Volume 3, No. 5, Sept-Oct 2012

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Analysis of TCP/IP Overhead on Overlapping Message Transfer and Computation in a Distributed Memory System Architecture

Mohamed Faidz Mohamed Said*[1], Mohd Nasir Taib[2] and Saadiah Yahya[3]
[1,3]Faculty of Computer & Mathematical Sciences, [2]Faculty of Electrical Engineering
Universiti Teknologi MARA, 40450 Shah Alam, Malaysia
[1]faidzms@ieee.org, [2]dr.nasir@ieee.org,, [3]saadiah@tmsk.uitm.edu.my

Abstract– High Performance Computing (HPC) has been commonly constructed due to the widely implementation of open source software and clustering technology. The growth of clustering technology is also due to the demand of the parallel programming either using shared memory systems (SMS) or distributed memory systems (DMS). The DMS hardware platform utilizing the Message Passing Interface (MPI) programming model is easier to build and scale than the SMS platform because of the direct access to local memory and mainly the communication is via explicit send/receive messages primitives. These message primitives consist of non-blocking and blocking communications. When the programming model of non-blocking communication is used, the messages can return soon without waiting for the finish of communication operation, thus allowing the overlap of message transfer and computation. By empirically measuring the time, rate and capturing the packets, vital information can be extracted from them. The objective of this research is to investigate the TCP/IP protocol statistics of the non-blocking and blocking communications applied on various message and overlap sizes. The benefit of understanding the communication overhead of these distinct MPI communication primitives has the advantage for the programmer to write efficient parallel software. In this research, a four-node PC cluster is built on a private dedicated LAN using the message-passing library MPICH as its parallel software. It is demonstrated conclusively that for a long message size, the large difference in the average Mbit per second for the packets shows that the non-blocking overlap messages provides a more efficient communication compared to the blocking messages, and therefore will eventually contribute to the improved performance of parallel applications.

***Keywords*:** MPICH, cluster computing, overlapping.

I. INTRODUCTION

High Performance Computing (HPC) currently has attracted good researchers and produced groundbreaking results over the last 13 years [1]. This technology has been applied to the commercial world over time with mixed results. The two main factors which pushed this technological momentum are the open source software and cluster computing. The use of open source Linux operating system as well as the cluster of personal computers (PCs) as an environment for HPC has been shown to be technologically promising and economically encouraging. The central initiative behind this HPC is due to the demand of parallel computing. Parallel computing is a fairly well-established field and several programming platforms and standards have evolved around it over the past two decades. The two common hardware platforms used in this parallel programming HPC are the shared memory systems (SMS) and the distributed memory systems (DMS) [2]. On the SMS platform, it makes effective use of data parallelism and can act on entire arrays at once by executing instructions on different indexes of an array in different processors. Consequently, this provides automatic parallelization with minimal effort needed.

This includes the High Performance FORTRAN as a language suited for this type of parallel programming. On the DMS platform, since the memory is distributed, it utilizes the message passing programming model where the communication has direct access to the local memory of a computer node and via the explicit send/receive primitives. Thus the situation is radically different since the message passing model is easier to build and scale compared to that on the SMS platform. The code written only has to be aware of the underlying distributed nature of the hardware and uses explicit primitives to exchange messages between different nodes. The two popular standards for writing parallel programs for PC clusters are the Message Passing Interface (MPI) and the Parallel Virtual Machines (PVM). Lately, parallel programming using the MPI has become the de facto standard for building parallel applications on PC clusters [3].

MPI programming model is easier to build and scale in the DMS platform than that in the SMS platform because of the direct access to local memory and the communication is via explicit primitives. Basically, these explicit primitives are the send and receive and their variant messages. The variants of these send and receive messages include the blocking and non-blocking communications [2]. The blocking communication is where it makes the send/receive request and waits until the reply is returned before it subsequently continues accordingly. Whereas, the non-blocking communication is where it makes the send/receive request and subsequently continues accordingly without waiting for a reply. There are distinct flows of communications between these two primitives. The programs which utilize these primitives will have dissimilar effects eventually. Presently, there are lacks of research in this field of programming approach. In terms of software development, by understanding the effects of these approaches, programmer will have the benefits of writing efficient parallel application software.

Historically the goal of achieving performance through the exploitation of parallelism is as old as electronic digital computing itself which emerged from the World War II era. Many different approaches have been devised with many commercial or experimental versions being implemented over the years [4]. Parallel computing architectures may be codified in terms of the coupling and the typical latencies involved in performing parallel operations [5, 6]. The eight major architecture classes are systolic computers [7], vector computers [8], single instruction multiple data (SIMD) architecture [9], dataflow models [10], processor-in-memory (PIM) architecture, massively parallel processors (MPPs) [11], distributed computing [12] and lastly commodity clusters [13, 14]. Commodity clusters may be subdivided into four classes and they are Superclusters, Cluster farms, Workstation clusters and Beowulf clusters. Beowulf clusters incorporate mass-market PC technology and employ commercially available networks such as Ethernet for local area networks. Thus, these characteristics are entirely unlike in a traditional parallel computer where it is built of highly specialized hardware and the architecture is custom built.

Beowulf computing is currently one of the parallel computing architectures that has been used extensively either in the teaching, industrial and commercial sectors. This class of computing is formed by a collection of more than one computer that are linked via a network. The success of this computing architecture is in general due to the exploitation of its physical commodity components that are easily available in the market. On top of that, the software employed by this type of computing are open codes that can be freely downloaded from the public domain. The term Beowulf cluster refers to a set of regular personal computers (PC) commonly interconnected through an Ethernet. It operates as a parallel computer but differs from other parallel computers in the sense that it consists of mass-produced commodity off-the-shelf (COTS) hardware. Usually, a parallel computer is built of highly specialized hardware and the architecture is chosen depending on the needs.

This makes it optimal for solving certain problems. However, it also makes it very expensive and since it often is more or less custom built, technical support is exclusive. By constructing a Beowulf cluster, these issues are solved. The penalty of going with a Beowulf cluster is in reduced communication capacity between the processors, since an Ethernet is much slower than a custom-built interconnect hardwired to a motherboard [11]. Recently, a rapid increase in the use of this type of clusters can be observed and this is due to mainly two reasons. Firstly, the magnitude of the PC market has allowed PC prices to decrease while sustaining dramatic performance increase. Secondly, the Linux community [15-27] has produced a vast asset of free software for these kinds of applications. Beowulf clusters emphasize no custom components, no dedicated processors, a private system area network and a freely available software base. Cluster computing involves the use of a network of computing resources to provide a comparatively economical package with capabilities once reserved for supercomputers. One of the initial work in developing a Beowulf cluster is carried out by Andersson [15] at the Department of Scientific Computing, Uppsala University, Sweden. On the architectural perspective, the Beowulf cluster can be divided into two types of variants. The first is the rack-mounted system and the second is the bladed system. Firstly, the rack-mounted system is a collection of individual system units placed together and this study uses this type of implementation. An example of this rack-mounted system is shown by Fig. 2 where it demonstrates a typical home-built Beowulf cluster [28].

Figure. 1. A 52-node Beowulf cluster [28]

One of the earliest prevalent clusters of this type is built by Andersson [15]. With the aids from the system technicians from National Supercomputer Center, Linkoping University, he develops a Beowulf cluster called Grendel which is built from 17 standard PC computers. Every computer consists of commodity off-the-shelf products and they are connected together with a fast Ethernet network. The other 16 PCs have exactly the same configuration, both hardware and software. The PCs have their own hard drives and each node runs its own operating system and accesses a common file area through the front-end PC. All of the installed software is free and public and the operating system used for all computers is RedHat Linux. The cluster system is then tested with several benchmarks, namely the LMbench 2.0 Benchmark, the Stream Benchmark and the NAS Parallel Benchmark 2.3 (NPB).

Secondly, the bladed system is a collection of individual motherboards put together within the close vicinity, like in computer laboratory. An example of this bladed system is demonstrated by Fig. 1 where it exhibits a 52-node Beowulf cluster [28] used by the McGill University pulsar group to search for pulsations from binary pulsars.

Figure. 2. A home-built Beowulf cluster [28]

An example of this type of implementation is done by Feng [29]. He presents a novel Beowulf cluster named Bladed Beowulf which is originally proposed as a cost-effective alternative to the traditional Beowulf clusters. In his later work, Feng [30] also introduces this Bladed Beowulf and its performance metrics.

Generally, there are many reviews on the preliminary works and discussions in many aspects of the cluster variants. These reviews and discussions include [31], [32], Underwood [33], Kuo-Chan [34], Yi-Hsing [35] and Farrell [36]. Uthayopas discusses the issues in building powerful scalable cluster [37] and also proposes system management for the Beowulf cluster [38]. Finally, Stafford [39] discusses the legacy and the future of Beowulf cluster with its founder, Donald Becker.

Recent years have shown an immense increase in the use of Beowulf clusters [15, 40]. Their role in providing multiplicity of data paths, increased access to storage elements both in memory and disk and scalable performance is reflected in the wide variety of applications of parallel computing, such as Slezak [41], Yu-Kwong [42] and Chi-Ho [43]. The research works cover both the two memory architecture, namely the shared memory and the message passing. Most of the researches on the later memory architecture are based on the MPICH, a software written by Gropp and Lusk from the Argonne National Laboratory, University of Chicago [44]. The comparison between the message-passing and shared address space parallelism is presented by Shan [45]. For the benchmark segment, two microbenchmarks that analyze network latency that more realistically represents the way that MPI is typically used is presented by Underwood [46].

