The High Performance Linpack (HPL) Benchmark Evaluation on UTP High Performance Computing Cluster

By

Wong Chun Shiang

16138

Dissertation submitted in partial fulfilment of

the requirements for the

Bachelor of Technology (Hons)

(Information and Communications Technology)

MAY 2015

Universiti Teknologi PETRONAS

Bandar Seri Iskandar

31750 Tronoh

Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

The High Performance Linpack (HPL) Benchmark Evaluation on UTP High Performance Computing Cluster

by

Wong Chun Shiang
16138

A project dissertation submitted to the
Information & Communication Technology Programme
Universiti Teknologi PETRONAS
In partial fulfillment of the requirement for the
BACHELOR OF TECHNOLOGY (Hons)
(INFORMATION & COMMUNICATION TECHNOLOGY)

Approved by,

____
(DR. IZZATDIN ABDUL AZIZ)

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

MAY 2015

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.

____
WONG CHUN SHIANG

ABSTRACT

UTP High Performance Computing Cluster (HPCC) is a collection of computing nodes using commercially available hardware interconnected within a network to communicate among the nodes. This campus wide cluster is used by researchers from internal UTP and external parties to compute intensive applications. However, the HPCC has never been benchmarked before. It is imperative to carry out a performance study to measure the true computing ability of this cluster.

This project aims to test the performance of a campus wide computing cluster using a selected benchmarking tool, the High Performance Linkpack (HPL). HPL is selected as a result of comparative studies and analysis with other HPC performance benhmarking tool. The optimal configuration of parameters of the HPL benchmark will be determined and run in the cluster to obtain the best performance.

Through this research project, it is the hope of the author that the outcome of this research project will help to determine the peak potential performance of the computing cluster.

ACKNOWLEDGEMENT

I would like to thank my supervisor, Mr Izzatdin Abul Aziz, for his guidance, help, motivation and patience. Without him, this project will not be able to be accomplished and I have learned a lot under his supervision about the whole research.

I would also want to thank to Universiti Teknologi Petronas (UTP) and the Computer and Information Sciences (CIS) department and the High Performance Computing (HPC) center for providing the resources and facilities needed to complete the project successfully. I would like to direct my gratitude to Mr Helmi for his helping hand and guidance whenever I face problems in completing the research project.

To my family members and friends, I truly appreciate your love and close support for me which compels me to finish the project.

For other people who have been involved directly or indirectly in my project, I am grateful for your help and offer my deepest thanks.

Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Background of Study

There are a number of factors which result in the formation and the ubiquity of computer clusters nowadays, such as the availability of high speed computer interconnections, the reduction in cost of components such as microprocessors, and emergence of parallel programs or software where distributed computers can work together. Computer clusters are increasingly used in a wide-ranging variety of purposes, sizes and compositions, varying between small clusters formed from a few computer nodes to huge clusters which form the fastest supercomputers in the world, capable of a few petaFLOPS (Floating Operations Per Second).

Benchmarking is needed in order to show the theoretical maximum performance and calculation ability of a computing cluster. By determining the max performance, the capability of a cluster and improvisation to the architecure and processes can be recognised. However, the parallel benchmark test have not been carried out in the High Performance Computing (HPC) Cluster in Universiti Teknologi Petronas (UTP) as of this time.

As such, the research project's main objectives are to investigate the importance of benchmark tools in gauging the performance of a computing cluster and carry out the profiling test on UTP's High Performance Computing Cluster. The test will use the HPL Linpack benchmark to measure performance.

The rest of this chapter will be structured as follows: Section 1.1 will explain about the problem statement of this project. And Section 1.2 will explain about the objectives section and also the scope section.

1.2 Problem Statement

Measuring and evaluating the performance of a computing cluster is important to identify the potential of a cluster and to develop strategic recommendations for its further development. As the performance of the HPC cluster in UTP has not been measured using any benchmarking tool, and thus its theoretical maximum potential performance cannot be readily determined and any potential improvements to the system cannot be identified.

1.3 Objectives

As such, some of the main objectives of this research are:

- To study on gauging the performance of a computing cluster through use of benchmarking tool.
- To carry out the benchmark testing tool in a small scale computing cluster such as UTP’s High Performance Computing Cluster.

1.4 Scope of Study

This benchmarking study will be limited to the High Performance Computing Cluster (HPCC) of Universiti Teknologi Petronas (UTP) as there is only a short period for this research to be completed. Only benchmark testing of the computer cluster located in UTP will be carried out.

Also, after the comparison study between different benchmarking tools has been completed, the HPL Linpack benchmark is chosen and only this benchmark tool will be implemented on the UTP HPC cluster. The selection of only one tool is needed to ensure consistent results and sufficient time to complete the research.

CHAPTER 2
LITERATURE REVIEW

2.1 Definition of a Computing Cluster

A computer cluster can be defined as a collection of stand-alone computers connected via a network which work together as a single system (El-Rewini & Abd-El-Barr, 2005). A computer cluster have each node set to perform the identical task, managed and planned by software and with all of its component subsystems are managed within a single administrative domain (Sterling, 2001). A cluster is normally enclosed within a room and handles as a single computer system (Bakery & Buyyaz, 1999). The components of a cluster, also known as nodes are connected to each other through networks such as fast Local Area Networks (LAN) or a hierarchy of networks or even several dispersed network structures (Bakery & Buyyaz, 1999), with each node running its own instance of an operating system (El-Rewini & Abd-El-Barr, 2005). In most circumstances, all of the nodes are similar in terms of hardware and operating system. In a small-scale computer cluster with Beowulf architecture, most cluster nodes contain commercial hardware and can perform operations independently (Sterling, 2001).

The UTP HPC cluster for the UTP campus comprises of sixty cluster nodes. Each of the nodes contain AMD processors and AMD / Nvidia GPUs in various configurations. Ten out of twenty cloud nodes in the HPC cluster have been chosen to run the benchmark, and a detailed specification of the Cloud Nodes is shown below.

