correct errors and adapt to changes in the environment, while enhancements provide new features and benefits. The objective during this phase is to maximize return on the IT/IS investment. The system developed shall be a well-designed system that is reliable, maintainable and scaleable. During this stage, the drawback of the system will be identified and future enhancements will be made.

3.2 Why MPI?

MPI (Message-Passing Interface) is a standard specification for message-passing libraries. MPICH is a portable implementation of the full MPI specification for a wide variety of parallel computing environments. MPICH contains, along with the MPI library itself, a programming environment for working with MPI programs. The programming environment includes a portable startup mechanism, several profiling libraries for studying the performance of MPI programs, and an X interface to all of the tools.

The current release supports tcp/ip and shared memory connections. Interprocess communication on a single machine is done through shared memory queues and communication between processes on separate machines is done through sockets.The code provided can be compiled using MS Visual C++ 6.0 and Visual Fortran 6.0. The dlls provided have the C and Fortran.

CHAPTER 4

4.0 Results and Discussions

This chapter compiles the current findings of the project work. There are several important and informative facts and information that comes from journals and online resources.

Processors	Sequential	Parallel
Time taken (*s*)	11.1156 s	3.5674s

Note : The number entered for both processing are 9 digits numbers

Table 4.1.1 : Results for numerical method calculation using sequential and parallel processors

4.1 Results

Based on the results of the tested sequential and parallel processing, the author found that parallel processing has much more better performance and faster processing. In his programming, the author have been using timer in computing a complex mathematical calculation using both sequential and parallel processing. Parallel processing and programming took much more less time to complete the task as it divide the task to the nodes (using three PCs and three Processors) rather than using a sequential processing and programming. The approximate time for sequential programming to calculate nine digits numbers of error in numerical method calculation took about nearly 11.11 seconds while parallel programming took only about 3 seconds. It is proven that using parallel programming and processing is much more efficient in cost and time consuming.

4.2 Discussions on potential for increased computational speed

For further explanation, the number of processes or processors will be identified as *p.* I will use the term “multiprocessor” to include all parallel computer systems that contain more than one processor.

4.2.1 Speedup factor

The first point of interest when developing solutions on a multiprocessor is the question of how much faster the multiprocessor solves the problem under consideration. In doing this comparison, one would use the best solution on the single processor, that is, the best sequential algorithm on the single processor system compare against the parallel algorithm under investigation on the multiprocessor.The *speedup factor,S(p),* is a measure of relative performance, which is defined as :

S(p) = Ts/ Tp

Ts = Execution time using single processor system (with the best sequential algorithm)
Tp = Execution time using a multiprocessor with p processors

Note : The speed up factor is normally a function of both p and the number of data items being processed.

S(p) gives the increase in speed in using the multiprocessor. Note that the underlying algorithm for the parallel implementation might not be the same as the algorithm on the single-processor system (and is usually different)

In theoretical analysis, the speedup factor can also be cast in terms of computational steps :

$$S(p) = \frac{\text{No. of computational steps using one processor}}{\text{No of parallel computational steps with } p \text{ processors}}$$

The maximum speedup possible is usually p with p processors (*linear speedup*). The speedup of p would be achieved when the computation can be divided into equal-duration processes, with one process mapped onto one processor and no additional overhead in the parallel

4.2.2 Efficiency

It is sometimes useful to know how long processors are being used on the computation,which can be found from the (system) efficiency. The efficiency,*E* is defined as

E= Ts/TpXp

Where Ts = Execution time using one processor

Tp = Execution time using multiprocessor

P = number of processors

Which leads to

$$E = \frac{S(p)}{p} \times 100\%$$

When E is given as a percentage. For example, if E = 50%, the procesors are being used half the time on actual computation, on average.The efficiency of 100% occurs where all the processors are being used on computations at all times and the speedup factor S(*p*), is *p*.

4.2.3 What is the Maximum Speedup?

Several factors will appear as overhead in the parallel version and limit the speedup, notably.

- Periods when not all the processors can be performing useful work and are simply idle
- Extra computations in the parallel version not appearing in the sequential version; for example, to recomputed constants locally.
- Communication time between processes.

It is reasonable to expect that some part of a computation cannot be divided into concurrence processes and must be perform sequentially. Let us assume that during some period perhaps an initialization period or the period before concurrent processes are set up, only one processor is doing useful work, and for the rest of the computation additional processors are operating on processes.

Assuming there will be some parts that are only executed on one processor, the ideal situation would be for all the available processors to operate simultaneously for the other times. If the fraction of the computation that cannot be divided into concurrent parts, the time to perform the computation with p processors is given by $ft_s + (1\text{-}f)t_s / p$, as illustrated in figure 4..1.Illustrated is the case with a single serial part at the beginning of the computation, but serial part could be distributed throughout the computation. Hence, the speedup factor given by

$$S(p) = \frac{ts}{ft\,s + (1\text{-}f)ts/p} = \frac{p}{1 + (p-1)f}$$

The equation known as Amdahl's law (Amdahl,1967). Figure 4.1 shows S(*p)* plotted against number of processors and against f. We see that indeed a speed improvement is indicated. However, the fraction of the computation if a significant increase in speed is to be achieved. Even with an infinite number of processors, the maximum speedup is limited to 1/f

For example, with only 5% of the computation being serial, the maximum speedup is 20, irrespective of the number of processors. Amdahl used this argument to promote single-processor systwms in 1960s.Of course, one can counter this by saying that even speedup of 20 would be impressive.

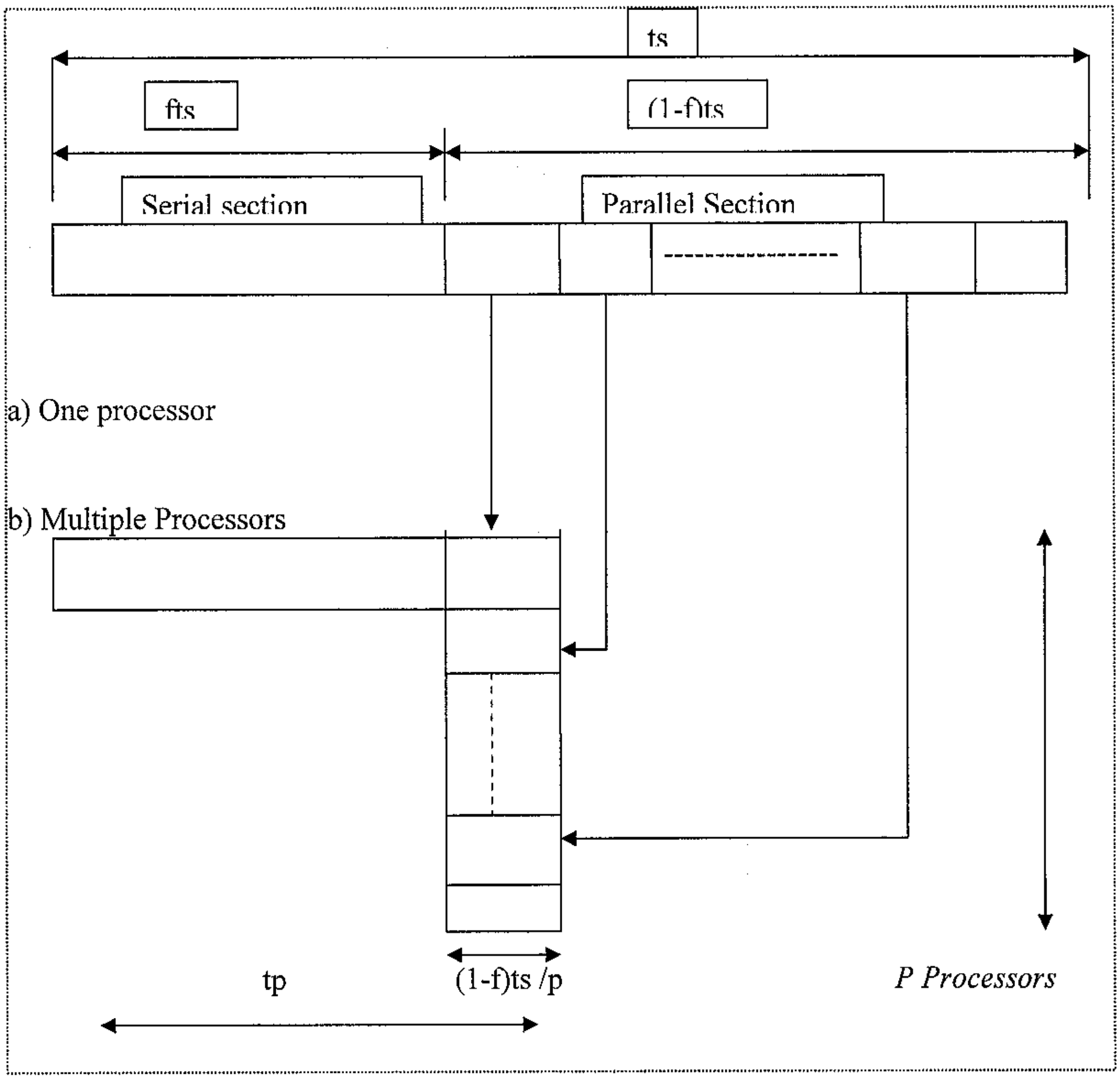
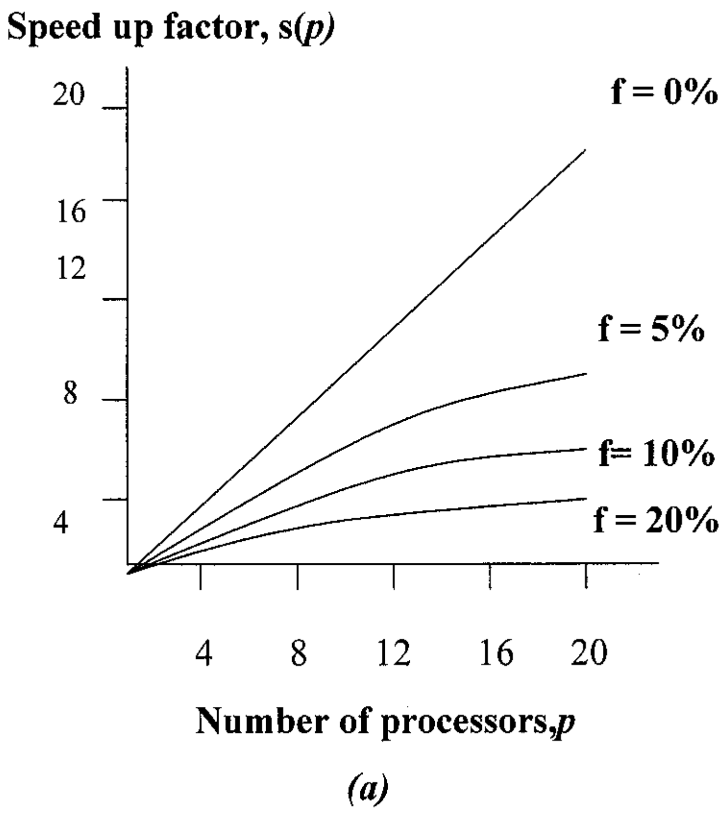