For comparing the communication types, the work is done by Coti [47] who presents scalability comparisons between MPI blocking and non-blocking check-pointing approaches and Grove [48] who presents tools to measure and model the performance of message-passing communication and application programs. He also presents a new benchmark that uses timing mechanism to measure the performance of a single MPI communication routine. For the communication-computation overlapping, several works have been conducted by researchers. They are Danalis [49] who presents approaches for improving communication-computation overlap in MPI collective operations, Sancho [50] who proposes a method to evaluate performance improvement for overlapping communication and computation and Sohn [51] who examines communication overlapping capability to improve performance of distributed memory machines. From the numerous reviews made, most of them deal with the issues of the computer communication techniques, computational complexity of scheduling and operating system. Most of these works also focus on the communication latency and load among the networked machines.

The network latency, the delay caused by communication between processors and memory modules over the network has been identified as a major source of degraded parallel computing performance. However, these researches have not ventured into the role and effect of the programming primitives used in the application software itself. In the overlapping issue, the analysis on the effect of data size and its TCP/IP protocols should provide useful information concerning the efficient use of parallel programming codes within the clusters of PC.

This research gap requires detailed analysis by using a new method. Therefore, this research project empirically attempts to look into this effect and how this overlap issue characterizes the operation of task, other than the completion time. The characterizations are based on different message sizes: short and long. The work would focus on the analysis on the TCP overhead and its rate.

The scope of this research is a collection of four computers that are connected to a switch via a network. Each computer is installed with Linux operating system and MPICH parallel software. Effects of the blocking, non-blocking and overlapping communication are measured by a program in C language which provides information of the time and rate based on the pertinent routines. The TCP/IP protocols are captured by a packet sniffer application that is able to monitor the network usages and extract vital information from them.

This paper is organized as follows. Section II gives the background theory by explaining in details the processes on the Beowulf parallel computing. Section III provides the methodology used in the experiment. Section IV provides the analyses and discussions on the communicational measurement while Section V presents the analyses on the TCP/IP measurements. Finally, Section VI presents the conclusion of the findings of this research.

II. THEORY

In message-passing programming on Beowulf computing, a programmer employs message-passing library in order to produce a desired application. This user-level library operates on two principal mechanisms.

The first is the method to create separate process for execution on different computer. Based on the Multiple Program Multiple Data (MPMD) model, there are separate programs for each processor. One processor executes master process while the other processes started from within master process, as depicted in Fig. 3.

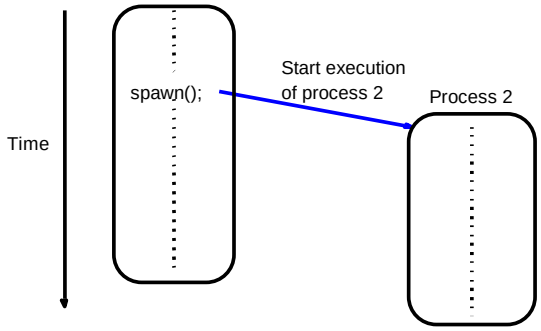

Figure. 3. Multiple Program Multiple Data (MPMD) model

The second is the method to send and receive messages. Basically, for the point-to-point send and receive primitives, passing a message between processes is performed using send() and recv() library calls as shown in Fig. 4.

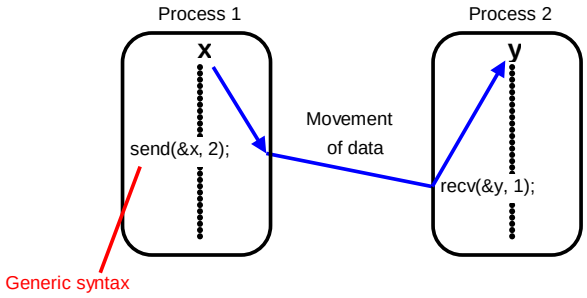

Figure. 4. Basic send and receive primitives

For the synchronous message passing, the routines actually return when message transfer completed. For the send routine, it waits until the complete message can be accepted by the receiving process before sending the message. While for the receive routine, it waits until the message it is expecting arrives. Synchronous routines intrinsically perform two actions: they transfer data and they synchronize processes. This is called blocking communication. The examples of the MPI blocking primitives are *MPI_Send()* and *MPI_Recv()*. The blocking primitives formats are *MPI_Send (buf, count, datatype, dest, tag, comm, request)* and *MPI_Recv (buf, count, datatype, src, tag, comm, request).*

However, for the asynchronous message passing, the routines do not wait for actions to complete before returning and it usually requires local storage for messages. In general, they do not synchronize processes but allow processes to move forward sooner. Thus, in this type of communication, the message-passing routines return before message transfer completed. Message buffer is needed between the source and the destination to hold message. This is called non-blocking communication and demonstrated in Fig. 5.

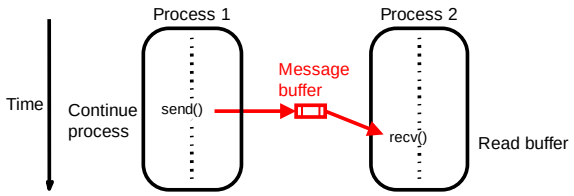

Figure. 5. Message-passing routines return before message transfer completed

The examples of the MPI non-blocking primitives are *MPI_Isend()* and *MPI_Irecv()*. For *MPI_Isend()*, the send will return immediately even before source location is safe to be altered. Meanwhile, for *MPI_Irecv()*, the receive will return even if no message to accept. The '*I*' in '*Isend*' and '*Irecv*' means Immediate. The primitives formats are *MPI_Isend (buf, count, datatype, dest, tag, comm, request)* and *MPI_Irecv(buf, count, datatype, src, tag, comm, request).*

In addition to these primitives, the overlapping of message transfer and computation is another technique in the Beowulf computer programming. While the message is being transmitted, a computer may also permit local computation to be done. For the purpose of comparison, there are two possible situations. The first one is without overlap and the second one is with overlap. These occurrences are shown in the following diagram in Fig. 6.

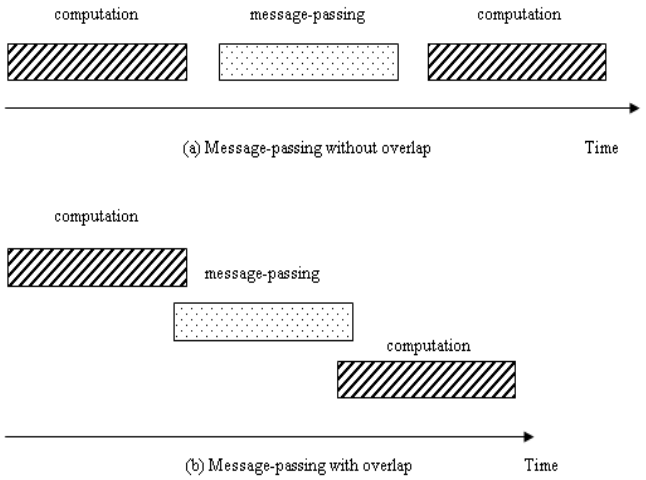

Figure. 6. Abstraction of computation and message-passing events (a) without overlap (b) with overlap

If part of the computation can not be done alongside with the message-passing, a computer only can compute and then do message-passing at different time, as depicted in case (a) of Fig. 6. However, if overlap is permitted, this will allow optimal use of completion time, as illustrated in case (b) of Fig. 6. As modern technology allows the later case, the direct benefit would be the overall reduced completion time. However, apart from the completion time, there are protocols effects associated which include the TCP/IP overhead and its rate. The overhead can be looked from the percentage of data segment, remote shell segment and the peer-to-peer short message of the TCP frames. The rate can be examined from the average packets per second, average bytes per second and average Mbit per second. These effects can be empirically measured as different message sizes and overlap sizes are applied.

III. METHODOLOGY

In order to accomplish this research, a sequence of development phases are performed (Fig. 7). It is crucial to organize the phases systematically as it is vital in ensuring a well-planned process completion.

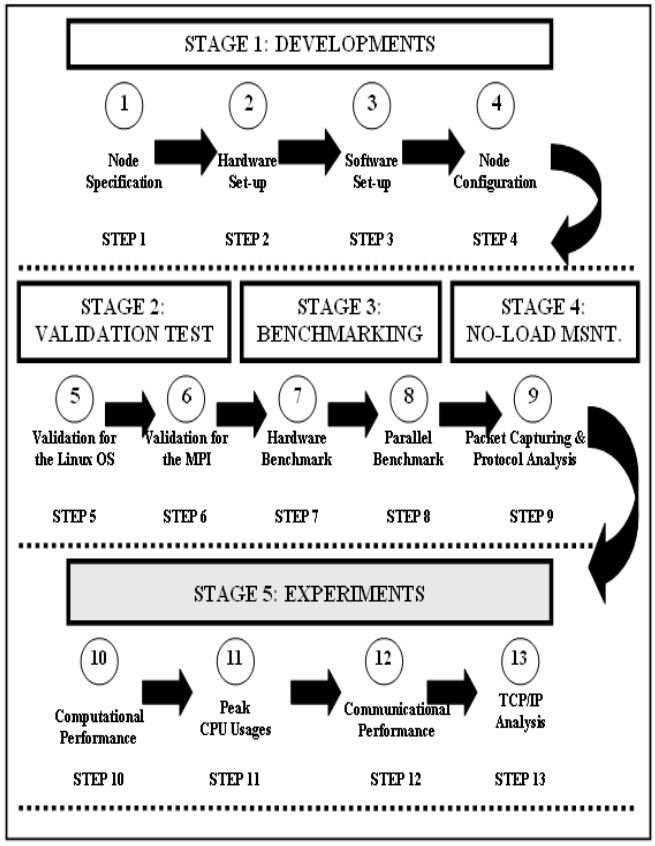

Figure. 7. Methodology for Beowulf cluster developments and experiments

In the early phase of doing this research, it practically starts with the developments stage (Stage 1), where there are four work phases of the developments work. These work phases are the node specification, the hardware set-up, the software set-up and the node configuration of the Beowulf cluster system. The work is arranged in this sequence to ensure that the proper hardware construction is created before setting up the software on top of it. The initial step in this node specification phase is specifying the master and the slave. In order to create a proper naming and numbering convention, the cluster system is conceptually divided into two main components, the master component and the slave component. The convention will have a name node together with a two-digit number. The master node is given a codename of *node00*. The two-digit number 00 is chosen to demonstrate the function of the master node as the front-end PC. Meanwhile the first slave node is given a name and number starting with *node01*. Thus the second node of this cluster system is *node02* and the subsequent third node is *node03*.