The suite of software running on the nodes include the Ubuntu Linux 14.0.1 LTS operating system, mpich2 Message Passing Interface (MPI) and Automatically Tuned Linear Algebra Software (ATLAS) as the Basic Linear Algebra Subprogram (BLAS).

A detailed list of technical specifications about a single UTP HPC cloud node is as follows (all of them are configured identically):

TABLE 1. Technical Specifications of Cloud Nodes

Processor	8 core AMD FX 8150 processor 3.1 GHz
Graphics Card	AMD HD7970
Memory	32 GB DDR3
Interconnection bet. nodes	Ethernet
Operating System	Ubuntu Linux 14.0.1 LTS

2.2 Working Principle Behind A Computing Cluster

The working principle of parallel computation enables the high number of calculations or floating point operations per second (FLOPS) by interconnected computer cluster nodes. Parallel computation are effective when the calculations can be conducted in parallel and are calculated at the same time by dividing them to be handled by different processors (Barney, 2010).

A single calculation process usually consists of multiple parts. These parts can be broken down and translated into multiple instructions. The commands in each part can be completed by multiple processors at the same time, while under the regulation or synchronisation of a central mechanism (Barney, 2010). The time taken for the problem to be resolved can be significantly shortened by spreading the work load among several processors.

Amdahl's law dictates this improvement in speed of execution when there are multiple processors (Rodgers, 1985). Where *n* is the number of computational threads, and *B* is the portion of the process that can only run in serial, the time *T(n)* for the process for be completed is:

$$T(n) = T(1)\left(B + \frac{1}{n}\,(1 - B)\right)$$

And the improvement in computation time, also known as speedup, *S(n)* is calculated by (Rodgers, 1985):

$$S(n) = \frac{T(1)}{T(n)} = \frac{T(1)}{T(1)\left(B + \frac{1}{n}(1 - B)\right)} = \frac{1}{B + \frac{1}{n}(1 - B)}$$

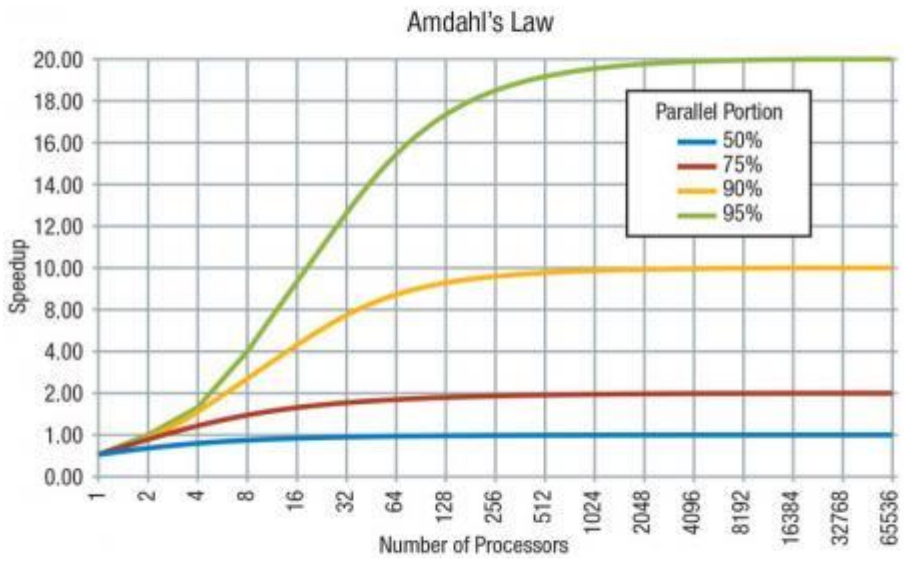


FIGURE 1. Graph of Amdahl's law

This graph illustrates the improvement in speed (or speedup) of a process relative to the percentage of portion in the process that can run in parallel. When the percentage of parallel portion in the process increases along with the number of processors, the speedup in processing time will have increase significantly. However the speedup will increase up to a certain number of processors (again depending on the percentage, higher percentage has a limit of higher number of processors), after which the speedup will plateau.

2.3 Factors For Employing Computer Clusters

Computer clusters have a faster execution speed as compared to a single computer, while typically being much more cost-effective than single computers of comparable speed or availability (Bader & Pennington, 2001). A computer cluster can have better availability Certain large and complex computational problems require parallel computing to be effectively and efficiently solved. Usual examples of such problems include weather modeling, computational fluid dynamics (CFD) simulation, designing semiconductors etc. There are also many real world problems and occurrences that are well-matched for parallel computations (Barney, 2010).

The processors currently available are better suited for parallel computation, with features such as hyperthreading and virtualisation (Barney, 2010). Parallel computation can be time saving and money saving. This is due to running a task using more computing resources will result in a shorter time to completion and less resource time usage (Barney, 2010). Parallel computations can also cost less money when using cheaper hardware which are off the shelf.

Clusters which load balancing feature distributes network service requests between the nodes of the cluster, so that the burden is equally distributed between them. This load balancing feature enables failure management, because if one of the nodes have failed, the software responsible for load balancing will redistribute its requests to the other nodes, and makes the failure invisible to the end user (Red Hat, n.d.). Load balancing also enables scalability, distributing to more nodes or less nodes as needed, which is useful and cost effective for organisations (Red Hat, n.d.).

To balance out the processes and operations of software which run on servers (a Web server, parallel processing programs etc), the requests from clients are allocated to several servers at one time (Microsoft, n.d.-b). All inbound client requests for service are accepted and forwarded to particular servers to evenly utilize the servers using load balancing technologies, for example load balancers. Each of the servers will simultaneously process several requests for service at one time. In the case of Web servers, several elements of the webpage shown to the client can be gathered from a few host machines in the cluster, so that the webpage can be loaded in a short time.