Figure 4.1 Parallelizing sequential problem – Amdahl's Law



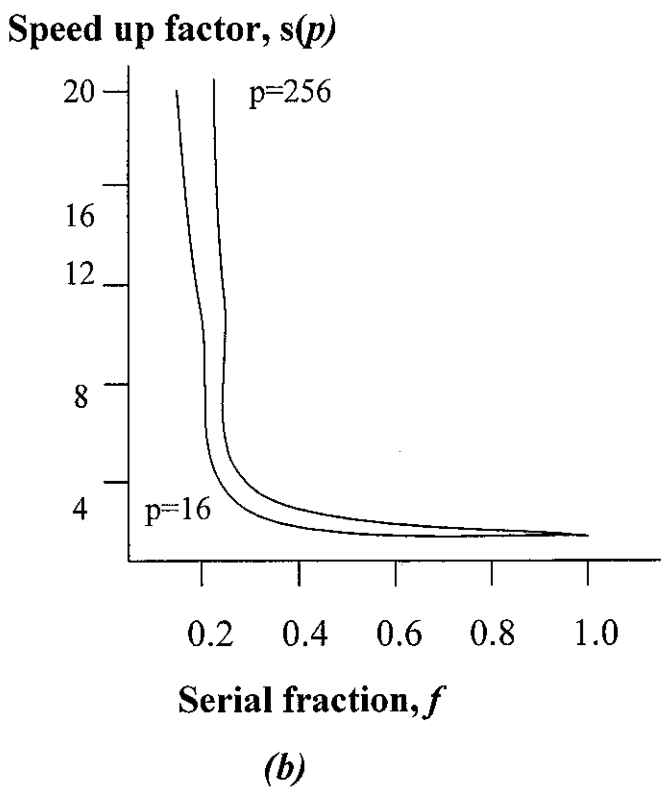


Figure 4.2 (a) Speedup against number of processors (b) Speed up against serial fraction

CHAPTER 5

5.0 Conclusion

There is a continual demand for greater computational power from computer systems than is currently possible. Areas requiring great computational speed include numerical simulation of scientific and engineering problems .Such problems often need huge quantity of repetitive calculations on large amount of data to give valid results. Computations must be completed within a 'reasonable' time. Therefore, cluster computing is the solution.

These cluster computing developments have created a beehive of activity throughout the world – new books, workshops and conferences, with participation from both academic and industry. Today, a wide range of applications are hungry for higher computing power, and even though single processor PCs and workstations now can provide extremely fast processing, the even faster execution that multiple processors can achieve by working concurrently is still needed.

5.1 Recommendation

There are many ways to enhance and improve this project. Below are the recommendations for the project and future enhancements:

i) The project should have interface instead of using command prompt in controlling it.

ii) The project should be able to run on multiple and different Operating Systems, e.g Windows and Linux run simultaneously as slaves.

iii) The research of the project should include details on differentiations in prices, performances, constraints and etc. between Supercomputers and Cluster PCs.

REFERENCES

[1] Users' Guide to mpich, a Portable Implementation of MPI
http://www-fp.mcs.anl.gov/~lusk/papers/mpich-guide/paper.html

[2] **MPICH**-V - Introduction
www.lri.fr/~gk/MPICH-V/

[3] Portable MPI Model Implementation over GM Version **mpich**-1.2.6..13b **...**
www.myri.com/scs/READMES/README-**mpich**-gm

[4] Mark Chu-Carroll and Lori Pollock, ``Composite Tree Parallelism: Language Support for General Purpose Parallel Programming'', Journal of Programming Languages, Vol. 5, Issue 1, pp. 1-36, 1997
Available at www.eecis.udel.edu/~pollock/**cluster**.html

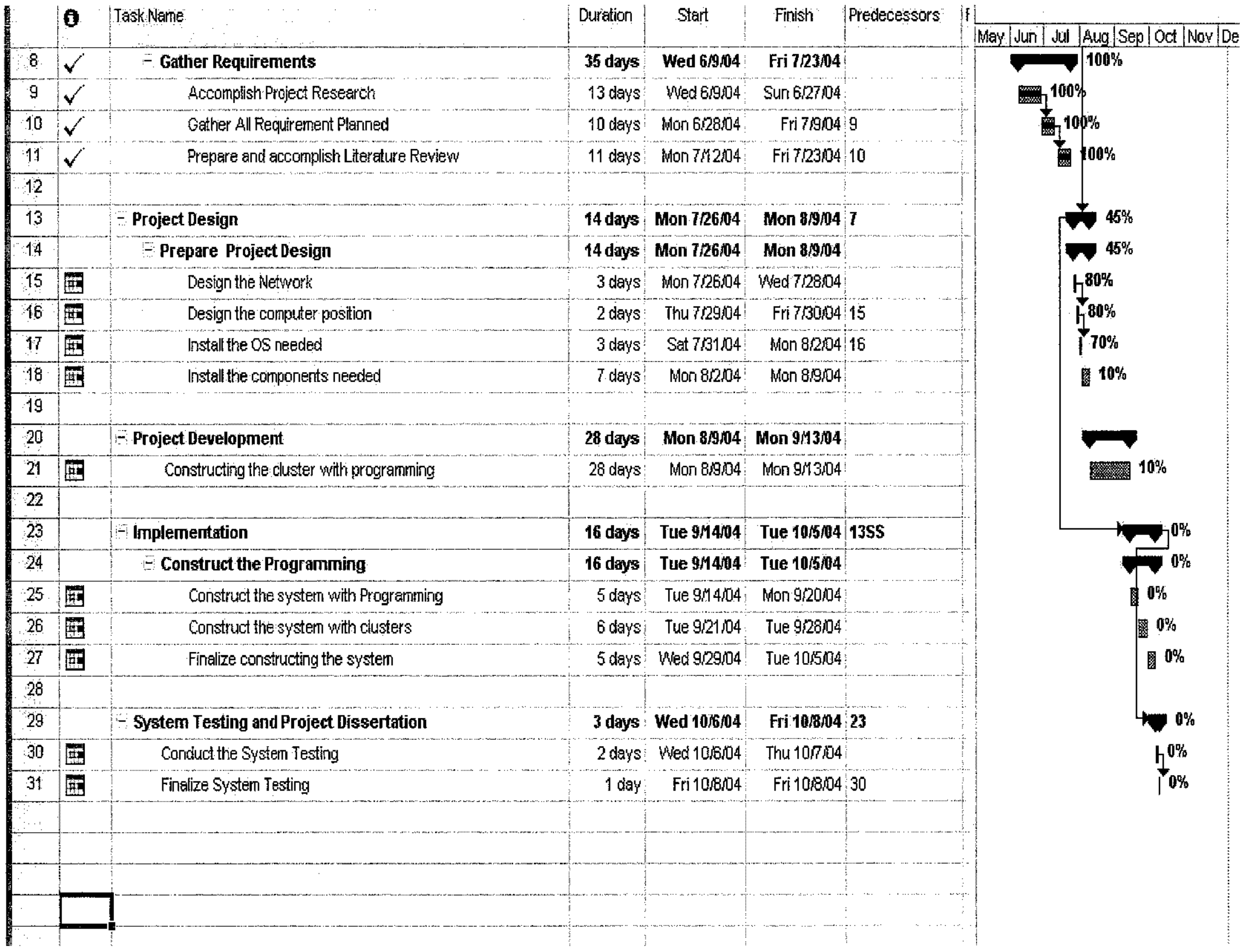
[5] Rajkumar Buyya 2000"High Performance Cluster Computing: Programming and Applications, Volume 2" Prentice Hall

[6] Rajkumar Buyya 2000 "High Performance Cluster Computing: Architecture and system design, Volume 1" Prentice Hall

[7] Barry Wilkinson "Parallel Programming – Techniques and Applications Using Networked Workstations and parallel Computers, second edition" Prentice Hall