The last consideration is the network interconnection. Due to the use of a network switch, the link topology being applied will be the star organization. Step 2 demonstrates the second work phase in the developments stage, namely the hardware set-up. This hardware installation phase covers the assembly work and the connections of the nodes through a network interconnect. All the nodes being used are complete standalone systems with monitors, hard drives, keyboards and their related peripherals. Basically the node is comprised of a CPU with a cache, a main memory, a personal computer interconnection (PCI) and a network interface card (NIC). This cluster system is conceptually a combination of four nodes namely individual PCs with a network interconnect device located at the centre of the arrangement. The general structure of this cluster system is presented in Fig. 8.

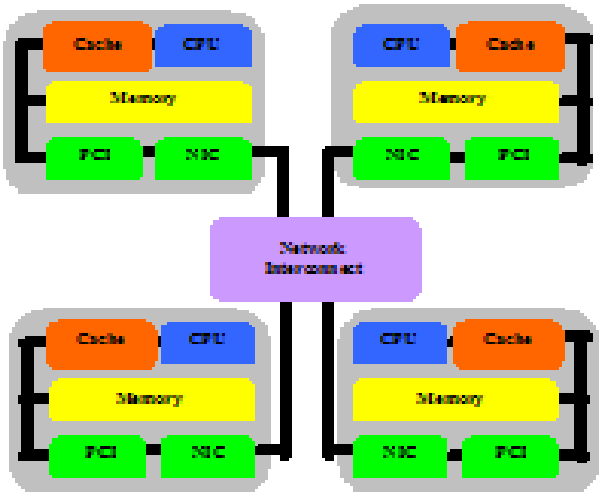

Figure. 8. Hardware set-up

Specifically, the physical units of the cluster system are of heterogeneous characteristics. The entire four nodes are connected through their respective RJ45 ports to a switch using unshielded twisted pair (UTP) cables. A full view of this cluster set-up is illustrated in Fig. 9.

Step 3 illustrates the third work phase in the developments stage; the software set-up. The software installation phase is generally divided into three components. The first software component is the *RedHat 9.0* OS. After the successful installation of the OS, the next component is the *MPICH 1.2.0* library. This software installation phase begins after the nodes are completely assembled physically.

Step 4 demonstrates the fourth work phase in the developments section; namely the node configuration (Fig. 10). The node configuration phase for the OS part consists of several tasks.

Figure. 9. A full view of the cluster set-up

These tasks involve the creation and modification of important system files to ensure that the system is fully functional. The MPI library also has specific essential files that have to be correctly set to run parallel program.

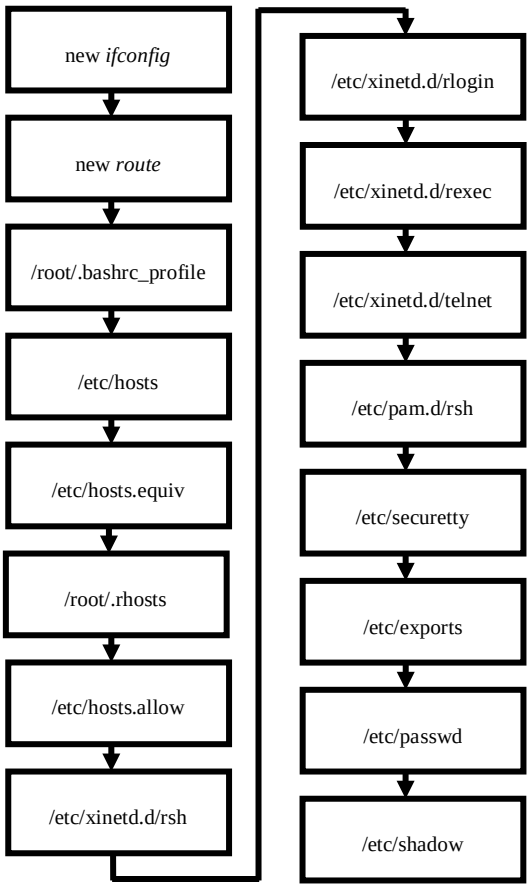

Figure. 10. Flowchart of node configuration

Lastly, the proper environment for the experimental data collection should be appropriately established to ensure that the data collection is consistent. This includes the correct command execution, file and directory location. The proper IP address and aliasing of all the nodes are primarily established in the */etc/hosts* file. Each node in the cluster has a similar hosts file with appropriate changes to the first line reflecting the hostname of that node itself. Thus, slave *node01* would have a first line of the text *192.168.0.9 node01* with the third line containing the IP and hostname of *node00*. All other nodes are configured in the same manner with the *127.0.0.1* localhost line is not removed. This file is edited on every cluster node by adding the names and IP addresses of every node in the cluster. This allows these machines to be accessed by name instead of by IP number. In general, the system files that need to be created and modified are illustrated by the flowchart as shown in Fig. 10.

For the validation test (Stage 2), there are two types of tests applied to ensure that the whole system is properly working and functioning. The first is the validation for the Linux OS installation and the second is the validation for the MPICH installation. In order to validate the successful setup of the Linux OS into each of the nodes, several attribute elements of the node system can be verified. These attribute elements will prove that the OS is correctly set-up and functioning. The elements consist of the configuration verification, the routing table verification, the data transfer verification, the re-verification of the overall performance and slave validation. These system elements are verified and confirmed on the master node as well as on all of the remote slave nodes. For the validation of the MPI installation (Step 6), *Hello World* program is applied. In this program, the MPI specifies the library calls to be used in a C program. The MPI program contains one call to *MPI_Init* and one call to *MPI_Finalize*. Therefore all other MPI routines must be called after *MPI_Init* and before *MPI_Finalize*. Furthermore the C program must also include the file *mpi.h* statement at the beginning of the program.

A. Benchmarking:

In this benchmarking phase (Stage 3), it describes the chosen benchmarking being used in the Beowulf cluster system. For the reliability testing, the performance of the developed cluster is tested using the authoritative benchmarks. There are two kinds of benchmark programs; the hardware benchmarks and the parallel benchmarks. For comparison purposes, the Grendel cluster system (G-cluster) is chosen [15]. The hardware benchmarks used is the LMbench 2.0 benchmark while the parallel benchmark applied is the NAS Parallel Benchmark 2.3 (NPB 2.3). LMbench is a set of small benchmarks used to measure performance of computer components which are vital for efficient system performance. The aim of these benchmark tests is to provide the real application figures that can be achieved by normal applications. The main performance bottlenecks of current systems are latency, bandwidth or a combination of these two. LMbench tests focus on the system's ability to transfer data between processor, cache, memory, disk and network. However, these tests do not measure the graphics throughput, computational speed or any multiprocessor features of a computer node. Since LMbench is highly portable, it should run as is with *gcc* as default compiler. This LMbench benchmark tests six different aspects of the system. These are the processor and processes, the context switching, the communication latency, the file and virtual memory system latencies, the communication bandwidths and the memory latencies.

Firstly, the results of LMbench 2.0 benchmark for the processor and processes are displayed below (Table 1). The times shown are in microseconds (µs). In the nineth test (Table 1), for creating a process through *fork+exec*, the exec proc measures the time it takes to create a new process and have that process perform a new task. The time taken to exec proc for this cluster is 344.0 µs compared to 706.2 µs. Lastly, in the tenth test, for creating a process through *fork+/bin/sh –c*, the shell proc measures the time it takes to create a new process and have the new process running a program by asking the shell to find that program and run it. The time taken to shell proc for this cluster is 2247 µs compared to 3605.3 µs. Generally, the comparison results from the LMbench tests for the processor and processes show that this Beowulf cluster produces significantly better performance of a small-scale cluster.

Table 1. LMbench 2.0 benchmark for the processor and processes – smaller is better

		This cluster	G-cluster
1	null call	0.45	0.27
2	null I/O	0.51	0.38
3	stat	1.78	3.72
4	open/close	2.38	4.63
5	select	5.937	26.3
6	signal install	0.79	0.77
7	signal handle/catch	2.59	0.95
8	fork proc	99.0	110.1
9	exec proc	344.0	706.2
10	shell proc	2247	3605.3

Secondly, the results of LMbench 2.0 benchmark for the communication latencies are exhibited below (Table 2). The times shown are in microseconds (µs). In the fifth test (Table 2), for the interprocess communication latency via TCP/IP, TCP measures the time it takes to send a token back and forth between a client/server. No work is done in the processes. The time taken for the TCP of this cluster is 14.1 µs compared to 16.4 µs.

Table 2. LMbench 2.0 benchmark for the local communication latencies (µs) – smaller is better

		This cluster	G-cluster
1	pipe	4.808	4.021
2	AF UNIX	9.46	8.34
3	UDP	11.9	11.5
4	RPC/UDP	21.4	26.4
5	TCP	14.1	16.4
6	RPC/TCP	25.9	39.1

Thirdly, the results of LMbench 2.0 benchmark for the local communication bandwidths are displayed below (Table 3). The measurements shown are in Mbytes per second (MB/s). In the third test, for reading and summing of a file, file reread measures how fast data is read when reading a file in 64KB blocks. Each block is summed up as a series of 4 byte integers in an unrolled loop. The benchmark is intended to be used on a file that is in memory. The bandwidth for the file reread of this cluster is 1149.3 MB/s compared to 332.9 MB/s. Generally, the comparison results from the LMbench tests for the local communication bandwidths show that this Beowulf cluster produces a better performance in the whole local communication bandwidths category tests conducted.