By distributing loads for processing by multiple nodes, the clients get serviced faster and in a shorter time. Besides that, more servers are utilized in processing requests, instead of only a few servers being occupied with requests while others are idling (Microsoft, n.d.-b).

2.4 Definition of a Benchmark

A benchmark's main function is to perform a selection of complex problem solving tests so as to evaluate the potential capability or performance of something. Generally, benchmarks are used against computer hardware to measure the maximum achievable performance under the test conditions, such as the floating point operations per second (FLOPs) of a CPU.

The capabilities and profile of the clusters and the factors which influence their performance need to be comprehended and analyzed in order to improve on the processes and performance of the clusters. The benchmark results also cancan also show how different configurations can may affect the performance of the computation.

A few different benchmarks are available. In this study, the High Performance Linpack (HPL) benchmarking tool and NAS Parallel benchmarking tools are compared and examined.

2.4.1 High Performance Linpack (HPL)

The HPL benchmarking tool is a portable application which works across various platforms and is written in C (Jack J Dongarra, Bunch, Moler, & Stewart, 1979). The Linpack benchmark was primarily an auxiliary program which is developed from the Linpack package. The function of the program is to analyse and solve linear equations using FORTRAN. Linpack Fortran n = 100 benchmark is one of the benchmarks, where the problem size is 100. Linpack n = 1000 benchmark has a problem size of 1000 while there is another benchmark known as Linpack's Highly Parallel Computing benchmark (J. J. Dongarra, 1990).

The Linpack Benchmark is a measurement of the computing power by gauging the rate of calculation of a computer. FORTRAN functions are run which decomposes and resolves a dense matrix into complex linear equations systems and linear least-squares problems in double precision (J. J. Dongarra, 1990). This enables the benchmarking tool to establish the floating-point computation speed of the computing cluster. One of the features of the Linpack benchmark is that it is able to handle distributed memory and vector supercomputers (J. J. Dongarra, 1990).

Developed from the Linpack benchmark, the High Performance Linpack (HPL) benchmark tool is a software suite developed for computers with distributed memory such as a computer cluster, that similar to the Linpack benchmark, calculates and resolves a dense linear system in double precision arithmetic of 64 bits (Petitet, Whaley, Dongarra, & Cleary, 2004). The HPL benchmark has been used by various organization and institutions as a standard yardstick to evaluate the general floating-point rate performance of their supercomputers since its introduction. The TOP500 project employs the HPL benchmark suite to use as a standard of measurement of performance benchmarking or profiling.

The HPL benchmark solves linear algebraic problems by breaking down the matrix using Lower-Upper (LU) factorization. LU factorization decomposes a matrix as a by multiplying the lower triangular part of matrix with the upper triangular part of matrix. One example of an equation that the benchmark tool solves is:

$$Ax = b; \; A \in R^{nxn}; x, b \in R^n \qquad (1)$$

The left side of the equation contains matrix A, while the right hand side is a vector b. The solution to the problem is established by calculating the factor of A.

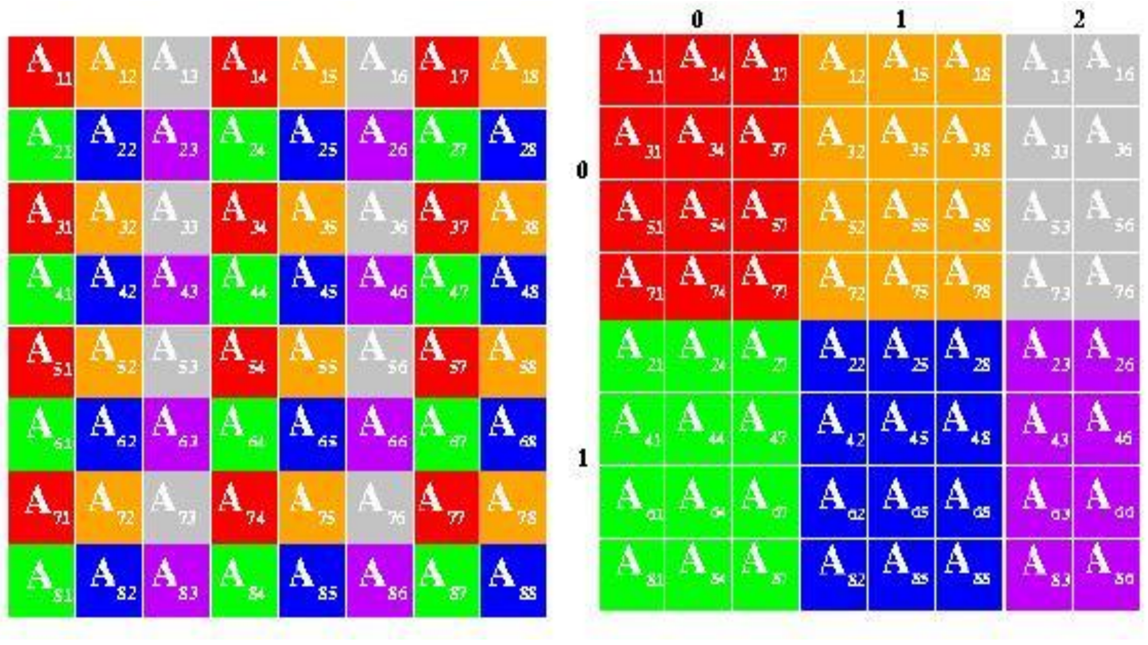


FIGURE 2. Structed Matrices (Petitet et al., 2004)

Provided a matrix *A* and right-hand-side vector *b*, the algorithm of the HPL performs an *LU* factorization calculation through partial pivoting of rows of the matrix *[A b] = [[L,U] y]* with the coefficient of *n*-by-*n*+1 in order to solve a linear system with the order n in equation (1) (J. Dongarra, Luszczek, & Petitet, 2001).