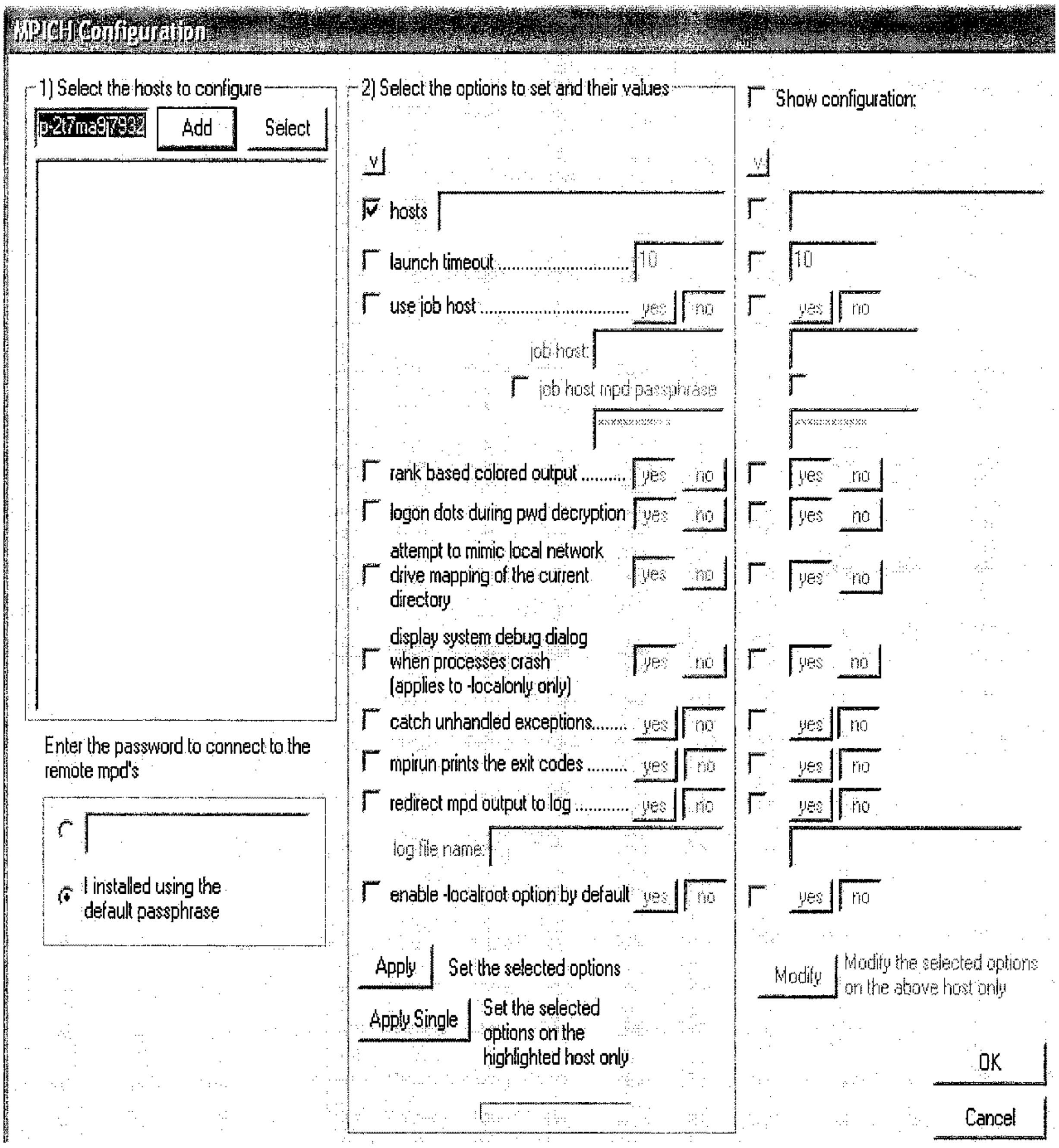
Appendices

Appendix 1

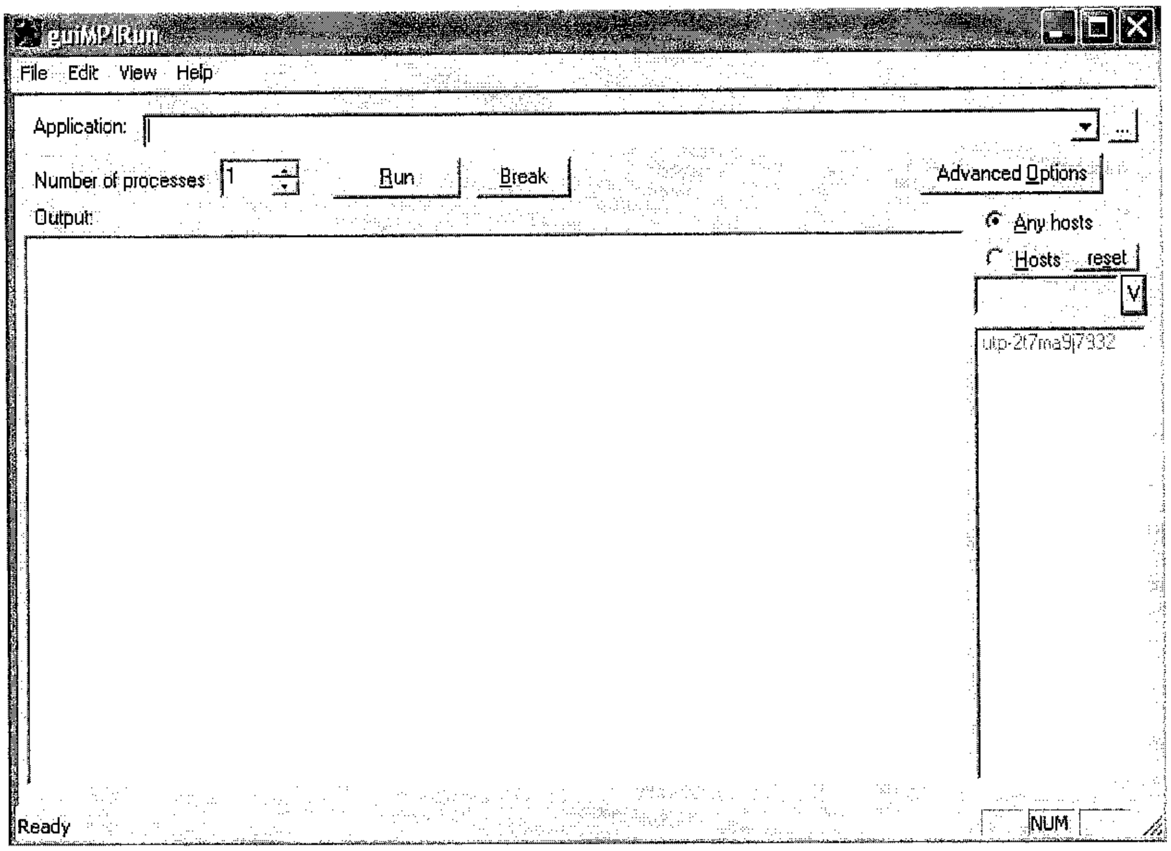


		Task Name	Duration	Start	Finish	Predecessors
8	✓	**Gather Requirements**	**35 days**	**Wed 6/9/04**	**Fri 7/23/04**	
9	✓	Accomplish Project Research	13 days	Wed 6/9/04	Sun 6/27/04	
10	✓	Gather All Requirement Planned	10 days	Mon 6/28/04	Fri 7/9/04	9
11	✓	Prepare and accomplish Literature Review	11 days	Mon 7/12/04	Fri 7/23/04	10
12						
13		**Project Design**	**14 days**	**Mon 7/26/04**	**Mon 8/9/04**	**7**
14		**Prepare Project Design**	**14 days**	**Mon 7/26/04**	**Mon 8/9/04**	
15	▦	Design the Network	3 days	Mon 7/26/04	Wed 7/28/04	
16	▦	Design the computer position	2 days	Thu 7/29/04	Fri 7/30/04	15
17	▦	Install the OS needed	3 days	Sat 7/31/04	Mon 8/2/04	16
18	▦	Install the components needed	7 days	Mon 8/2/04	Mon 8/9/04	
19						
20		**Project Development**	**28 days**	**Mon 8/9/04**	**Mon 9/13/04**	
21	▦	Constructing the cluster with programming	28 days	Mon 8/9/04	Mon 9/13/04	
22						
23		**Implementation**	**16 days**	**Tue 9/14/04**	**Tue 10/5/04**	**13SS**
24		**Construct the Programming**	**16 days**	**Tue 9/14/04**	**Tue 10/5/04**	
25	▦	Construct the system with Programming	5 days	Tue 9/14/04	Mon 9/20/04	
26	▦	Construct the system with clusters	6 days	Tue 9/21/04	Tue 9/28/04	
27	▦	Finalize constructing the system	5 days	Wed 9/29/04	Tue 10/5/04	
28						
29		**System Testing and Project Dissertation**	**3 days**	**Wed 10/6/04**	**Fri 10/8/04**	**23**
30	▦	Conduct the System Testing	2 days	Wed 10/6/04	Thu 10/7/04	
31	▦	Finalize System Testing	1 day	Fri 10/8/04	Fri 10/8/04	30

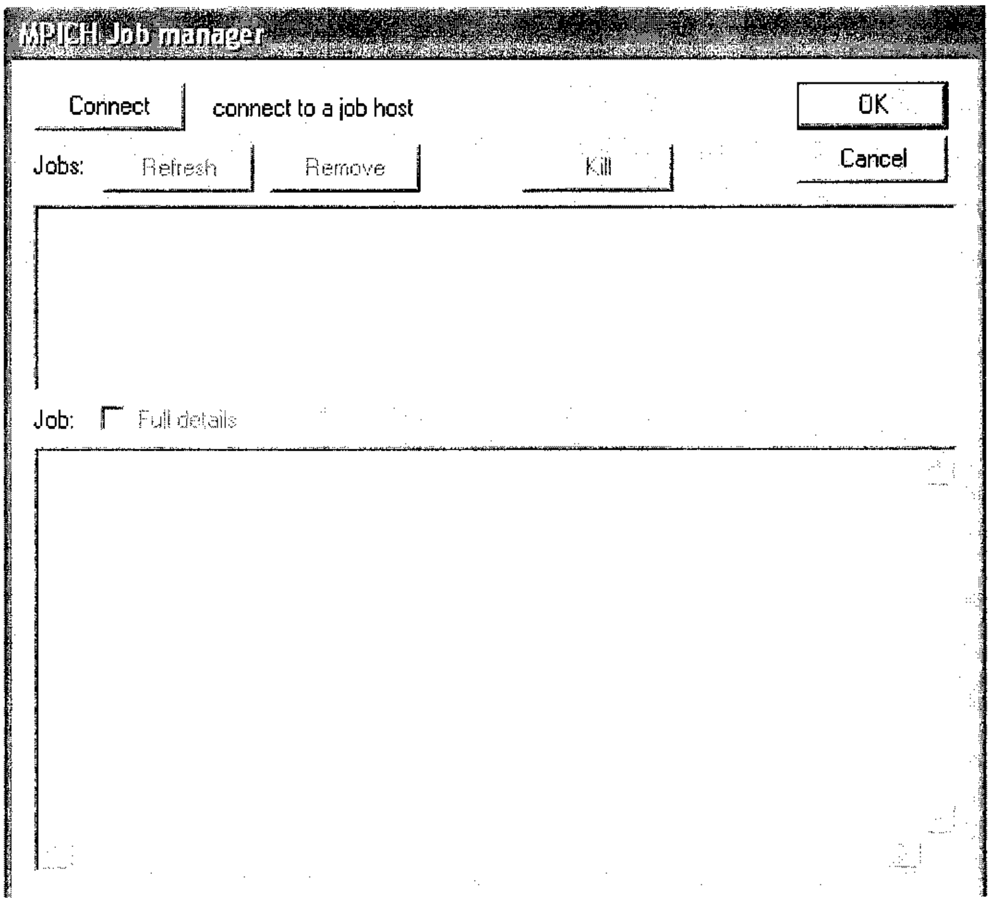
Appendix 2


MPICH Configuration
1) Select the hosts to configure
p-2t7ma97932
Add
Select
Enter the password to connect to the remote mpd's
I installed using the default passphrase
2) Select the options to set and their values
hosts
launch timeout
10
use job host
yes
no
job host:
job host mpd passphrase
rank based colored output
logon dots during pwd decryption
attempt to mimic local network drive mapping of the current directory
display system debug dialog when processes crash (applies to -localonly only)
catch unhandled exceptions
mpirun prints the exit codes
redirect mpd output to log
log file name:
enable -localroot option by default
Apply
Set the selected options
Apply Single
Set the selected options on the highlighted host only
Show configuration:
Modify
Modify the selected options on the above host only
OK
Cancel