Table 3. LMbench 2.0 benchmark those are for the local communication bandwidths (MB/s) – bigger is better

		This cluster	G-cluster
1	pipe	1181	790.7
2	AF UNIX	2033	516.3
3	file reread	1149.3	332.9
4	Mmap reread	1164.3	462.0
5	Bcopy (libc)	369.8	300.6
6	Bcopy (hand)	387.9	264.1
7	mem read	1522	481.7
8	mem write	522.4	361.6

Finally, the results of LMbench 2.0 benchmark for the memory latencies are presented below (Table 4). The measurements shown are in nanosecond (ns).

Table 4. LMbench 2.0 benchmark for the memory latencies

		This cluster	G-cluster
1	L1 cache	0.836	2.279
2	L2 cache	7.7070	19.0
3	Main memory	118.5	151.0

For the memory read latencies, L1 cache, L2 cache and Main memory measure the time it takes to read memory with varying memory sizes and strides respectively. The entire memory hierarchy is measured onboard and external caches, main memory and TLB miss latency. It does not measure the instruction cache. Generally, the comparison results from the LMbench tests for the memory latencies show that this Beowulf cluster produces a better performance for a cluster since smaller is better.

For the parallel benchmarks, NAS Parallel Benchmark 2.3 (NPB) is applied. G-cluster supports two implementations of the message passing interface (MPI); LAM and MPICH. The results of this cluster show those of class A while the results of G-cluster are those of class B. These comparisons are shown in the following Table 5. The unit used is million instructions per second (mop/s).

Table 5. NAS Parallel Benchmark 2.3 output (mop/s) - bigger is better

Benchmark code	This cluster	G-cluster
MG	7.03	1.31
FT	5.45	5.62 (4 procs)
LU	392.17	163.16
SP	235.84	360.99 (4 procs)
BT	187.52	no result for 1 and 4 procs

The MG test uses a multigrid method to compute the solution of the three-dimensional scalar Poisson equation. It partitions the grid by successively dividing it in two, starting with the z dimension, then the y and x dimensions, until all processors are assigned. In this first test, the rate for this cluster is 7.03 mop/s compared to 1.31 mop/s. The FT test contains the computational kernel of a three-dimensional FFT-based spectral method. It performs 1-D FFTs in the x and y dimensions on a distributed 3-D array, which is done entirely within each processor, and then continues with an array transposition which requires an all-to-all communication. The final FFT is then performed. In this second test, the rate for this cluster is 5.45 mop/s compared to 5.62 mop/s. The LU test simulates a CFD application which uses successive over-relaxation (SSOR) to solve a block lower-block upper triangular system of equations, derived from an unfactored implicit finite-difference discretization of the Navier-Stokes equations in three dimensions. In this third test, the rate for this cluster is 392.17 mop/s compared to 163.16 mop/s.

The SP test simulates a CFD application that solves uncoupled systems of equations resulting from an implicit finite-difference discretization of the Navier-Stokes equations. It solves scalar pentadiagonal systems for a full diagonilization of the above scheme. In this fourth test, the rate for this cluster is 235.84 mop/s compared to 360.99 mop/s. The BT test originates from the same problem as SP, but instead of solving scalar systems, it solves block-triangular systems of 5 x 5 blocks. In this fifth test, the rate for this

cluster is 187.52 mop/s. However, comparison can not be made since there are no results obtained for the 1 and 4 processors for the G-cluster. In general, the NAS parallel benchmark (NPB) tests conducted are used to measure the overall system performance. They are divided into two groups depending on the utilization of CPU, memory and network: kernel benchmarks and application benchmarks. The kernel benchmarks are intended to put pressure on the Linux kernel with its implementation of the TCP/IP stack, while the application benchmarks concentrate more on CPU and memory utilization. In a nutshell, the comparison results for the two NPB tests show that this Beowulf cluster produces a better performance.

Next is the methodology for Stage 4 - the no-loading measurement. The purpose of obtaining the measurements of this cluster system during the period of no loads is to investigate the existence of packets. By exploring this period, the packets involved with its protocol details can be studied. These protocols existing during this period will provide the absolute comparison to that during the period of an application being run. In the packet capturing phase, the sniffing program named *Ethereal* is installed inside the master node. The data obtained are the elapsed time in seconds, the time between the first and the last packet in seconds, the packet count, the average packets per second, the bytes of traffic, the average bytes per second and the average megabit per second. These values measured of one run period are compared accordingly to that of another run period.

B. Algorithm Design:

In order to make the required measurement program, an algorithm is firstly designed. The program is coded using C language because of its suitable attribute and more flexible than the others. The program is tested to ensure it is correct and modifications will be done from time to time if needed. It starts with the program initialization and specifying the program parameters. To start a program, *MPI_Init()* is used before calling any MPI function. All processes are enrolled in a universe called *MPI_Comm_World*. Each process is given a unique rank number from *0* up to *p-1* for *p* processes. To terminate a program, *MPI_Finalize()* is used. To measure the execution time between two points in the code, *MPI_Wtime()* routines are used together with the appropriate variables. Thus, initially, the program may call the *MPI_Init* and later call *MPI_Comm_rank()* and *MPI_Comm_size()*. The body of the measurement program runs the test and the times are recorded. The rates and times are inversely proportional. To record the total amount of time that the test takes, the *MPI_Wtime()* function is used since the MPI timer is an elapsed timer:
start_time = MPI_Wtime(); run_time = MPI_Wtime() - start_time. The time function is a function of several other routines of the first data length (*first*), the last data length (*last*), process 1 (*proc1*), process 2 (*proc2*), the communication test (*commtest*) and the context of the message-passing operation (*msgctx*): *time_function (first,last,incr,proc1,proc2, commtest, msgctx)*; In the main program, the communication test (*commtest*) is identified as double data type, and it is a function of the protocol used. This *commtest* is a function of *&argc*, *argv* and *protocol_name* where the protocol name is of char data type and the options are blocking, nonblocking and overlap. The context of the message-passing operation (*msgctx*) is a function of *proc1* and *proc2*. Finally, the program ends with *MPI_Finalize();* and *return 0*. The *mpptest* of the Linux performance test is chosen because it has the merit of having the same purpose of the required measurement. The flowchart of the measurement program algorithm is shown by Fig. 11.

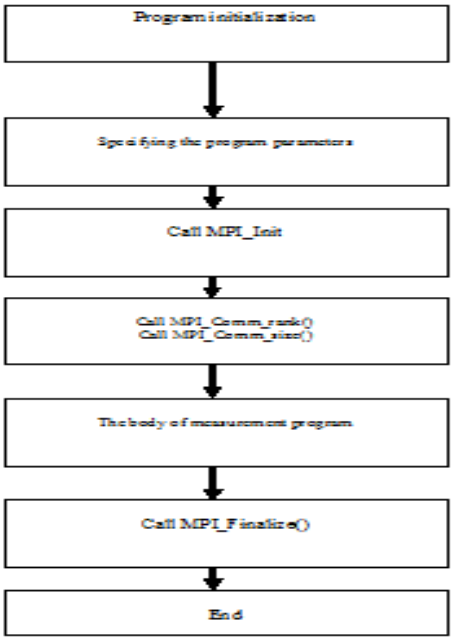

Figure. 11. Measurement Algorithm

All the experiments conducted for the blocking and non-blocking operations are performed based on various message sizes either at lower level or near saturation level to compare the effect of different packet sizes. The flowchart of the measurement methodology is displayed in Fig. 12.

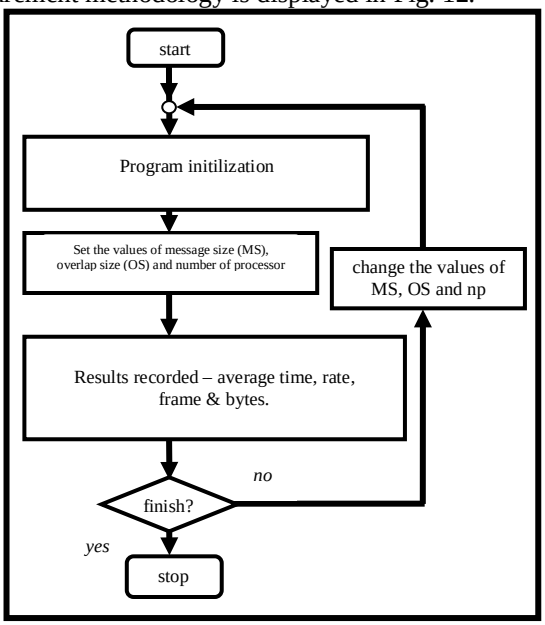

Figure. 12. Measurement methodology

IV. RESULTS AND DISCUSSIONS

In this section, it shows the results for the experiments on the communicational performance (Step 12). The comparisons are made from the perspective of the rate (bandwidth) as the message sizes are changed. Fig. 13 provides the results of the non-blocking operations on the Beowulf cluster based on the different message sizes and number of processors.

The lowest line is the measurement for np=2, the middle line is the measurement for np=3 and the highest line is the measurement for np=4. By adding more processors, the rate of non-blocking communication for each np generally increases up to a certain saturation level. The saturation levels are different for each np. Therefore, all non-blocking operations with different np show almost the same characteristics of gradually rising and becoming stable during saturation level.