The decomposition of the dense matrix A is then commenced and the final outcome of the calculation are well-structured matrices where every one of its elements are non-zero. Calculating the equation of *U x = y* in the upper triangular resolves into the solution *x* given that the lower triangular factor *L* is applied to b as the factorization progresses. The only unpivoted part of the matrix is the lower triangular matrix *L* and is not returned to the calculation (Petitet et al., 2004).

To make sure load balancing is well-adjusted and the ability to scale to multiple computers, the results of calculation is allocated onto a two-dimensional P-by-Q grid of processes and structured using block-cyclic organisation. The matrix with *n*-by-*n*+1 coefficient is then segregated into NB-by-NB blocks according to logic, which are intermittently distributed into the P-by-Q process grid. The process is repeated for both width and height of the matrix (Petitet et al., 2004).

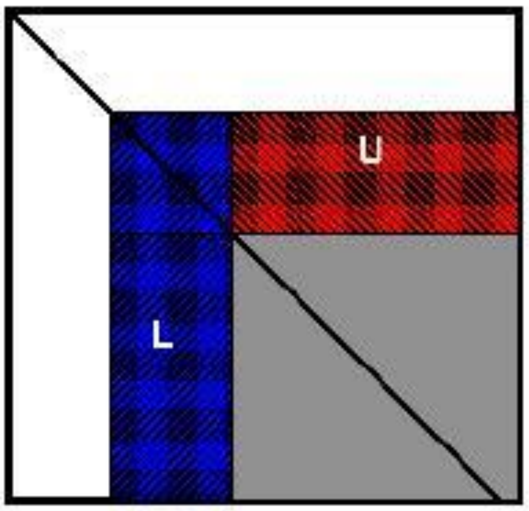


FIGURE 3. LU matrix factorization (Petitet et al., 2004)

The main iteration of the LU factorization calculation will select and employ the right-looking variant. Each repetition of the calculation loop will factorize a section of NB columns and after that, updates to the trailing submatrix is applied. The identical block size NB that was intended for distribution of data is used to logically divide the computation into partitions (Petitet et al., 2004).

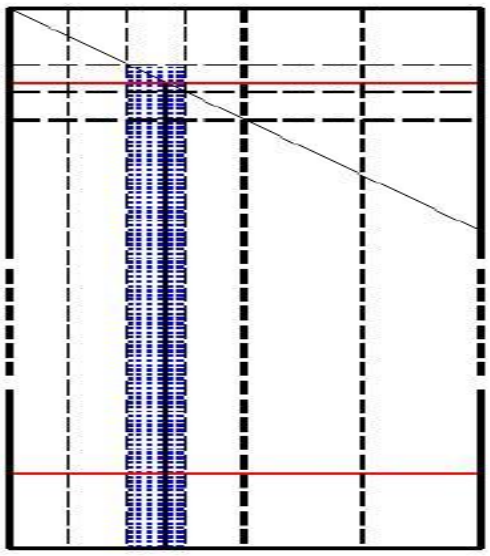


FIGURE 4. Panel factorization (Petitet et al., 2004)

Every one of the panel factorization happens in one column of processes at a specific repetition of the main iteration and according to the distribution system's Cartesian property. This specific calculation method is an important part of the critical path in the complete process. There are three recursive variants of matrix multiplication methods offered to the user, which are the Crout method, left-looking method and right-looking method. The user is similarly permitted to adjust the number of sub-panels the main panel is separated into when separation occurs in the repetition of algorithm. Another selection factor for the user is the criteria to stop the run-time of the recursion, such as how many columns are left to factorise.

Upon reaching this maximum limit, factorisation of the sub-panel will be calculated according to the user selected variant out of the three matrix-vector based variant (Crout, left- or right-looking) (Imran, Nor, & Othman, 2013). After that, every panel of column is communicated in a single process which merges the pivot search, the accompanying swap and broadcast procedure of the pivot row. The three processes are executed at the same time using a binary-exchange (leave-on-all) reduction (Petitet et al., 2004).

The resulting panel of columns is transmitted to the other process columns using broadcast with the completion of the panel factorization operation. When the panel has been broadcasted to the other columns, updates are applied to the resulting submatrix with the last panel in the look-ahead pipe. This is due to the factorization of the panel is one of the critical operations in the algorithm, so with the completion of factorization and broadcasting of panel k, the panel of k+1 will be factorized and broadcasted to the other columns ensues (Petitet et al., 2004).

This method is known in literature as "look-ahead" or "send-ahead" method. The user can choose several depth values of look-ahead for this software. A zero depth value brings about no lookahead in normal situations, which means the panel presently being broadcast will affect the following submatrix. The look-ahead technique retains all the panels of columns which are presently in the look-ahead pipe by using up extra memory. According to the authors, the value 1 or 2 of look-ahead depth value possibly enables the greatest improvement in terms of performance (Petitet et al., 2004).

2.4.1 NAS Parallel Benchmark

The NAS Parallel Benchmark (Bailey et al., 1991) (NPB) suite are a set of computer programs. The benchmark is designed for testing parallel computer clusters in order to gauge the performance of parallel computer clusters. NAS Parallel Benchmarks are frequently used by organisations as an alternative of HPL to measure of cluster performance.

The NAS benchmark was created as the widespread kernel benchmarks such as Livermore Loops, the Linpack benchmark and the NAS Kernels are more suited to evaluate vector supercomputers, and not the highly parallel machines popularly used nowadays (Bailey et al., 1991).

There are eight benchmark modules available in the NAS Parallel Benchmark suite. In the newest NPB version, there are additional modules are included (UA, DC and DT). The five problem sizes available for each of the applications are class A, class B, class C, class D and class E, increasing in problem size with each class. The detailed working behind each of the NAS benchmarks is explained as follows (Bailey et al., 1991):

TABLE 2. Operations of Each NAS Parallel Benchmark Modules.

Benchmark Module	Operations
Multi Grid	Employs the V-cycle multigrid technique to find the approximate solution to a three-dimensional (3D) discrete Poisson equation.
Conjugate Gradient	Applies the inverse iteration to approximate the lowest eigenvalue of a complex sparse symmetric positive-definite matrix problem. The conjugate gradient method is a subprocedure used to resolve the system of linear equations.