Appendix 3


guiMPIRun
File Edit View Help
Application:
...
Number of processes 1
Run
Break
Advanced Options
Output:
Any hosts
Hosts
reset
V
utp-2t7ma9j7332
Ready
NUM

Appendix 4


MPICH Job manager
Connect
connect to a job host
OK
Jobs:
Refresh
Remove
Kill
Cancel
Job:
Full details

Appendix 5

Here are the steps in setting up MPI for windows:

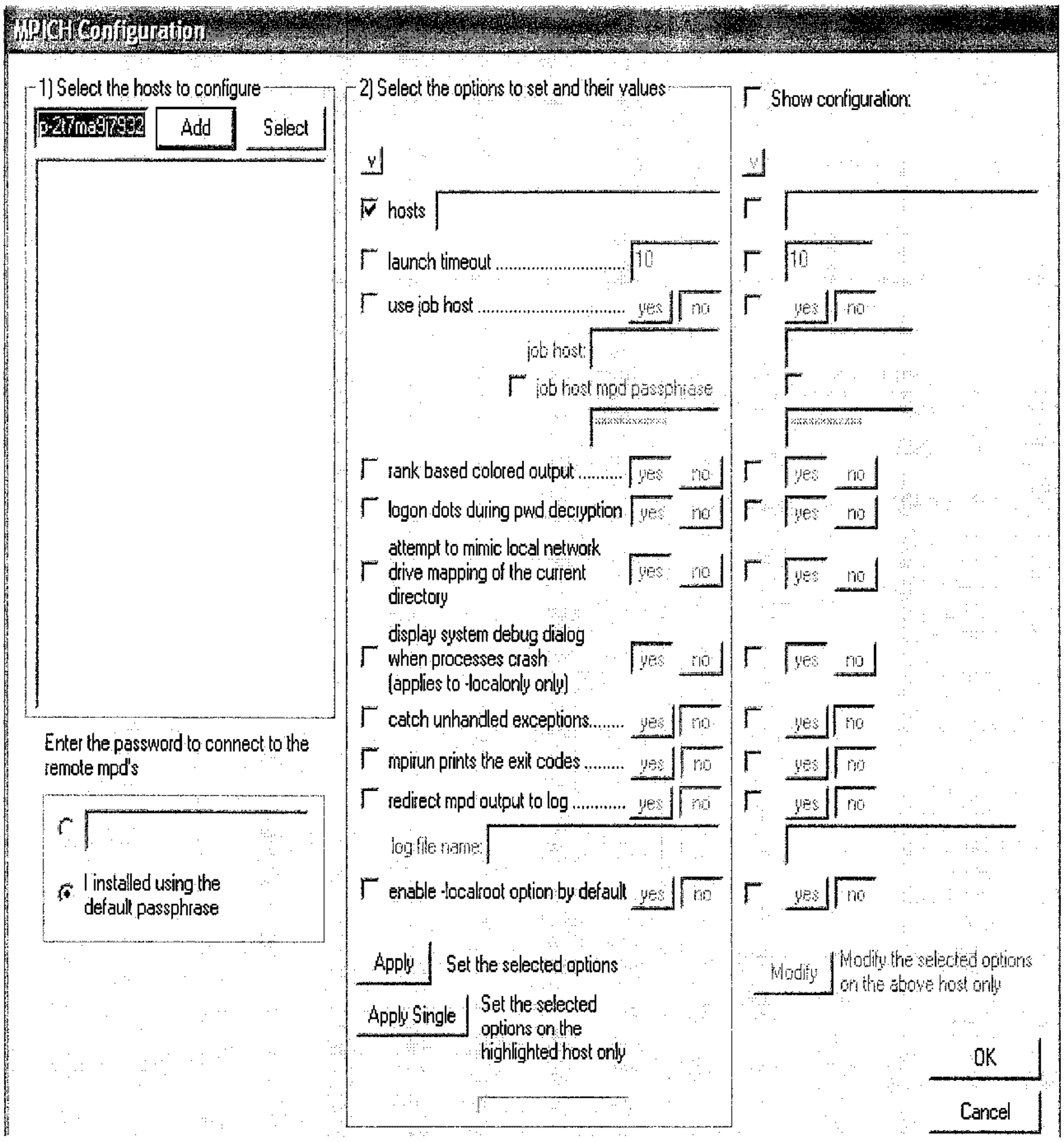
1) Prepare at least 3 PCs with interconnected on a switch with isolated Internet Protocol address
2) Download MPICH installer
3) Install the installer on each node
4) Configure the MPICH configuration files

5) Compile an MPI application

- After executing mpich.nt.1.2.x.exe, two SDK directories will be created
- SDK contains the libraries and include files necessary to compile an mpi application with MS Visual C++ 6 and/or Visual Fortran 6.

6) Run an application.

- Each launcher provides an application launcher called MPIRun.exe. Instructions and command options can be found in the users manual and a short list of options is presented by executing mpirun -help.
- If you installed the binary distribution and selected all the defaults then these are the steps to get mpirun to work:
    - After running setup on all the nodes, execute MPIConfig.exe (Start=>Programs=>MPICH=>mpd=>mpich configuration tool)
        - Select all the nodes where you installed mpich.
        - Click Apply. This will set the hosts entry on all the nodes



MPICH Configuration

1) Select the hosts to configure

p-2t7ma3j7932 Add Select

Enter the password to connect to the remote mpd's

○

[✓ I installed using the default passphrase]

2) Select the options to set and their values

[✓] hosts

☐ launch timeout 10

☐ use job host .... yes no

job host:

☐ job host mpd passphrase

☐ rank based colored output .... yes no

☐ logon dots during pwd decryption yes no

☐ attempt to mimic local network drive mapping of the current directory yes no

☐ display system debug dialog when processes crash (applies to -localonly only) yes no

☐ catch unhandled exceptions........ yes no

☐ mpirun prints the exit codes .... yes no

☐ redirect mpd output to log .... yes no

log file name:

☐ enable -localroot option by default yes no

Apply Set the selected options

Apply Single Set the selected options on the highlighted host only

☐ Show configuration:

☐

☐ 10

☐ yes no

☐

☐ yes no

☐ yes no

☐ yes no

☐ yes no

☐ yes no

☐ yes no

☐ yes no

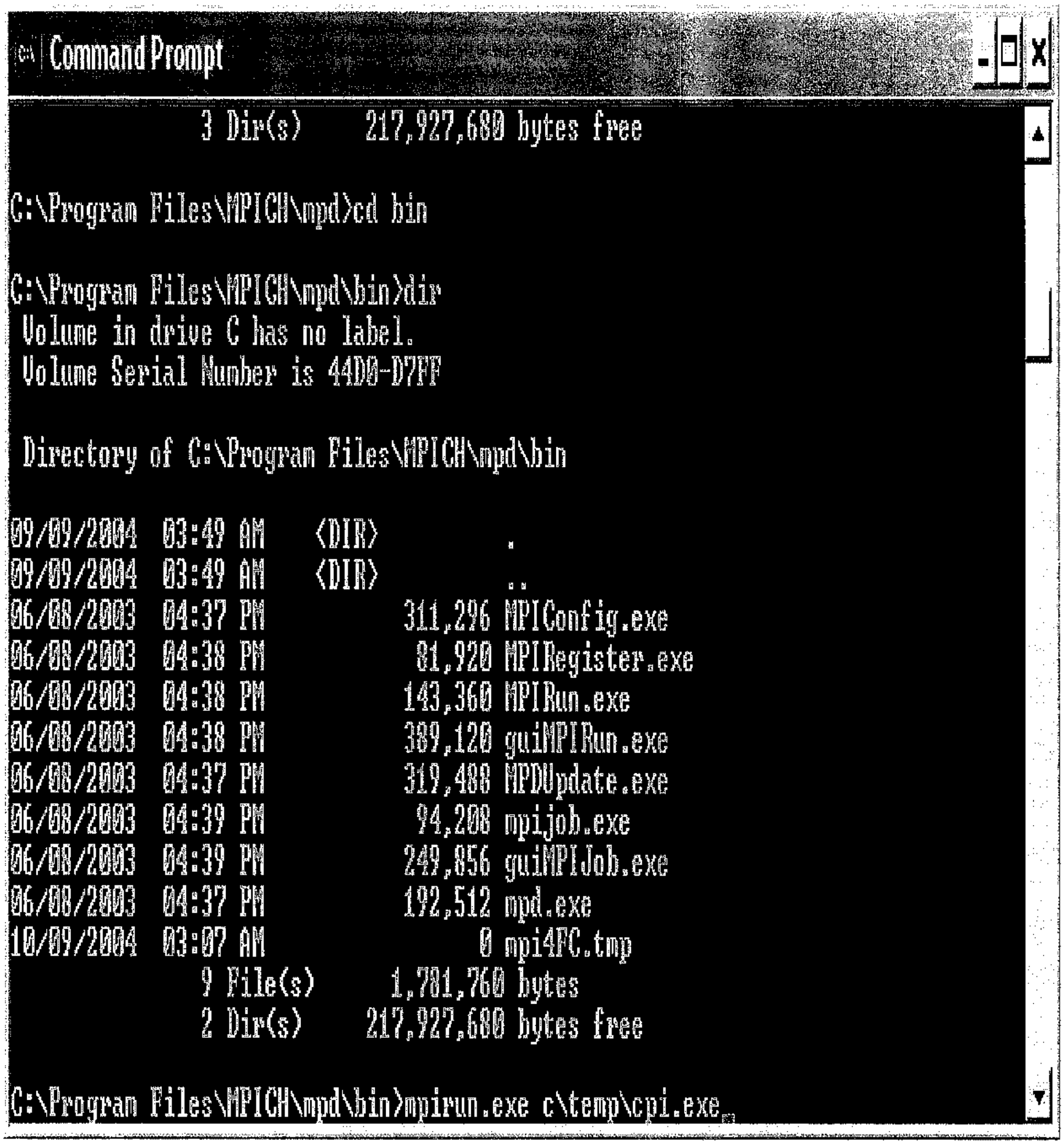
☐ yes no

Modify Modify the selected options on the above host only

OK

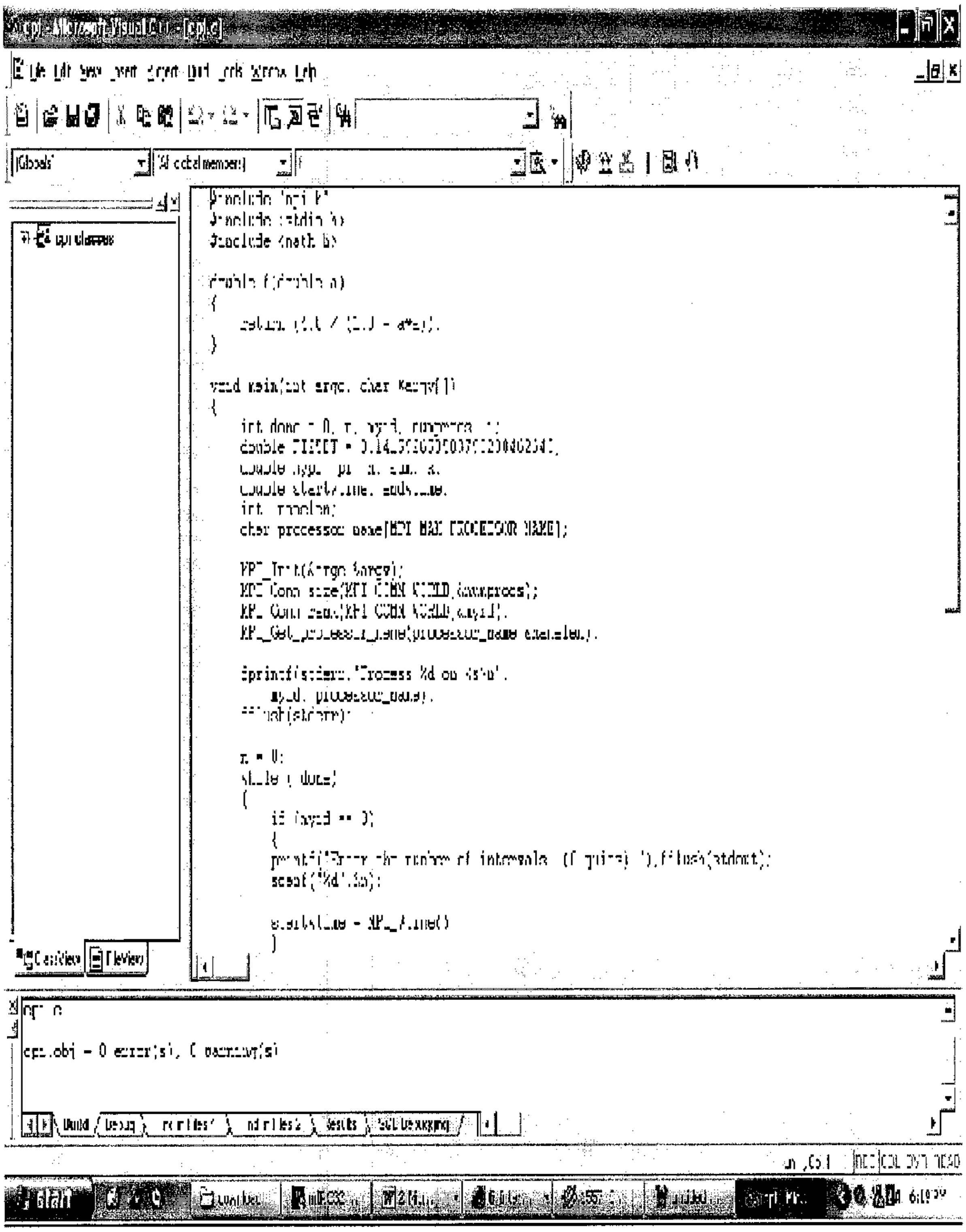
Cancel

Figure 3.3.6.1 The interface for MPI configuration



```
               3 Dir(s)     217,927,680 bytes free

C:\Program Files\MPICH\mpd>cd bin

C:\Program Files\MPICH\mpd\bin>dir
 Volume in drive C has no label.
 Volume Serial Number is 44D0-D7FF

 Directory of C:\Program Files\MPICH\mpd\bin

09/09/2004  03:49 AM    <DIR>          .
09/09/2004  03:49 AM    <DIR>          ..
06/08/2003  04:37 PM           311,296 MPIConfig.exe
06/08/2003  04:38 PM            81,920 MPIRegister.exe
06/08/2003  04:38 PM           143,360 MPIRun.exe
06/08/2003  04:38 PM           389,120 guiMPIRun.exe
06/08/2003  04:37 PM           319,488 MPDUpdate.exe
06/08/2003  04:39 PM            94,208 mpijob.exe
06/08/2003  04:39 PM           249,856 guiMPIJob.exe
06/08/2003  04:37 PM           192,512 mpd.exe
10/09/2004  03:07 AM                 0 mpi4FC.tmp
               9 File(s)      1,781,760 bytes
               2 Dir(s)     217,927,680 bytes free

C:\Program Files\MPICH\mpd\bin>mpirun.exe c\temp\cpi.exe
```

Figure 3.3.6.2 The interface for executing the parallel and sequential programming



```
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f(double a)
{
    return (4.0 / (1.0 + a*a));
}

void main(int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime, endwtime;
    int  namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    fprintf(stderr,"Process %d on %s\n",
        myid, processor_name);
    fflush(stderr);

    n = 0;
    while (!done)
    {
        if (myid == 0)
        {
        printf("Enter the number of intervals: (0 quits) ");fflush(stdout);
        scanf("%d",&n);

        startwtime = MPI_Wtime();
        }
```

```
cpi.obj - 0 error(s), 0 warning(s)
```

Figure 3.3.6.3 The interface for programming platform

Below are the coding for parallel programming using c command n visual c++ compiler. The library used are MPI library

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f(double a)
{
    return (4.0 / (1.0 + a*a));
}

void main(int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime, endwtime;
    int  namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    fprintf(stderr,"Process %d on %s\n",

                                                    myid,
processor_name);
                                                    fflush(
stderr);

    n = 0;
    while (!done)
    {
        if (myid == 0)
        {

printf("Enter the number of intervals: (0 quits)
");fflush(stdout);
scanf("%d",&n);


startwtime = MPI_Wtime();
        }
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
        if (n == 0)
            done = 1;
        else
        {
            h   = 1.0 / (double) n;
            sum = 0.0;
            for (i = myid + 1; i <= n; i += numprocs)
            {
                x = h * ((double)i - 0.5);
                sum += f(x);
            }
            mypi = h * sum;


            MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM,
0, MPI_COMM_WORLD);


            if (myid == 0)
                                                       {
                printf("pi is approximately %.16f, Error
is %.16f\n",

pi, fabs(pi - PI25DT));

                                                   endwtim
e = MPI_Wtime();

                                                   printf(
"wall clock time = %f\n", endwtime-startwtime);
                                                       }
        }
    }
    MPI_Finalize();
}
```

Appendix 6

How MPI works?

Message-Passing Routines

Basic send and receive routines

Send and receive message-passing calls often have the form

```
Send (parameter_list)
Recv (parameter_list)
```

Where send () is placed in the source process originating the message, and recv () is placed in the destination process to collect the message being sent. The actual parameters will depend the software and in some cases can be complex.The simplest set of parameters would be the destination ID and message in send () and the source ID and the name of the location for receiving message in recv ().For the C language, we might have the call