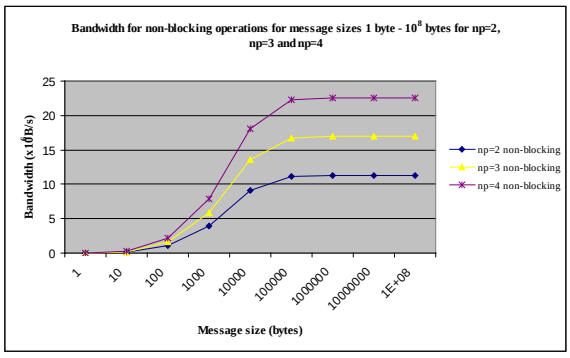

Figure. 13. Rate for the non-blocking operations for np=2, np=3 and np=4

Similarly, Fig. 14 provides the results of the blocking operations on the Beowulf cluster based on the different message sizes and number of processors. The lowest line is the measurement for np=2, the middle line is the measurement for np=3 and the highest line is the measurement for np=4. Similarly, by adding more processors, the rate of blocking communication for each np generally increases up to a certain saturation level. The saturation levels are different for each np. Therefore, all blocking operations with different np show nearly the same characteristics of gradually rising and becoming stable during saturation level.

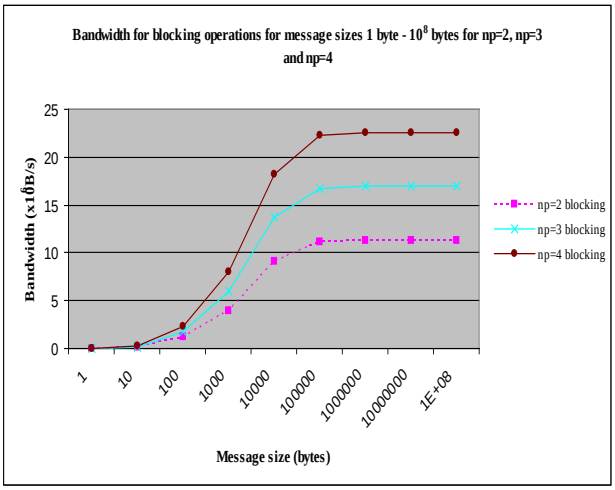

Figure. 14. Bandwidth for the blocking operations for np=2, np=3 and np=4

Subsequently, Fig. 15 summarizes both results on the non-blocking and blocking operations for np=2, np=3 and np=4 in one graph. Generally, both operations show almost the same rate of message passing between different sizes and number of processors. The rate differences between these operations are very minimal as per each np. This should indicate that the use of the non-blocking or blocking routines in this cluster computing has very little effects on the overall performance in terms of the rate of message transmission. Either routine could be applied without having to reconsider the overall impact on the running application.

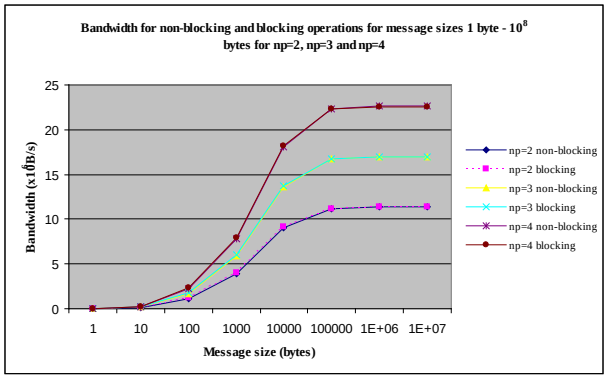

Figure. 15. Rate comparison between non-blocking and blocking operations for np=2, np=3 and np=4

A. Overlapping Results:

In this subsection, it exhibits the results of the average round-trip time on the overlapping experiments (Step 12). Initially, the experiments are performed using smaller message sizes, ranging from 0 bytes up to 1024 bytes with the 2n stride. The result of the average round-trip time for the non-blocking overlap communication is displayed in Fig. 16.

Next, the experiments are similarly performed for the blocking communication using smaller message sizes, ranging from 0 bytes up to 1024 bytes with the 2n stride. The comparison on these different communications from the perspective of the average round-trip time is made. The average round-trip time comparison for different sizes is shown in Fig. 16. Generally, both operations show almost the same time for different sizes.

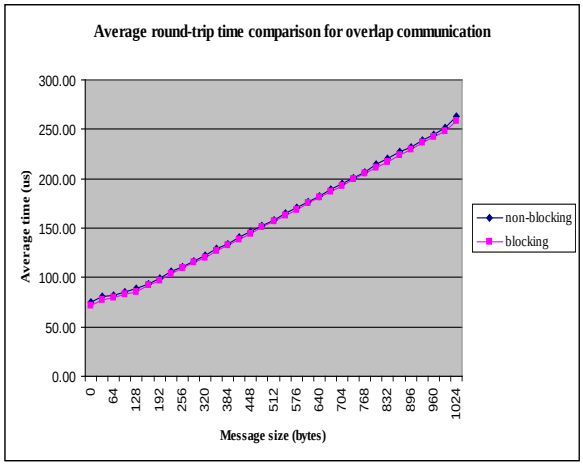

Figure. 16. Average round-trip time comparison

As well, the comparisons on these different communications are displayed from the perspective of the round-trip rate. Fig. 17 shows the round-trip rate comparison for different sizes. Both operations show almost the same rate for different sizes. The slightly lower line corresponds to the non-blocking routine measurement, while the slightly higher line corresponds to the blocking routine measurement. By adding message sizes, the rate of both communication routines generally increases up to a certain saturation level. The saturation levels are also nearly at the same point. Therefore, both routines show the same characteristics of gradually rising and becoming stable during saturation level.

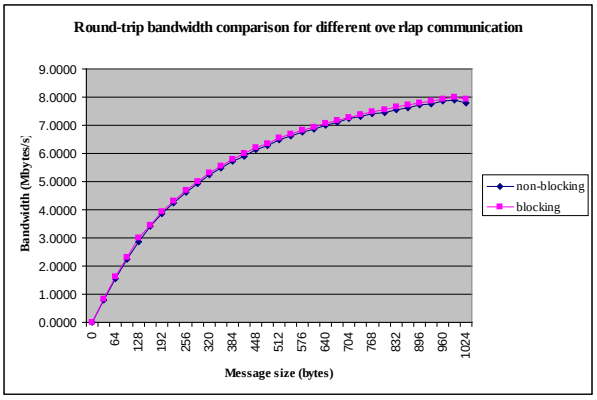

Figure. 17. Rate comparison

The comparison of the rates for the overlap communication is also made. The overlap sizes are changed from one byte to 1,000,000 bytes. Fig. 18 demonstrates the results for the 1,000,000 bytes overlap size.

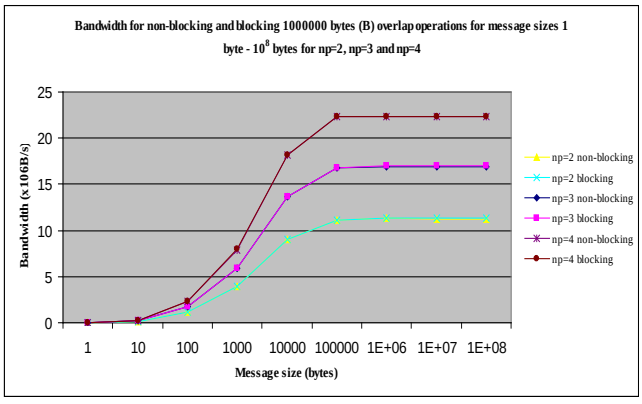

Figure. 18. Rates for non-blocking and blocking overlap operations for np=2, np=3 and np=4

The graph shows that the rate increases gradually until it nearly reaches its saturation level. The comparison of rates exhibited are generally significant upon reaching the 1,000,000 bytes level of the message size. The levels below the 100,000 bytes message sizes appear to show insignificant levels of comparisons in the range of less than 10 MB per second. It is likely that the levels below the 100,000 bytes message sizes are generally less responsive to the change of message sizes.

The results presented generally show a significant increase of the rates from lower np to higher np as well as from smaller message size to larger message. These results may be explained by considering the increase of the message size itself provide an increasingly higher packet transfer among the nodes. Also, the rate is generally proportional to the number of processors. Thus, both overlap routines of different np show the same characteristics of gradually rising and becoming stable during saturation level.

V. TCP/IP ANALYSIS

In this section, it presents the experiment results on the TCP/IP measurement for the non-blocking communication routine. This routine is chosen because based on the previous experiments, both routines show very little differences in rate and average time taken. This section is divided into two subsections; firstly the results on short messages and secondly the results on long messages.

A. Results on Short Messages:

For the short messages, the results are broken into two different overlap sizes; no overlap (0 byte) and small overlap (two bytes). Firstly, the following results are the experiment conducted on the two bytes message size, non-blocking and no overlap. The protocol statistics shows that the frames collected are comprised of 100% transmission control protocol (TCP). The data segment and the remote shell segment occupy 73.41% and 0.14% respectively while the peer-to-peer short message constitutes 24.21% of these TCP frames. The results also show 2,650.53 average packets per second (avg. packets/sec), 293,327.10 the average bytes per second (avg. bytes/sec) and 2.35 average Mbit per second (avg. Mbit/sec). Secondly, the following results are the experiment conducted on the overlap size of two bytes. The frames collected are comprised of 99.97% internet protocol (IP) and 0.03% address resolution protocol (ARP). For the IP frames, TCP represents 99.97%. The data segment occupies 72.87%, the remote shell segment occupies 0.14% while the peer-to-peer short message constitutes 24.05% of these TCP frames. The results also show 658.81 avg. packets/sec, 72,824.47 avg. bytes/sec and 0.58 avg. Mbit/sec.