Fast Fourier Transform	Apply the fast Fourier transform (FFT) method to resolve a three-dimensional (3D) partial differential equation (PDE).
Integer Sort	Utilizes bucket sort algorithm to assign a list of integers positions in the final sorted list accordingly (Grün & Hillebrand, 1998).
Embarrassingly Parallel	Employing the Marsaglia polar process to produce independent Gaussian random variates.
Block Tridiagonal, Scalar Pentadiagonal, Lower-Upper symmetric Gauss-Seidel	Find the answer to a nonlinear PDEs synthetic system using three different process of calculations involving block tridiagonal, scalar pentadiagonal or symmetric successive over-relaxation (SSOR) problem solving.
Unstructured Adaptive	Find the solution to a heat problem of a ball in motion with convection and diffusion effects. The mesh has to be adaptive to the conditions and is recalculated every five steps of calculation and the memory is retrieved dynamically and erratically.

2.4.2 Comparative Study Between HPL and NAS Parallel Benchmarks

TABLE 3. Comparative Study Between HPL and NAS Parallel Benchmarks

Benchmarks	**HPL Linpack**	**NAS Parallel**
Problem solving method	Solves a dense matrix problem using LU factorization	Solves calculations involving simulations. Conjugate Gradient (CG) uses the inverse iteration to approximate the lowest eigenvalue of a complex sparse symmetric positive-definite matrix problem. The conjugate gradient method is a procedure used to resolve linear equations (Bailey et al., 1991)
Modules	Single (LU factorization)	Multiple (11 modules in total)
Suited for Parallel Computation	Designed for Parallel Computers	Designed for Parallel Computers
Used widely by organisations as standard	Yes (TOP 500 list)	No

After having a comparison study between HPL and NAS benchmarks, it is the decision of the author to use HPL as HPL measures performance using less number of modules. Having less number of modules possibly will take less time than complete than the NAS benchmark due to less number of modules to complete. The TOP500 organisation uses this benchmark to list the world's fastest computer clusters (Strohmaier, Dongarra, Simon, Meuer, & Meuer, 2015), which can help to determine the performance.

CHAPTER 3

METHODOLOGY

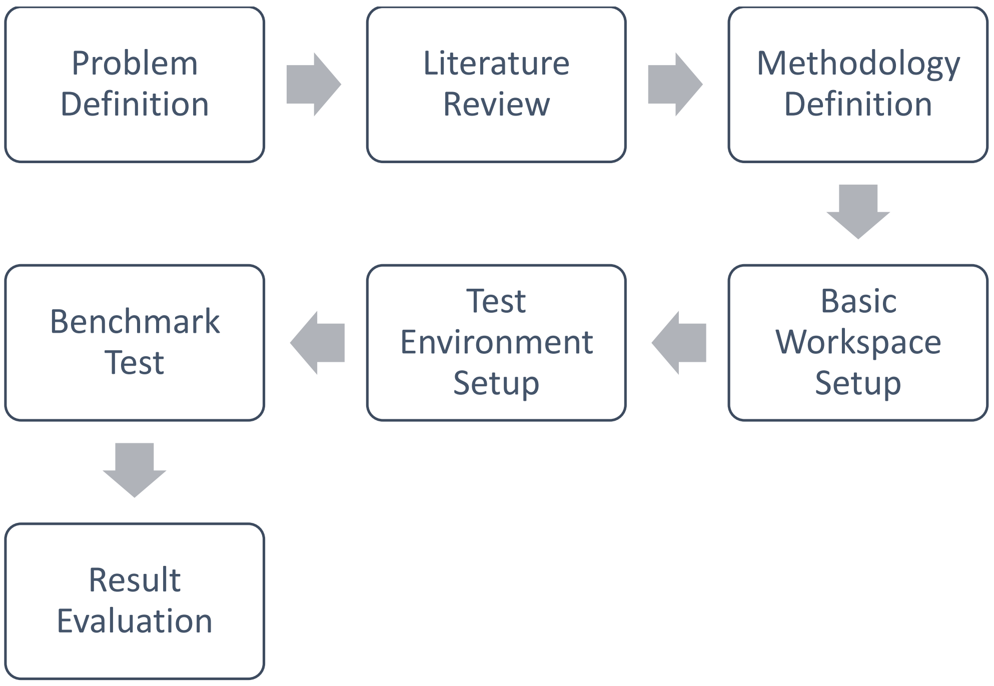


FIGURE 5. Research Methodology

This figure shows the development methodology of this final year project. There are six main stages: Problem Definition, Literature Review, Methodology Defintion, Basic Workspace Setup, Test Environment Setup, Benchmark Test, and Result Evaluation.

3.1 Problem Definition

In this part of the methodology, the problem is defined and clearly detailed. Measuring and evaluating the performance of a cluster of computers is important so that the potential of a cluster is recognized and the managers can have an idea for what to be further developed to improve the system. The problem which the author has defined is that the performance of the HPC cluster in UTP has not been measured using a benchmark tool.

3.2 Literature Review

In the literature review part, all the related literature in relation to the benchmarks and the functionality and inner workings of the benchmark software are reviewed. This includes reviewing the steps to be taken and the details on the execution of the benchmark tools.

A comparative study between the different benchmarks is carried out to select the most suitable benchmark tool to benchmark the UTP HPC campus cluster. The differences between the HPL and NAS benchmarking tools have been compared and contrasted, and

Also, literature review on optimal configurations of the benchmark cluster have been undertaken. The information gathered through literature review is that there are seven parameters in the HPL benchmark which need to be optimised for the best results, which are: problem size, N, distribution size, NB,

3.3 Methodology Definition

After the related literature have been reviewed, all the processes which need to be completed in order to compile, configure and execute the selected benchmark, which is High Performance Linpack (HPL), will be listed out and examined.