```
Send  ( &X,destination_id);
```

In the source process, and the call

```
Recv (&y,source_id)
```

In the destination process, to send the data x in the source process to y in the destination process. The order of parameters depends upon the system. We will show the process identification after the data and use an & with a single data element, as the specification usually calls for a pointer here. In this example, x must have been preloaded with the data to be sent and x and y must be the same type and size.Often, we want to send more complex messages then simply one data element, and then a more powerful message formation is needed. Various mechanisms are provided for send/receive routines for efficient code and flexibility.

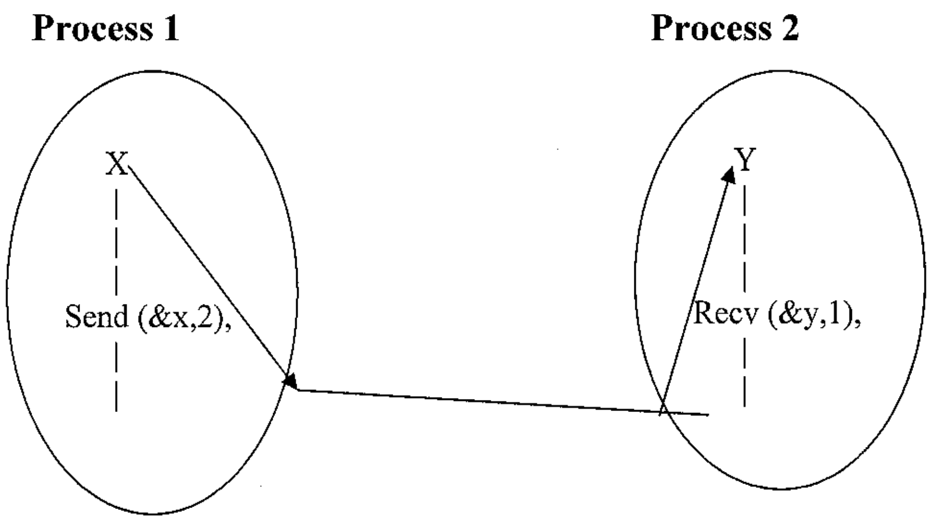


Figure 4.2.4.1 Passing a message between processes using send () and recv () library calls

Sychronous message passing

The term synchronous is used for routines that return when the message transfer has been completed. A synchronous send routine will wait until the complete message that it has sent has been accepted by the receiving process before returning. A synchronous receive routine will wait the message it is expecting arrives and the message is stored before returning. A pair of processes, one with a synchronous send operation and one with a matching synchronous recieveoperation, will be synchronized, with neither the source process nor the destination process able to proceed until the message has been passed from the source process to the destination process.Hence, synchronous routines intrinsically perform two actions: they transfer data, and they synchronize processes.The term rendezvous is used to describe the meeting and synchronization of two processes through synchronous send/receive operations.

Synchronous send and receive operations do not need message buffer storage.They suggest some form of signaling, such as three way protocol in which the source first sends a "request to send" message to the destination.When the destination is ready to accept the message, it returns an acknowledgement. Upon receiving this acknowledgement, the source sends the actual messages. Synchronous message-passing is shown in figure 4.2.4.2 using the 3 –way protocol. In figure 4.2.4.2 (a), process 1 reaches its send() before process 2 reaches its recv().At that time, process 2 must awaken process 1 with some form of "signal", and then both can participate in the message transfer. Note that in figure 4.2.4.2 (a), the message is kept in the source process until it can be sent.In figure 4.2.4.2 (b), process 2 reaches its recv () before process 1 has reached its send ().Now, process 2 must be suspended until both can participate in the message transfer. The exact mechanism for suspending and awakening processes is system dependent.

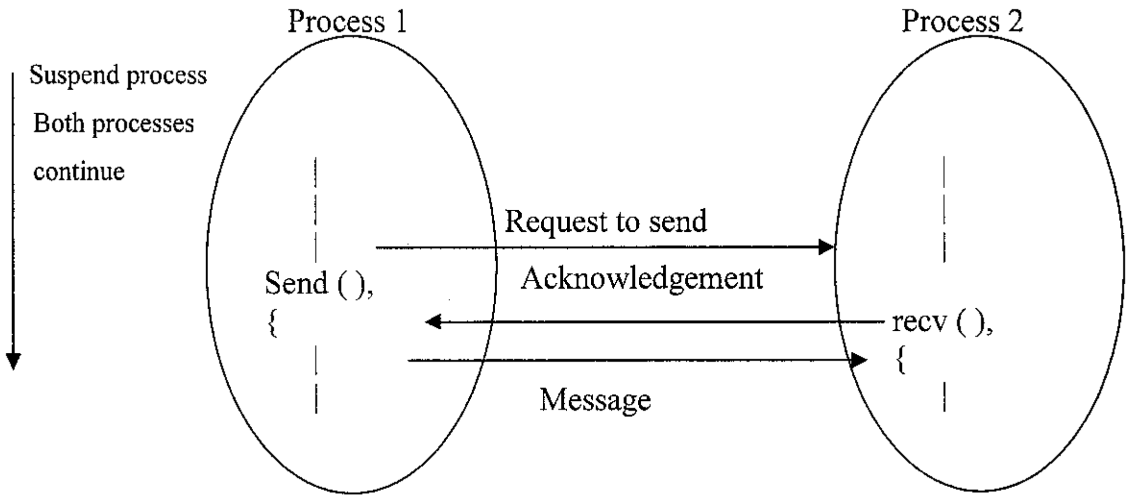


Figure 4.2.4.2 (a) When send () occur Before recv ()

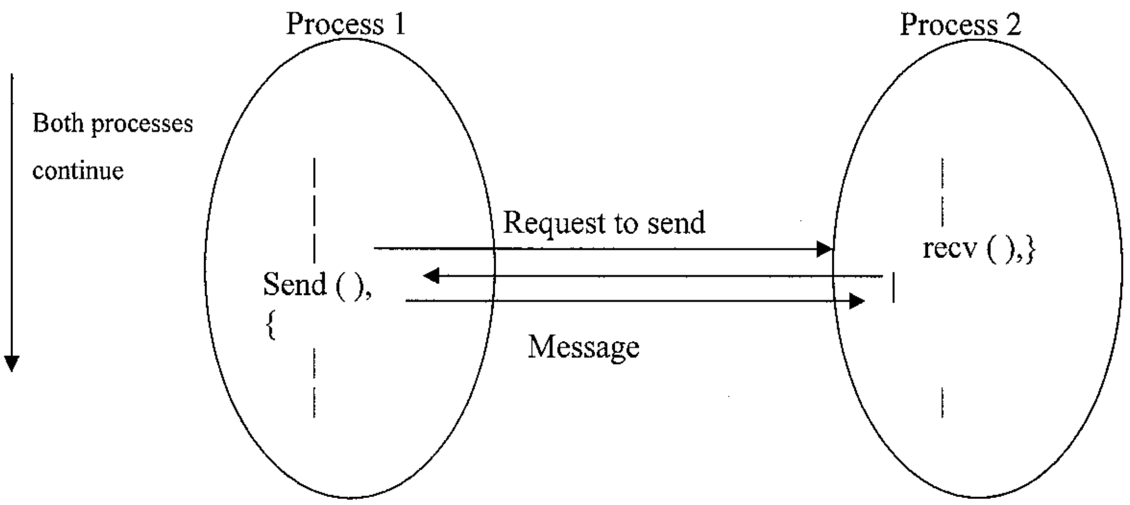


Figure 4.2.4.2 (b) When recv () occur Before send ()

4.2.4.3 Message Selection

To provide greater flexibility, messages can be selected by a message tag attached to the message.The message tag is typically a user-chosen positive integer (including zero) that can be used to differentiate between different type of messages being sent.Then specific receive routines can be made to accept only messages with a specific message tag and ignore other messages. Amessage tag will be an additional parameter in send () and recv (), usually immediately following the source / destination identification. For example, to send a message, x, with message tag 5 from a source process, 1, to a destination process, 2, and assign to y, we might have

Send (&x, 2 , 5)

In the source process and

recv (&y, 1 , 5) in the destination process. The message tag is carried within the message.If special type matching is not required, a wild card can be used in place of a message tag, so that the recv () will match with any send ().

4.2.4.4 Broadcast

There are usually many other message-passing and related routines that provide desirable features. A process is frequently required to send the same message more than one destination process. The term broadcast is used to describe sending the same message to all the processes concerned with the problem.

Broadcast is illustrated in figure 4.2.4.4.The processes that are to participate in the broadcast must be identified, typically by first forming a named group of processes to be used as a parameter in the broadcast routines. In Figure 4.2.4.4, process 0 is identified as the root process within the broadcast parameters.The root process could be any process in the group. In this example, the root process holds the data to be broadcast in buf. Figure 4.2.4.4 shows each process executing the same broadcast routine, which is very convenient for SPMD model, in which all processes have the same program.Figure 4.2.4.4 also shows the root receiving the data, which the arrangement used in MPI but it depends upon the message passing system.

Process 0 Process 1 Process

p-1

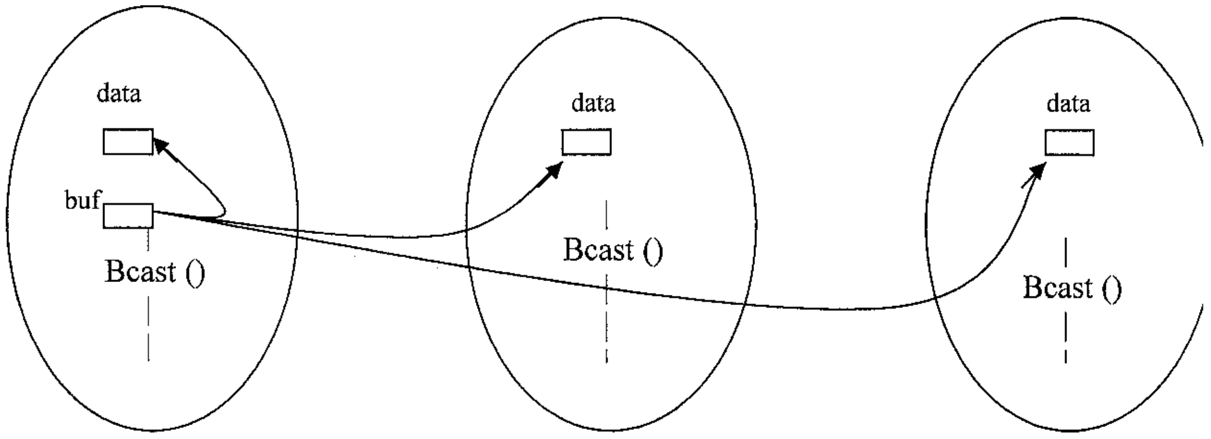


Figure 4.2.4.4 Broadcast Operation

Appendix 8

```
#include <stdio.h>
#include <math.h>
#ifndef HUGE_VAL
#define HUGE_VAL 10.0e38
#endif


#include "mpi.h"
#include "mpptest.h"
#include "getopts.h"
int __NUMNODES, __MYPROCID;

#ifdef HAVE_STDLIB_H
#include <stdlib.h>
#endif

#ifndef DEFAULT_AVG
#define DEFAULT_AVG 50
#endif

#include <string.h>

/* Forward declarations */
void PrintHelp( char *[] );

/*
   This is a simple program to test the communications performance of
   a parallel machine.
 */

/* If doinfo is 0, don't write out the various text lines */
static int    doinfo = 1;

/* Scaling of time and rate */
static double TimeScale = 1.0;
static double RateScale = 1.0;

/* The maximum of the MPI_Wtick values for all processes */
static double gwtick;

/* This is the number of times to run a test, taking as time the minimum
```

```
   achieve timing.
   (NOT CURRENTLY IMPLEMENTED)
   This uses an adaptive approach that also stops when
   minThreshTest values are within a few percent of the current minimum

n_avg - number of iterations used to average the time for a test
n_rep - number of repitition of a test, used to sample test average
to avoid transient effects
*/
static int    minreps       = 30;
/* n_stable is the number of tests that must not (significantly, see
   repsThresh) change the results before mpptest will decide that no
   further tests are required
*/
static int    n_stable;
static double repsThresh    = 0.05;

/* n_smooth is the number of passes over the data that will be taken to
   smooth out any anomolies, defined as times that deviate significantly from
   a linear progression
 */
static int    n_smooth      = 5;
char   protocol_name[256];

/*
   We would also like to adaptively modify the number of repetitions to
   meet a time estimate (later, we'd like to meet a statistical estimate).