The TCP data segment and the TCP overhead are analyzed overall. For the short messages, the comparison on TCP data segment on two bytes message size and blocking is illustrated in Fig. 19.

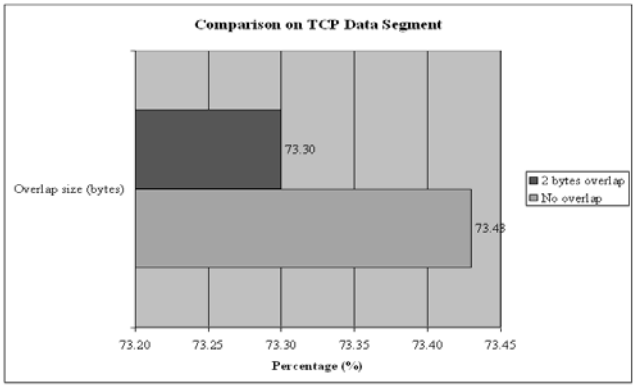

Figure. 19. Comparison on TCP data segment on two bytes message size and blocking

The TCP Data segment percentage for two bytes overlap (73.30%) decreases 0.18% from that for no overlap (73.48%). This result suggests that increasing overlap size will reduce the portion of data transfer in the cluster system.

The comparisons on the avg. packets/sec, avg. bytes/sec and avg. Mbit/sec are demonstrated in Fig. 20, Fig. 21 and Fig. 22 respectively.

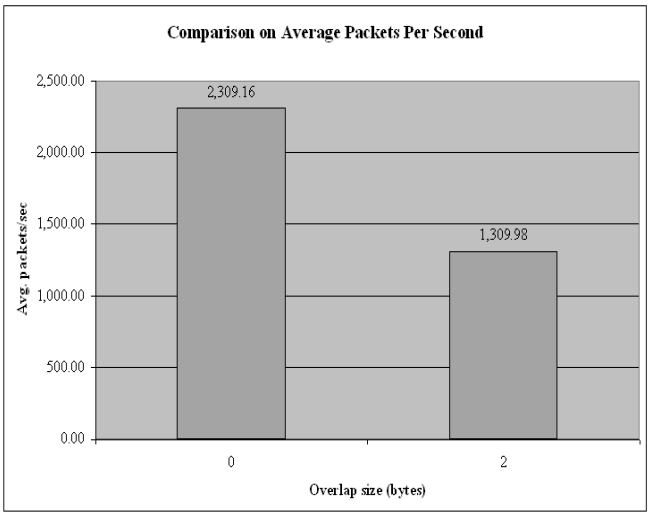

Figure. 20. Comparison on average packets per second on two bytes message size and blocking

Firstly, the average packets per second for two bytes overlap (1,309.29) drops 999.18 from that for the no overlap (2,309.16). This result suggests that increasing overlap size will reduce the rate of packet transfer in the cluster system.

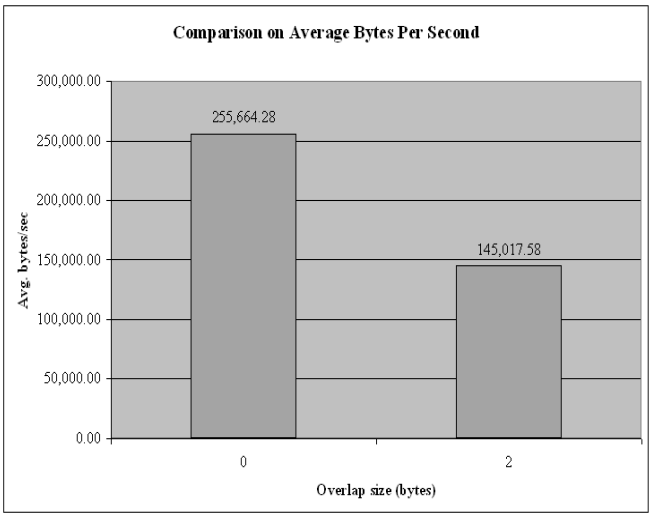

Figure. 21. Comparison on average bytes per second on two bytes message size and blocking

Secondly, the average bytes per second for two bytes overlap (255,664.28) drops 110,646.70 from that for the no overlap (145,017.58). This result suggests that increasing overlap size will reduce the rate of data transfer in the cluster system.

Thirdly, the average Mbits per second for two bytes overlap (2.05) drops 0.89 from that for the no overlap (1.16). TCP data segment percentage for two bytes overlap (73.30%) decreases 0.18% from that for no overlap (73.48%). These results indicate that increasing overlap size on four nodes, two bytes message size and blocking operation will decrease the message transfer rate in the cluster system.

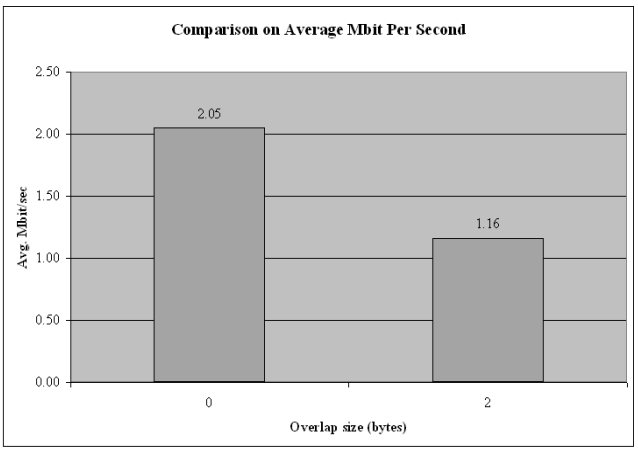

Figure. 22. Comparison on average Mbit per second on two bytes message size and blocking

Conclusively, for a small message size, the overlap causes a lower percentage of the TCP data segment compared to that of the non-overlap. Likewise, the message rate is also smaller than that of the non-overlap.

B. Results on Long Messages:

In this second subsection, the results on the long messages are demonstrated. The results on this long message segments are broken into four different overlap sizes; no overlap (0 byte), small overlap (2 bytes), medium overlap (1024 bytes) and long overlap (8192 bytes). Firstly, the following results are the experiment conducted on the non-blocking and no overlap. The frames collected are comprised of 100% TCP. The data segment occupies 64.70%, the remote shell segment occupies 0.06% while the peer-to-peer short message constitutes 5.32% of these TCP frames. The results also show 1,265.89 avg. packets/sec, 1,106,782.52 avg. bytes/sec and 8.85 avg. Mbit/sec.

Repetitively, a series of different overlap sizes are conducted from two, 1024 to 8192 bytes. For 8192 bytes, the frames collected are comprised of 100% TCP. The data segment occupies 65.08%, the remote shell segment occupies 0.05% while the peer-to-peer short message constitutes 5.32% of these TCP frames. The results also show 5,756.19 avg. packets/sec, 5,066,982.85 avg. bytes/sec and 40.54 avg. Mbit/sec.

Again repetitively, a series of different overlap sizes are conducted on the blocking communication. The comparison on TCP data segment for two, 1024 and 8192 bytes message size is illustrated in Fig. 23. The TCP data segment percentage for two bytes (64.70%) increases 8.16% to that for 1024 bytes (72.86%). Likewise, the TCP data segment percentage for 1024 bytes increases 0.57% to that for 8192 bytes (73.43%). This result suggests that for the blocking communication, increasing message size will reduce the portion of data transfer in the cluster system.

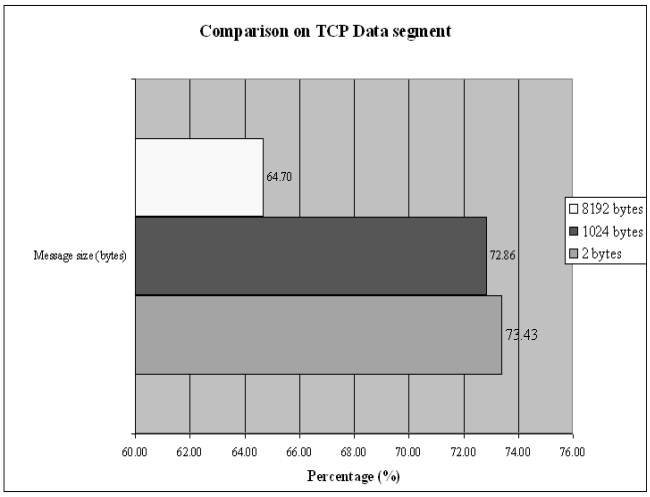

Figure. 23. The comparison on TCP Data segment on blocking, no overlap for two, 1024 and 8192 bytes message sizes

The comparisons on average packets per second, average bytes per second and average Mbit per second are demonstrated in Fig. 24, Fig. 25 and Fig. 26 respectively. The average packets per second for 8192 bytes size (1,132,335.03) raise drops 876,670.75 from that for the two bytes size (255,664.28) but drops 511,435.88 from that for the 1024 bytes size (1,388,106.63). The same phenomena can be seen on the average bytes per second as well as on average Mbit per second. However, different effect is observed for the TCP data segment percentage for three message sizes. These results indicate that the message transfer rate in the cluster system will be increased for the very large size compared to that for small increment.