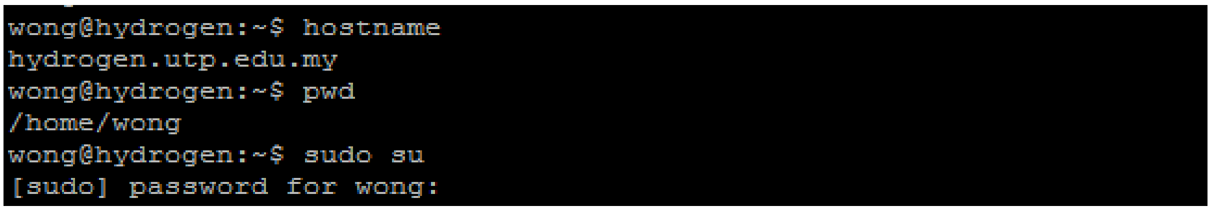
3.4 Basic Workspace Setup

Setting up the workspace involves creating user account, ensuring accesss to required resources such as usernames and passwords to server nodes, and dowloading the essential files.

3.4.1 Creating New User

The steps to create a new user are as follows:

1. Obtain administrative permission by logging in as a super user.
2. Use the *useradd* command to create new user, example: *useradd johnsmith.*
3. To set password for johnsmith, use the command *passwd johnsmith.* A New Password prompt will appear.
4. Enter the password as desired into the field and press ENTER.
5. Reconfirm the new password and the password is successfully set for the user account.

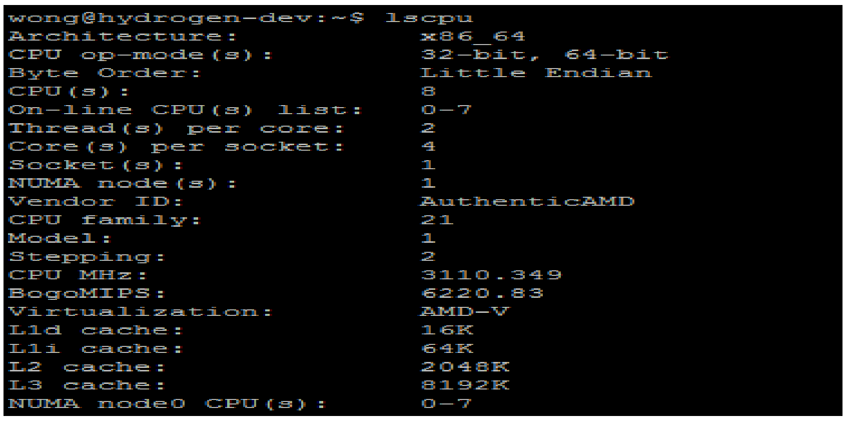


```
wong@hydrogen:~$ hostname
hydrogen.utp.edu.my
wong@hydrogen:~$ pwd
/home/wong
wong@hydrogen:~$ sudo su
[sudo] password for wong:
```

FIGURE 6. Screenshot of logging as super user (su)

3.4.2 Checking CPU specifications

To check the specifications of the CPU, use the command *lscpu*. The specifications of the CPU is then listed, including the architecture, CPU operation modes, vendor, MHz of the CPU, cores per CPU etc.



```
wong@hydrogen-dev:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             AuthenticAMD
CPU family:            21
Model:                 1
Stepping:              2
CPU MHz:               3110.349
BogoMIPS:              6220.83
Virtualization:        AMD-V
L1d cache:             16K
L1i cache:             64K
L2 cache:              2048K
L3 cache:              8192K
NUMA node0 CPU(s):     0-7
```

FIGURE 7. Screenshot of checking CPU specifications (lscpu)

From the screenshot, the localhost node has an 8-core AMD CPU, with each core running at around 3.1 GHz processing speed and having 64-bit architecture. All the CPUs are online.

3.4.3 Checking top running processes

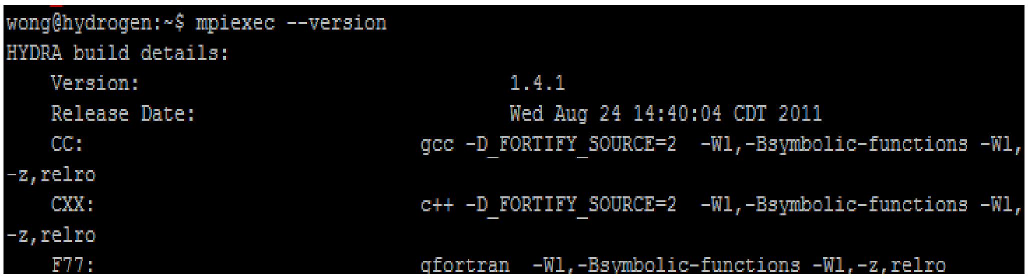
To check the top running processes currently in the system, use the command *ps*. The list of processes will be shown, including the user running the process and percentage of CPU usage and memory usage etc.

FIGURE 8. Screenshot of top running processes and details (ps)

As shown in the screenshot, there are 2 running tasks in the system and 222 sleeping tasks. The CPU usage, memory usage, and swap usage is also shown. Under that, the details on each of the processes are shown including PID, user who launced the task, stats such as percentage of CPU and memory used, how long the process has been running and the command being run.

3.4.4 Checking MPI version

To check the version of Message Passing Interface (MPI) used by the system, use the command *mpiexec --version*. The list of processes will be shown, including the user running the process and percentage of CPU usage and memory usage etc.



```
wong@hydrogen:~$ mpiexec --version
HYDRA build details:
    Version:                                 1.4.1
    Release Date:                            Wed Aug 24 14:40:04 CDT 2011
    CC:                              gcc -D_FORTIFY_SOURCE=2  -Wl,-Bsymbolic-functions -Wl,
-z,relro
    CXX:                             c++ -D_FORTIFY_SOURCE=2  -Wl,-Bsymbolic-functions -Wl,
-z,relro
    F77:                             gfortran  -Wl,-Bsymbolic-functions -Wl,-z,relro
```

FIGURE 9. Screenshot of MPI version (mpiexec --version)

As shown in the screenshot, the MPI version running in the system is OpenMPI version 1.4.1.