   One relatively easy way to do this is to use a linear estimate (either
   extrapolation or interpolation) based on 2 other computations.
   That is, if the goal time is T and the measured tuples (time,reps,len)
   are, the formula for the local time is s + r n, where

   r = (time2/reps2 - time1/reps1) / (len2 - len1)
   s = time1/reps1 - r * len1

   Then the appropriate number of repititions to use is

   Tgoal / (s + r * len) = reps
 */
static double Tgoal = 1.0;
/* If less than Tgoalmin is spent, increase the number of tests to average */
static double TgoalMin = 0.5;
static int autoavg = 0;

/* This structure allows a collection of arbitray sizes to be specified */
```

```
#define MAX_SIZE_LIST 256
static int sizelist[MAX_SIZE_LIST];
static int nsizes = 0;

/* We wish to control the TOTAL amount of time that the test takes.
  We could do this with gettimeofday or clock or something, but fortunately
  the MPI timer is an elapsed timer */
static double max_run_time = 15.0*60.0;
static double start_time = 0.0;

/* All test data is contained in an array of values. Because we may
  adaptively choose the message lengths, provision is made to maintain the
  list elements in an array, and for many processing tasks (output, smoothing)
  only the list version is used. */

/* These are used to contain results for a single test */
typedef struct _TwinResults {
   double t,            /* min of the observations (per loop) */

max_time,       /* max of the observations (per loop) */
        sum_time;       /* sum of all of the observations */
   int   len;           /* length of the message for this test */
   int   ntests;        /* number of observations */
   int   n_avg;         /* number of times to run a test to get average


time */
   int   new_min_found;  /* true if a new minimum was found */
   int   n_loop;         /* number of times the timing loop was


run and accepted */
   struct _TwinResults *next, *prev;
   } TwinResults;

TwinResults *AllocResultsArray( int );
void FreeResults( TwinResults * );
void SetResultsForStrided( int first, int last, int incr, TwinResults *twin );
void SetResultsForList( int sizelist[], int nsizes, TwinResults *twin );
void SetRepsForList( TwinResults *, int );
int RunTest( TwinResults *, double (*)(int,int,void *), void *, double );
int RunTestList( TwinResults *, double (*)(int,int,void*), void* );
int SmoothList( TwinResults *, double (*)(int,int,void *), void * );
int RefineTestList( TwinResults *, double (*)(int,int,void *),void *,

int, double );
```

```
void OutputTestList( TwinResults *, void *, int, int, int );
double LinearTimeEst( TwinResults *, double );
double LinearTimeEstBase( TwinResults *, TwinResults *, TwinResults*, double );
TwinResults *InsertElm( TwinResults *, TwinResults * );

/* Initialize the results array of a given list of data */

/* This structure is used to provice information for the automatic
  message-length routines */
typedef struct {
   double (*f)( int, int, void * );
   int    reps, proc1, proc2;
   void *msgctx;
   /* Here is where we should put "recent" timing data used to estimate
     the values of reps */
   double t1, t2;
   int    len1, len2;
   } TwinTest;

int main( int argc, char *argv[] )
{
   int    dist;
   double (* BasicCommTest)( int, int, void * ) = 0;
   void *MsgCtx = 0; /* This is the context of the


message-passing operation */
   void *outctx;
   void (*ChangeDist)( int, PairData ) = 0;
   int  n_avg, proc1, proc2, distance_flag, distance;
   int  first,last,incr, svals[3];
   int      autosize = 0, autodx;
   double   autorel;
   double   wtick;
   char     units[32];        /* Name of units of length */

   MPI_Init( &argc, &argv );
   MPI_Comm_size( MPI_COMM_WORLD, &__NUMNODES );
   MPI_Comm_rank( MPI_COMM_WORLD, &__MYPROCID );

   /* Get the maximum clock grain */
   wtick = MPI_Wtick();
   MPI_Allreduce(     &wtick,     &gwtick,     1,     MPI_DOUBLE,     MPI_MAX,
MPI_COMM_WORLD );

   /* Set the default test name and labels */
   strcpy( protocol_name, "blocking" );
   strcpy( units, "(bytes)" );

   if (SYArgHasName( &argc, argv, 1, "-help" )) {
                                                                              if
(__MYPROCID == 0) PrintHelp( argv );
                                                                              MPI_F
inalize();
                                                                              return
0;
   }

   if (__NUMNODES < 2 && !SYArgHasName( &argc, argv, 0, "-memcpy" )) {
                                                                              fprintf(
stderr, "Must run mpptest with at least 2 nodes\n" );
                                                                              MPI_F
inalize();
                                                                              return
1;
   }

/* Get the output context */
   outctx = SetupGraph( &argc, argv );
   if (SYArgHasName( &argc, argv, 1, "-noinfo" ))   doinfo    = 0;

   /* Proc1 *must* be 0 because of the way other data is collected */
   proc1         = 0;
   proc2         = __NUMNODES-1;
   distance_flag = 0;
   if (SYArgHasName( &argc, argv, 0, "-logscale" )) {
                                                                              svals[0
]    = sizeof(int);
                                                                              svals[1
]    = 131072;   /* 128k */
                                                                              svals[2
]    = 32;
   }
   else {
                                                                              svals[0
]    = 0;
                                                                              svals[1
]    = 1024;
                                                                              svals[2
]    = 32;
   }
   if (SYArgHasName( &argc, argv, 1, "-distance" ))  distance_flag++;
   SYArgGetIntVec( &argc, argv, 1, "-size", 3, svals );
   nsizes = SYArgGetIntList( &argc, argv, 1, "-sizelist", MAX_SIZE_LIST,

sizelist );

   if (SYArgHasName( &argc, argv, 1, "-logscale" )) {
                                                                                                   /*  Use
the sizelist field to specify a collection of power of
                                                                                                     two
sizes.  This is a temporary hack until we have something

better.  You can use the -size argument to set min and max values
                                                                                                     (the
stride is ignored) */
                                                                                                   int k;
                                                                                                   nsizes
= 0;
                                                                                                   if
(svals[0] == 0) {

sizelist[nsizes++] = 0;
                                                                                                     k   =
4;
                                                                                                   }
                                                                                                   else {
                                                                                                     k   =
svals[0];
                                                                                                   }
                                                                                                   while(
k <= svals[1] && nsizes < MAX_SIZE_LIST ) {

sizelist[nsizes++] = k;
                                                                                                     k  *=
2;
                                                                                                   }
                                                                                                   /*
Need to tell graphics package to use log/log scale */
                                                                                                   DataSc
ale( outctx, 1 );
   }

   /* Get the number of tests to average over */
   n_avg         = DEFAULT_AVG;
   if (SYArgHasName( &argc, argv, 1, "-autoavg" )) {
      autoavg = 1;
```

```
                                                          n_avg
= 5;  /* Set a new default.  This can be overridden */
   }
   SYArgGetInt( &argc, argv, 1, "-n_avg", &n_avg ); /* was -reps */

   if (SYArgGetDouble( &argc, argv, 1, "-tgoal", &Tgoal )) {          if
 (TgoalMin > 0.1 * Tgoal) TgoalMin = 0.1 * Tgoal;
   }
   SYArgGetDouble( &argc, argv, 1, "-rthresh", &repsThresh );

   SYArgGetInt( &argc, argv, 1, "-sample_reps", &minreps );
   n_stable = minreps;
   SYArgGetInt( &argc, argv, 1, "-n_stable", &n_stable );

   SYArgGetDouble( &argc, argv, 1, "-max_run_time", &max_run_time );
   if (SYArgHasName( &argc, argv, 1, "-quick" ) ||                    SYArg
 HasName( &argc, argv, 1, "-fast"  )) {
      /* This is a short cut for
       -autoavg -n_stable 5 */
      autoavg  = 1;
      n_avg    = 5;
      n_stable = 5;
     }

     autosize = SYArgHasName( &argc, argv, 1, "-auto" );
     if (autosize) {                                                autodx
  = 4;                                                              SYArg
  GetInt( &argc, argv, 1, "-autodx", &autodx );                     autorel
  = 0.02;                                                           SYArg
  GetDouble( &argc, argv, 1, "-autorel", &autorel );
      }

   /* Pick the general test based on the presence of an -gop, -overlap, -bisect
     or no arg */
      SetPattern( &argc, argv );
      if (SYArgHasName( &argc, argv, 1, "-gop")) {                  /*    we
   need to fix this cast eventually */                              BasicC
   ommTest = (double (*)(int,int,void*))
```

```
int MsgSize;
char cbuf[32];
if (SYArgHasName( &argc, argv, 1, "-sync" )) {
BasicCommTest = round_trip_b_overlap;
strcpy( protocol_name, "blocking" );
}
else {
/* Assume -async */
BasicCommTest = round_trip_nb_overlap;
strcpy( protocol_name, "nonblocking" );
}
MsgSize = 0;
SYArgGetInt( &argc, argv, 1, "-overlapmsgsize", &MsgSize );
MsgCtx = OverlapInit( proc1, proc2, MsgSize );
/* Compute floating point lengths if requested */
if (SYArgHasName( &argc, argv, 1, "-overlapauto")) {
OverlapSizes( MsgSize >= 0 ? MsgSize : 0, svals, MsgCtx );
}
strcat( protocol_name, "-overlap" );
if (MsgSize >= 0) {
sprintf( cbuf, "-%d bytes", MsgSize );
}
else {
strcpy( cbuf, "-no msgs" );
}
strcat( protocol_name, cbuf );
TimeScale = 0.5;
```

```
RateScale = 2.0;
    }
    else if (SYArgHasName( &argc, argv, 1, "-memcpy" )) {
int use_vector = 0;
MsgCtx    = 0;
ChangeDist = 0;
TimeScale = 1.0;
RateScale = 1.0;
use_vector = SYArgHasName( &argc, argv, 1, "-vector" );
/* memcpy_rate_int, memcpy_rate_double */
if (SYArgHasName( &argc, argv, 1, "-int" )) {
    if (use_vector) {
    }
    else {
BasicCommTest = memcpy_rate_int;
strcpy( protocol_name, "memcpy-int" );
    }
}
else  if (SYArgHasName( &argc, argv, 1, "-double" )) {
    if (use_vector) {
BasicCommTest = memcpy_rate_double_vector;
strcpy( protocol_name, "memcpy-double-vector" );
    }
    else {
```

```
BasicCommTest = memcpy_rate_double;
strcpy( protocol_name, "memcpy-double" );
  }
}
#ifdef HAVE_LONG_LONG
else if (SYArgHasName( &argc, argv, 1, "-longlong" )) {
  if (use_vector) {
BasicCommTest = memcpy_rate_long_long_vector;
strcpy( protocol_name, "memcpy-longlong-vector" );
  }
  else {
BasicCommTest = memcpy_rate_long_long;
strcpy( protocol_name, "memcpy-longlong" );
  }
}
#endif
else {
BasicCommTest = memcpy_rate;
strcpy( protocol_name, "memcpy" );
}
  }
  else {
/* Pair by default */
BasicCommTest = GetPairFunction( &argc, argv, protocol_name );
MsgCtx = PairInit( proc1, proc2 );
```

```
                                                                   Chang
eDist = PairChange;
                                                                   if
(SYArgHasName( &argc, argv, 1, "-debug" ))

PrintPairInfo( MsgCtx );
                                                                   TimeS
cale = 0.5;
                                                                   RateSc
ale = 2.0;
   }
   first = svals[0];
   last  = svals[1];
   incr  = svals[2];
   if (incr == 0) incr = 1;

/*
  Finally, we are ready to run the tests.  We want to report times as
  the times for a single link, and rates as the aggregate rate.
  To do this, we need to know how to scale both the times and the rates.