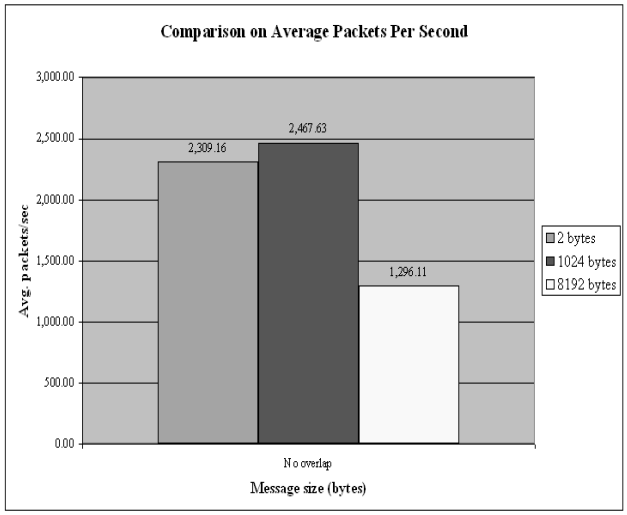

Figure. 24. Comparison on average packets per second on blocking, no overlap for two, 1024 and 8192 bytes message sizes

Generally, for the non-overlap experiment, the TCP rate percentage will drop as the message size increases. However, similar effect is not seen for the small size region. The results for the small message size suggest that the load of a message has little effect on the message rates; however the rate would be significantly affected for the bigger message sizes.

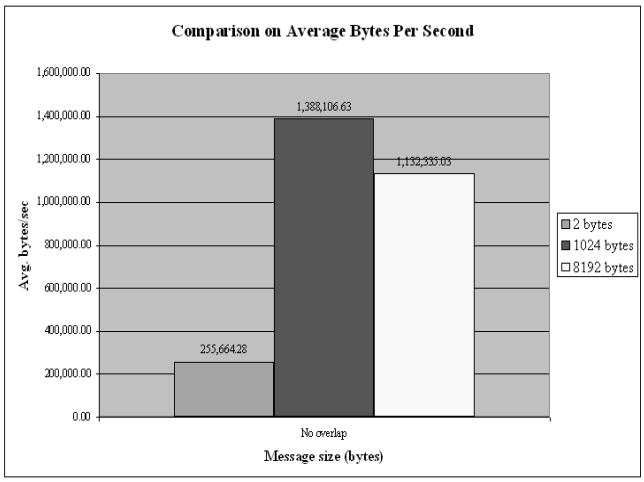

Figure. 25. Comparison on average bytes per second on blocking, no overlap for two, 1024 and 8192 bytes message sizes

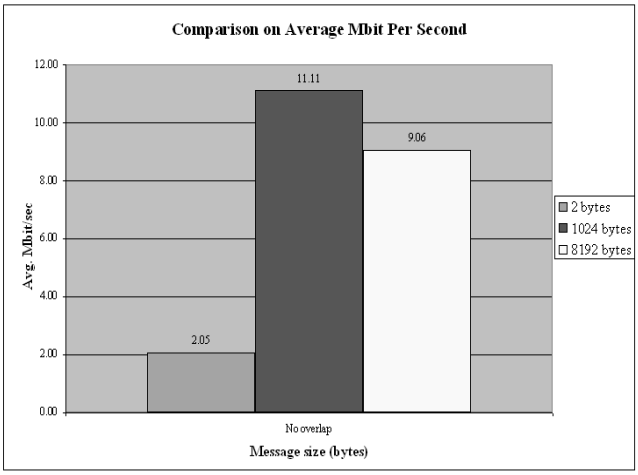

Figure. 26. Comparison on average Mbit per second on blocking, no overlap for two, 1024 and 8192 bytes message sizes

As a summary, Fig. 27 shows the overall TCP/IP measurement comparison between the two message sizes and with the addition of middle sizes of 1024 bytes. The sizes conceptually represent the MPI program loads. Only the blocking MPI routines are considered here since there are very small differences between the two different MPI routines during the previous analysis performed. As in Fig. 27, as the message sizes are changed from short to long, the percentage of the TCP data segment generally increases. It indicates that the TCP/IP overhead gets higher as the size is added. For a small message size, the two-byte overlap illustrates a lower percentage of the TCP data segment compared to that of the non overlap. For the two-byte overlap, the message rate is also smaller than that of the non overlap. This is indicated by the results on the average packets per second, average bytes per second and average Mbit per second. The same phenomena can also be observed for the large message sizes.

Thus, looking from this overlap issue, the general characteristics of the parallel operation on the Beowulf cluster demonstrates that as the message transfer is overlapped with computation, the TCP/IP overhead of the packet decreases.

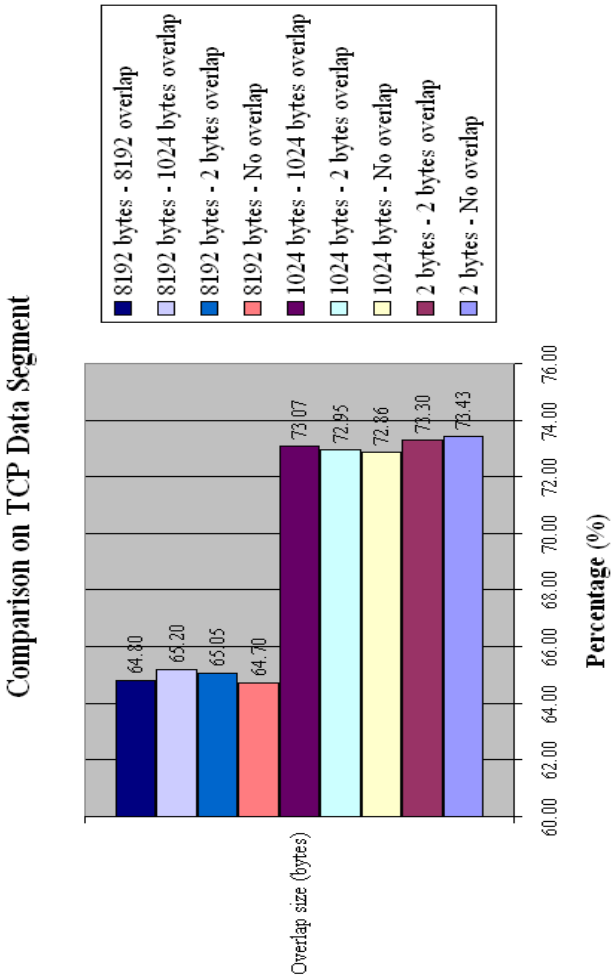

Figure. 27. The comparison on TCP Data segment on four nodes, blocking for two bytes, 1024 bytes and 8192 bytes message sizes

Through this research study, the measurement methodology implemented on the Beowulf cluster computing provides an alternative approach of finding the comparison between utilizing different program routines in an application. The study also focused on the effects of overlapping the message transfer and computation by focusing on the TCP/IP segment and the rate of the message.

VI. CONCLUSION

This research provides many novel findings on the developed Beowulf cluster system with its message-passing implementation. This Beowulf cluster has been compared to other cluster in many benchmarks as to exhibit that this setup has a comparable high-performance computing capability.

This cluster system shows the use the distributed memory system utilizing the message-passing interface programming model where the communication is via explicit messages primitives. These message primitives consist of the blocking and non-blocking communications. The blocking communication involves the send/receive request and waits until the reply is returned. However, when the programming model of non-blocking communication is used, the messages can return soon without waiting for the finish of communication operation because the communication operation can be managed by communication system in bottom layer of system. Therefore, the processor could treat the computation at the same time of dealing with the communication by the communication unit. This eventually allows the overlap of computation and communication. From this study, the research shows that the message rate will increase as the number of nodes increases. The average round-trip time also shows very small difference between the two MPI routines. It is demonstrated that for a long message size, the large difference in the average Mbit per second for the packets shows that the non-blocking overlap messages provides a more efficient communication compared to the blocking messages. The same phenomena can also be observed for the large message sizes. Therefore, this summarizes that the inherent characteristics of the parallel operation that as the message transfer is increasingly overlapped with computation, the TCP/IP overhead of the packet decreases, as illustrated in Fig. 28.

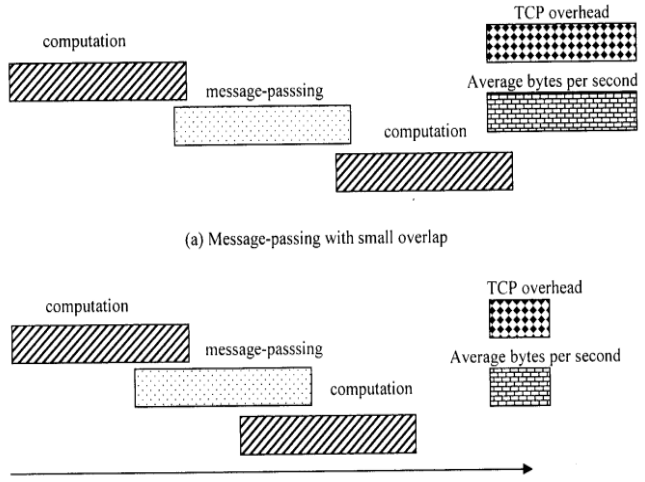

Figure. 28. Changes in TCP/IP overhead

This research introduces an alternative method to observe this phenomenon. By looking into the information on the packet data, time and rate based on the two different MPI routines, detailed studies can be examined which lead to the percentage of the TCP data segment and the rate of message transfer. The benefit of understanding the communication overhead of these distinct MPI communication primitives has the advantage for the programmer to write efficient parallel software and therefore will eventually contribute to the improved performance of parallel applications. Finally, the studies obtained from this research could be applied as key guidelines in developing parallel application program for future researches that employ similar Beowulf cluster computing system.

A. Recommendations for Future Research:

The communications performed by the MPI library routines require buffer space to complete the operation. Future research can look into the effect on the TCP overhead when

the size of this buffer space is changed. Apart from the point-to-point communication, future work could also examine the collective communication. There are the group message-passing routines where these routines send messages to a group of processes. They also receive messages from a group of processes. These collective routines are broadcast, scatter, gather and reduce. Hence, the comparison between the point-to-point and collective communications could provide the efficiency comparison on both categories of the MPI routines.