3.5 Test Environment Setup

One of the benchmark configurations needed in running the benchmark is to setup the High Performance Linpack configurations. Determining and selecting the correct compiler for C and FORTRAN is one of the parts in this stage. The message passing interface (MPI) also needs to be preinstalled and setup. For the UTP cluster, mpich2 will be used.

The Basic Linear Algebra Subprograms (BLAS) which is responsible for the basic mathematical procedures involving vector and matrix also needs to be setup and configured. The BLAS library contains a specific collection of low-level subprocedures for common linear algebraic operations. BLAS contains 3 levels: Level 1 is employed in vector procedures, Level 2 completes processes between matrix and vector, whereas at Level 3, matrix-matrix processes are calculated. BLAS is frequently utilised in creating software tools which need to perform linear algebra, such as Linpack and LAPACK, as they have high efficiency, are transferrable across platforms and have many open source implementations (Strohmaier et al., 2015).

The Automatically Tuned Linear Algebra Software (ATLAS) is another BLAS routine alternative which employs practical procedures for better execution and portability across platforms. The interfaces of ATLAS are written using C and includes some LAPACK routines (Strohmaier et al., 2015). The Linpack does not have high efficiency in resolving matrix computations because of the memory access method by both the algorithm and software, which decreases the overall efficiency, and has been superseded by LAPACK (Strohmaier et al., 2015). LAPACK is a suite of software which can resolve linear algebra involving matrices, with distinctive specialization towards series of linear equations, least squares calculations, eigenvalue calculations, and decomposition of singular value (Strohmaier et al., 2015). The basis of the software emulates the use of block partitioned matrix techniques in order to accomplish great performance on systems with RISC architecture, vector computers, and parallel processors with common memory (Strohmaier et al., 2015). The ATLAS library will be used as BLAS library for the HPL benchmark.

In the following phase, the parameters to HPL.dat is tuned. There are 17 parameters which need to be assigned to HPL.dat. In these 17 parameters, only seven of these parameters are usually configured according to the cluster during benchmarking process. The default good start value will normally be used for the remaining parameters as suggested by HPL. The seven parameters which need to be configured are: problem size (N), processor grid (P x Q), broadcast (BCAST), block size (NB), panel factorization (PFACT), recursive panel factorization (RFACT), and look-ahead depth (DEPTH).

$$N = \sqrt{Total\ amount\ of\ Memory\ (in\ bytes)} \times 0.80 \qquad (2)$$

The solution to equation (2) shown above is theoretically the best problem size, N to be solved. Research by Petitet,Whaley, Dongarra and Cleary suggest that the largest size of N which can fit around 80% of the memory should be used (Petitet et al., 2004). However, when assigning the size of N too large, data swap can happen between memory and disk, which will lead to a reduction in the overall performance, as the system will need to read from the disk instead of directly from the memory. Thus, only 80% of the total problem size will be utilised, with the remaining 20% left for other uses (Petitet et al., 2004).

The processor grid (P x Q) parameters that denote the size or proportions of the process grid. In the processor grid (P x Q), P represents the process rows while Q represents the process columns. The size of both P and Q should be determined by the physical interconnection network. For a mesh or a switch network, which is preferred, the values of P and Q should be approximately equal, with Q having a slightly larger value than P. Nevertheless, a the research conducted in Universiti Teknologi MARA on their Khaldun Sandbox Cluster, after trying a number of configurations, their findings was that the best values for the processor grid are P is 2 while Q is 16 in order to achieve the best benchmark results among the configurations of 26.88 Gflops (Imran et al., 2013). This finding is a little contrasting with the recommended processor grid configuration, and will need to be checked out.

In HPL, the block size, NB is used to allocate the data and also determines the level of detail in computation. From the perspective of data distribution, having a smaller block size will result in a better distributed load equilibrium. Nevertheless, in the view of computation, when the value of NB is too small, the NB can be the limiting factor of the computational performance by a large factor due to almost none of data will be reused in the highest level of the memory hierarchy (Petitet et al., 2004). This problem will result in increase of the number of messages. Therefore, the values of block size for highest performance usually lies between 32 to 256 intervals. The best values largely relies on the computation per communication performance ratio of a system (Petitet et al., 2004). A balanced between performance and data distribution size needs to be achieved for the best performance in benchmark result. There are also panel factorization and recursive panel factorization variants to choose from, which are: right-looking variant, left-looking variant and Crout variant. These three variants are different methods in which these computations are carried out (Petitet et al., 2004), and can have minor performance differences in the result.

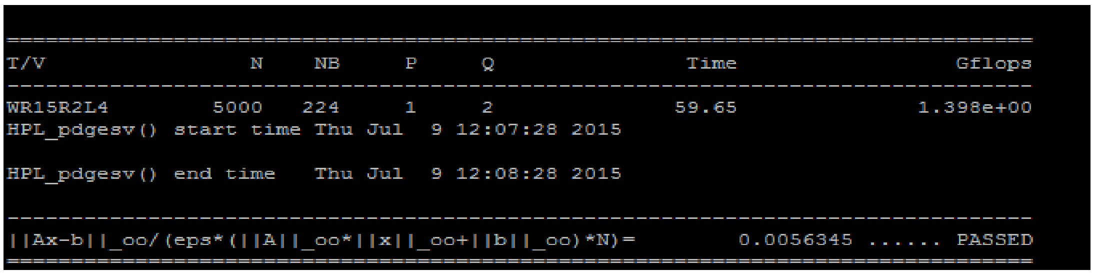
Once the factorization of the panel had been calculated, HPL broadcasts panel factorization column from one process column to other process columns. There are 6 alternatives of broadcast algorithms available can be employed, which are Increasing-ring, Modified Increasing-ring, Increasing-2-ring, Modified Increasing-2-ring, Long bandwidth reducing and modified Long bandwidth reducing (Petitet et al., 2004). Research has suggested that the Modified Increasing-ring algorithm is one of the best in efficiency (Microsoft, n.d.-a). However, in the research conducted in Universiti Teknologi MARA on their Khaldun Sandbox Cluster, the results from their benchmark test saw that the Modified Increasing-ring does not perform as well as compared to the Long bandwidth reducing algorithm. The Long bandwidth reducing algorithm allows them to obtain their best results of 31.33 Gflops (Imran et al., 2013). This findings will be tested when benchmarking the UTP HPC cluster.