  Times: scaled by the number of one-way trips measured by the base testing
  code.  This is often 2 trips, or a scaling of 1/2.

  Rates: scaled by the number of simultaneous participants (as well as
  the scaling in times).  Compute the rates based on the updated time,
  then multiply by the number of participants.  Note that, for a single
  sender, time and rate are inversely proportional (that is, if TimeScale
  is 0.5, RateScale is 2.0).

 */

   start_time = MPI_Wtime();

/* If the distance flag is set, we look at a range of distances.  Otherwise,
  we just use the first and last processor */
   if (doinfo && __MYPROCID == 0) {
                                                                   Header
Graph( outctx, protocol_name, (char *)0, units );
   }
   if(distance_flag) {
                                                                   for(dist
ance=1;distance<GetMaxIndex();distance++) {

proc2 = GetNeighbor( 0, distance, 0 );
```

```
                                                                        if
(ChangeDist)

                                                                      (*Chan
geDist)( distance, MsgCtx );

time_function(n_avg,first,last,incr,proc1,proc2,


BasicCommTest,outctx,


autosize,autodx,autorel,MsgCtx);
                                                                      }
  }
  else{
                                                                      time_f
unction(n_avg,first,last,incr,proc1,proc2,BasicCommTest,outctx,

autosize,autodx,autorel,MsgCtx);
  }

/*
  Generate the "end of page".  This allows multiple distance graphs on the
  same plot
 */
  if (doinfo && __MYPROCID == 0)
                                                                      EndPa
geGraph( outctx );
  EndGraph( outctx );

  MPI_Finalize();
  return 0;
}

/*
  This is the basic routine for timing an operation.

  Input Parameters:
. n_avg - Basic number of times to run basic test (see below)
. first,last,incr - length of data is first, first+incr, ... last
      (if last != first + k * incr, then actual last value is the
      value of first + k * incr that is <= last and such that
      first + (k+1) * incr > last, just as you'd expect)
. proc1,proc2 - processors to participate in communication.  Note that
      all processors must call because we use global operations to
```

```
       manage some operations, and we want to avoid using process-subset
       operations (supported in Chameleon) to simplify porting this code
. CommTest  -  Routine to call to run a basic test.  This routine returns
       the time that the test took in seconds.
. outctx -  Pointer to output context
. autosize - If true, the actual sizes are picked automatically.  That is
       instead of using first, first + incr, ... , the routine choses values
       of len such that first <= len <= last and other properties, given
       by autodx and autorel, are satisfied.
. autodx - Parameter for TST1dauto, used to set minimum distance between
       test sizes.  4 (for 4 bytes) is good for small values of last
. autorel - Relative error tolerance used by TST1dauto in determining the
       message sizes used.
. msgctx - Context to pass through to operation routine
 */
void time_function( int n_avg, int first, int last, int incr,
                                                                    int
proc1, int proc2, double (*CommTest)(int,int,void*),

void *outctx, int autosize, int autodx,

double autorel, void *msgctx)
{
  int   distance, myproc;
  int   n_without_change;  /* Number of times through the list without


changes */

  myproc  = __MYPROCID;
  distance = ((proc1)<(proc2)?(proc2)-(proc1):(proc1)-(proc2));

  /* Run test, using either the simple direct test or the automatic length
   test */
  if (autosize) {
                                                                 TwinR
esults *twin;
                                                                 int k;

                                                                 twin  =
AllocResultsArray( 1024 );
                                                                 SetRes
ultsForStrided( first, last, (last-first)/8, twin );

                                                                 /*  Run
tests */
```

```
                                                                SetRep
sForList( twin, n_avg );
                                                                for
(k=0; k<minreps/5; k++) {
                                                                  int
kk;
                                                                  for
(kk=0; kk<5; kk++)

                                                                (void)
RunTestList( twin, CommTest, msgctx );
                                                                  /*
Don't refine on the last iteration */
                                                                  if  (k
!= minreps-1)

RefineTestList( twin, CommTest, msgctx, autodx, autorel );

                                                                }
                                                                for
(k=1; k<n_smooth; k++) {
                                                                  if
(!SmoothList( twin, CommTest, msgctx )) break;

                                                                }
                                                                /*
Final output */
                                                                if
(myproc == 0)

OutputTestList( twin, outctx, proc1, proc2, distance );
                                                                FreeRe
sults(twin);
  }
  else {
                                                                TwinR
esults *twin;
                                                                int k;
                                                                if
(nsizes) {
                                                                  twin
= AllocResultsArray( nsizes );

SetResultsForList( sizelist, nsizes, twin );

                                                                }
                                                                else {

nsizes = 1 + (last - first)/incr;
```

```
                                                                twin
= AllocResultsArray( nsizes );

SetResultsForStrided( first, last, incr, twin );
                                                              }

                                                              /*  Run
tests */
                                                              SetRep
sForList( twin, n_avg );
                                                              n_with
out_change = 0;
                                                              for
(k=1; k<minreps; k++) {
                                                                if
(RunTestList( twin, CommTest, msgctx )) {

n_without_change = 0;
                                                                }
                                                                else

n_without_change++;
                                                                if
(n_without_change > n_stable) {
#if DEBUG_AUTO

                                                              printf(
"Breaking because stable results reached\n" );
#endif

                                                              break;
                                                                }
                                                              }
                                                              for
(k=1; k<n_smooth; k++) {
                                                                if
(!SmoothList( twin, CommTest, msgctx )) break;
                                                              }
                                                              /*
Final output */
                                                              if
(myproc == 0)

OutputTestList( twin, outctx, proc1, proc2, distance );
                                                              FreeRe
sults(twin);
```

```
   }
   if (myproc == 0)
                                                                        DrawG
raph( outctx, 0, 0, 0.0, 0.0 );
}



/***********************************************************************
********
  Utility routines

************************************************************************
*******/

void PrintHelp( char *argv[] )
{
 if (__MYPROCID != 0) return;
 fprintf( stderr, "%s - test individual communication speeds\n", argv[0] );

 fprintf( stderr,
"Test a single communication link by various methods.  The tests are \n\
combinations of\n\
 Protocol: \n\
 -sync       Blocking sends/receives    (default)\n\
 -async      NonBlocking sends/receives\n\
 -ssend      MPI Syncronous send (MPI_Ssend) and MPI_Irecv\n\
 -force      Ready-receiver (with a null message)\n\
 -persistant  Persistant communication\n\
 -put        MPI_Put (only on systems that support it)\n\
 -get        MPI_Get (only on systems that support it)\n\
 -vector     Data is separated by constant stride (only with MPI, using UBs)\n\
 -vectortype  Data is separated by constant stride (only with MPI, using \n\
           MPI_Type_vector)\n\
\n\
 Message data:\n\
 -cachesize n Perform test so that cached data is NOT reused\n\
\n\
 -vstride n   For -vector, set the stride between elements\n\
 Message pattern:\n\
 -roundtrip   Roundtrip messages        (default)\n\
 -head       Head-to-head messages\n\
 -halo       Halo Exchange (multiple head-to-head; limited options)\n\
  \n" );
PrintHaloHelp();
```

```
fprintf( stderr, "\
-memcpy      Memory copy performance (no communication)\n\
-memcpy -int Memory copy using a for-loop with integers\n\
-memcpy -double Memory copy using a for-loop with doubles\n\
-memcpy -longlong Memory copy using a for-loop with long longs\n" );

fprintf( stderr,
"  Message test type:\n\
(if not specified, only communication tests run)\n\
-overlap     Overlap computation with communication (see -size)\n\
-overlapmsgsize nn\n\
           Size of messages to overlap with is nn bytes.\n\
-bisect      Bisection test (all processes participate)\n\
-bisectdist n Distance between processes\n\
 \n" );

fprintf( stderr,
"  Message sizes:\n\
-size start end stride                (default 0 1024 32)\n\
           Messages of length (start + i*stride) for i=0,1,... until\n\
           the length is greater than end.\n\
-sizelist n1,n2,...\n\
           Messages of length n1, n2, etc are used.  This overrides \n\
           -size\n\
-logscale    Messages of length 2**i are used.  The -size argument\n\
           may be used to set the limits.  If -logscale is given,\n\
           the default limits are from sizeof(int) to 128 k.\n\
-auto        Compute message sizes automatically (to create a smooth\n\
           graph.  Use -size values for lower and upper range\n\
-autodx n    Minimum number of bytes between samples when using -auto\n\
-autorel d   Relative error tolerance when using -auto (0.02 by default)\n");

fprintf( stderr, "\n\
Detailed control of tests:\n\
-quick       Short hand for -autoavg -n_stable 5\n\
           this is a good choice for performing a relatively quick and\n\
           accurate assessment of communication performance\n\
-n_avg n     Number of times a test is run; the time is averaged over this\n\
           number of tests (default %d)\n\
-autoavg     Compute the number of times a message is sent automatically\n\
-tgoal  d    Time that each test should take, in seconds.  Use with \n\
           -autoavg\n\
-rthresh d   Fractional threshold used to determine when minimum time\n\
           has been found.  The default is 0.05.\n\
-sample_reps n  Number of times a full test is run in order to find the\n\
           minimum average time.  The default is 30\n\
  -n_stable n  Number of full tests that must not change the minimum \n\
          average value before mpptest will stop testing.  By default,\n\
          the value of -sample_reps is used (i.e.,no early termination)\n\
  -max_run_time n  Maximum number of seconds for all tests.  The default\n\
          is %d\n\
\n", DEFAULT_AVG, (int)max_run_time );

fprintf( stderr, "\n\
  Collective operations may be tested with -gop [ options ]:\n" );
PrintGOPHelp();

PrintGraphHelp();
PrintPatternHelp();
fflush( stderr );
}
```