VII. REFERENCES

[1]. E. Hagersten and G. Papadopoulos, "Scanning the Technology. Parallel Computing in the Commercial Marketplace: Research and Innovation at Work," Proceedings of the IEEE, vol. 87 no. 3, 1999.

[2]. A. Grama, A. Gupta, G. Karypis, and V. Kumar, Introduction to Parallel Computing, Second ed. London: Pearson - Addison Wesley, 2003.

[3]. MPI - The Message Passing Interface (MPI) standard. [Online]. Available: http: //www.mpi-forum.org/, accessed on 30 Jan 2006.

[4]. G. K. Thiruvathukal, "Guest Editors' Introduction: Cluster Computing," Computing in Science & Engineering, vol. 7 no. 2, pp. 11-13, 2005.

[5]. M. J. Flynn, "Very High-Speed Computing Systems," Proceedings of the IEEE, vol. 54 no. 12, pp. 1901-1909, 1969.

[6]. M. Flynn, "Multiprocessors," in Chapter 8 Lectures, 1998.

[7]. K. T. Johnson, A. R. Hurson, and B. Shirazi, "General-Purpose Systolic Arrays," Computer, vol. Nov. 1993, pp. 20-31, 1993.

[8]. G. R. Luecke, B. Raffin, and J. J. Coyle, "Comparing the Communication Performance and Scalability of a Linux and a NT Cluster of PCs, a Cray Origin 2000, an IBM SP and a Cray T3E-600," 1999, pp. 26-35.

[9]. M. Sung, "SIMD Parallel Processing," Architectures Anonymous, vol. 6, pp. 11, 2000.

[10]. R. M. Karp and R. E. Miller, "Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing," J. Applied Mathematics, vol. 14, pp. 1390-1411, 1966.

[11]. H. Kai, W. Choming, W. Cho-Li, and X. Zhiwei, "Resource Scaling Effects on MPP Performance: The STAP Benchmark Implications," IEEE Transactions on Parallel and Distributed Systems, vol. 10 no. 5, pp. 509-527, 1999.

[12]. K. Watanabe, T. Otsuka, J. I. Tsuchiya, H. Amano, H. Harada, J. Yamamoto, H. Nishi, and T. Kudoh, "Performance Evaluation of RHiNET 2/NI: A Network Interface for Distributed Parallel Computing Systems," 2003, pp. 318-325.

[13]. M. Faidz, M. N. Taib, and S. Yahya, "Overlapping Effect of Message-Passing and Computation in a Beowulf Cluster Computing," unpublished, 2012.

[14]. M. Faidz, M. N. Taib, and S. Yahya, "Analysis of the MPI Communication Performance in a Distributed Memory System Architecture," unpublished, 2012.

[15]. K.-J. Andersson, D. Aronsson, and P. Karlsson, "An Evaluation of the System Performance of a Beowulf Cluster. Internal Report No. 2001:4," http://www.nsc.liu.se/grendel, 2001.

[16]. "Linux links, http://www.linuxlinks.com," accessed on 1 Feb 2007.

[17]. M. Perry, "Building Linux Beowulf Clusters," http://fscked.org/writings/clusters/cluster.html 2000.

[18]. "Linux Start, http://www.linuxlinks.com/," accessed on 1 Feb 2007.

[19]. "The Beowulf Underground," http://beowulf-underground.org/, accessed on 1 Feb 2007.

[20]. "The Linux HOWTO Index, http://sunsite.unc.edu/mdw/HOWTO," accessed on 1 Feb 2007.

[21]. "RedHat Linux, http://www.redhat.com/," accessed on 1 Feb 2007.

[22]. "Mandrake Linux, http://www.redhat.com/," accessed on 1 Feb 2007.

[23]. "Linux Software for Scientists, http://www.llp.fu-berlin.de/baum/linuxlist-a.html," accessed on 30 Jan 2007.

[24]. "Linux Gazette," http://www.linuxgazette.com/, accessed on 1 Feb 2007.

[25]. "Linux Journal's Linux Resources, http://www.ssc.com/linux," accessed on 1 Feb 2007.

[26]. S. Blank, "Using MPICH to Build a Small Private Beowulf Cluster," http://www.linuxjournal.com/article/5690, 2002.

[27]. "Scientific Applications on Linux, http://sal.kachinatech.com/," accessed on 1 Feb 2007.

[28]. "Beowulf (computing), http://en.wikipedia.org/wiki/Beowulf_(computing)#Original_Beowulf_HOWTO_Definition," accessed on 4 Dec 2007.

[29]. W. Feng, M. Warren, and E. Weigle, "The Bladed Beowulf: A Cost-Effective Alternative to Traditional Beowulfs," in Proc. The IEEE International Conference on Cluster Computing (CLUSTER'02), 2002.

[30]. W. Feng, M. Warren, and E. Weigle, "Honey, I Shrunk the Beowulf!," in Proc. The International Conference on Parallel Processing (ICPP'02), 2002.

[31]. L. Wen-lang, X. An-dong, and R. Wen, "The Construction and Test for a Small Beowulf Parallel Computing System," in Proc. Third International Symposium on Intelligent Information Technology and Security Informatics (IITSI), Jinggangshan, China, 2010, pp. 767-770.

[32]. M. Warren, E. H. Weigle, and W.-C. Feng, "High-Density Computing: A 240-Processor Beowulf in One Cubic Meter," in Proc. The IEEE/ACM SC2002 Conference (SC'02), 2002.

[33]. K. D. Underwood, R. R. Sass, and I. Walter B. Ligon, "Cost Effectiveness of an Adaptable Computing Cluster," in Proc. The ACM/IEEE SC2001 Conference (SC'01), 2001.

[34]. H. Kuo-Chan, C. Hsi-Ya, S. Cherng-Yeu, C. Chaur-Yi, and T. Shou-Cheng, "Benchmarking and Performance Evaluation of NCHC PC Cluster," National Center for High-Performance Computing, Hsinchu, Taiwan, 2000, pp. 923-928.

[35]. C. Yi-Hsing and J. W. Chen, "Designing an Enhanced PC Cluster System for Scalable Network Services," in Proc. 19th International Conference on Advanced Information Networking and Applications (AINA 2005), 2005, pp. 163-166.

[36]. P. Farrell, "Factors Involved in the Performance of Computations on Beowulf Clusters," Electronic Transactions on Numerical Analysis, vol. 15, pp. 211-224, 2003.

[37]. P. Uthayopas, T. Angskun, and J. Maneesilp, "On the Building of the Next Generation Integrated Environment for Beowulf Clusters," in Proc. The International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'02), 2002, pp. 1-6.

[38]. P. Uthayopas, S. Paisitbenchapol, T. Angskun, and J. Maneesilp, "System Management Framework and Tools for Beowulf Cluster," Computer and Network System Research Laboratory, Kasetsart University, Bangkok, 2000.

[39]. J. Stafford, "Beowulf Founder: Linux is Ready for High-Performance Computing," SearchOpenSource.com, 2004.

[40]. R. Kunz and J. Watson, "Clusters - Modern High Performance Computing Platforms," Penn State Applied Research Laboratory, 2004.

[41]. D. Fernandez Slezak, P. G. Turjanski, D. Montaldo, and E. E. Mocskos, "Hands-On Experience in HPC with Secondary School Students," IEEE Transactions on Education, vol. 53, pp. 128-135, 2010.

[42]. K. Yu-Kwong, "Parallel program execution on a heterogeneous PC cluster using task duplication," 2000, pp. 364-374.

[43]. L. Chi-Ho, P. Kui-Hong, and K. Jong-Hwan, "Hybrid parallel, evolutionary algorithms for constrained optimization utilizing PC clustering," 2001, pp. 1436-1441.

[44]. W. Gropp and E. Lusk, User's Guide for MPICH, a Portable Implementation of MPI Version 1.2.0: Argonne National Laboratory, University of Chicago, 1996.

[45]. H. Shan, J. P. Singh, L. Oliker, and R. Biswas, "Message Passing and Shared Address Space Parallelism on an SMP Cluster," Parallel Computing, vol. 29 no. 2, pp. 167-186, 2002.

[46]. K. D. Underwood and R. Brightwell, "The Impact of MPI Queue Usage on Message Latency," in Proc. International Conference on Parallel Processing (ICPP 2004), 2004, pp. 152-160.

[47]. C. Coti, T. Herault, P. Lemarinier, L. Pilard, A. Rezmerita, E. Rodriguez, and F. Cappello, "MPI Tools and Performance Studies - Blocking vs. Non-Blocking Coordinated Checkpointing for Large-Scale Fault Tolerant MPI," in Proc. 2006 ACM/IEEE Conference on Supercomputing Tampa, Florida 2006, pp. 127.

[48]. D. Grove and P. Coddington, "Precise MPI Performance Measurement using MPIBench," http://parallel.hpc.unsw.edu.au/HPCAsia/papers/72.pdf, 2001.

[49]. A. Danalis, K.-Y. Kim, L. Pollock, and M. Swany, "Transformations to Parallel Codes for Communication-Computation Overlap," in Proc. 2005 ACM/IEEE Conference on Supercomputing 2005, pp. 58.

[50]. J. C. Sancho, K. J. Barker, D. J. Kerbyson, and K. Davis, "Quantifying the Potential Benefit of Overlapping Communication and Computation in Large-Scale Scientific Applications," in Proc. ACM/IEEE SC 2006 Supercomputing Conference (SC '06), Tampa, FL, 2006, pp. 17-32.

[51]. A. Sohn, P. Yunheung, K. Jui-Yuan, Y. Kodama, and Y. Yamaguchi, "Communication Studies of Single-threaded and Multithreaded Distributed-Memory Machines," in Proc. Fifth International Symposium On High-Performance Computer Architecture, Orlando, FL, 1999, pp. 310-314.