The look-ahead depth is also an important parameter. Look-ahead denotes the ability of the algorithm to reorganize the most efficient operations to run with the least efficient, and store the panels being currently factorized in the memory for a better performance. A look-ahead depth of 1 is recommended by most use cases (Petitet et al., 2004).

3.6 Benchmark Test

Once all the benchmark configuration processes have been completed, test runs will be conducted using the benchmarking tool on the 10 cloud nodes of the HPC cluster of 4 nodes to determine the optimal benchmark configuration and parameters. After the test runs are successful, the benchmark tool will be configured with the optimal parameters from evaluating the test runs and the final test run is carried out. Details on the optimal parameters and results of the test runs are detailed more in Chapter 5: Results and Discussion part. The results from the benchmark tool will be recorded and evaluation of the results will be carried out in the following section.

3.7 Results Evaluation



```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR15R2L4        5000   224     1     2              59.65              1.398e+00
HPL_pdgesv() start time Thu Jul  9 12:07:28 2015

HPL_pdgesv() end time   Thu Jul  9 12:08:28 2015

--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0056345 ...... PASSED
================================================================================
```

FIGURE 10. Screenshot of a HPL test result

The results obtained after the benchmark tool has been executed, as shown in Figure 10, is then collected and evaluated in order to determine if the benchmark test is successful and ensure that no problems are encountered during the execution. In addition, the results will be examined to make sure that the configuration on the benchmark has been optimised for the best results.

The results from running the benchmark tool will be specified and detailed more in the Chapter 5: Results and Discussion in this paper.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Test Run Results

Seven test runs are conducted on the HPC cluster based on the seven main parameters to be configured in the HPL benchmark, which are Number of Nodes (P x Q), Problem Size (N), Distribution Size (NB), Factorization (FACT), Recursive Factorization (RFACT), Broadcast (BCAST) and Look-ahead Depth (DEPTH). The results and discussion from each of the test runs are shown and detailed in the following sections.

4.1.1 Number of Nodes, P x Q

The purpose of the first testing is to determine how the number of nodes, P x Q, influence the speed of processing and the time taken to solve the problem. The other parameters to be tested are fixed. The results of the test are as follows:

TABLE 4. Number of Nodes and Speed of Processing.

Number of Nodes	Time Taken (s)	Speed of Processing (Gflops)	Speedup in Processing
1	29.62	2.815	1.00
2	59.65	1.398	0.50
4	39.86	2.092	0.75
6	26.21	3.181	1.13
8	23.17	3.597	1.28
10	18.79	4.436	1.58

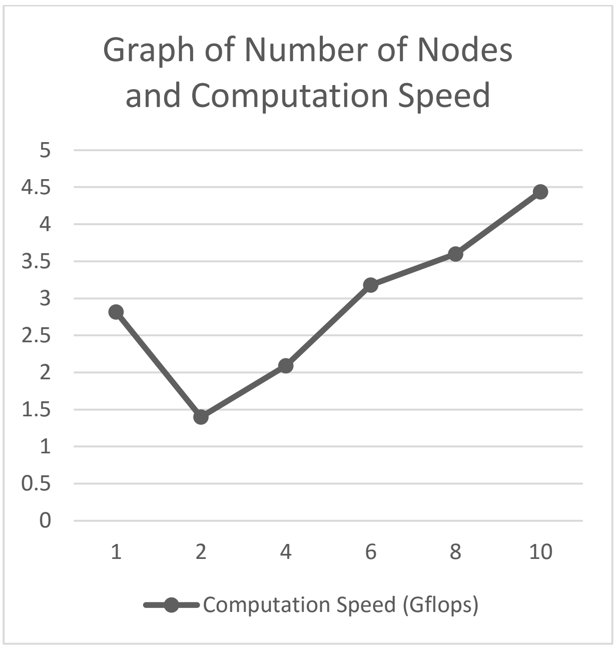


FIGURE 11. Number of Nodes and Speed of Processing.

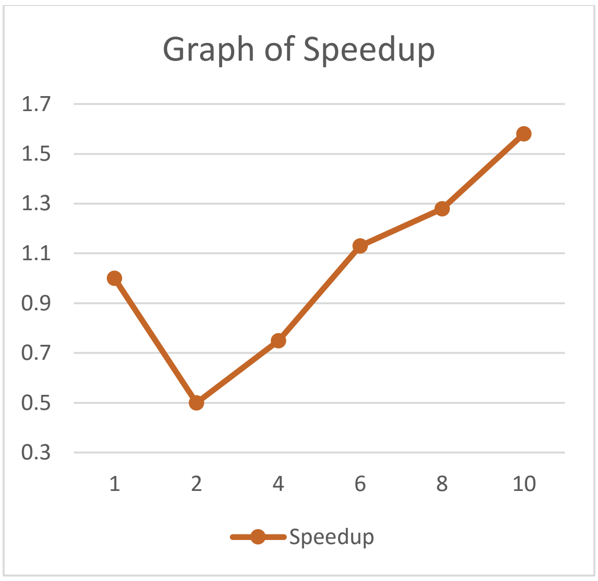


FIGURE 12. Number of Nodes and Speedup in Processing.

As number of nodes increases from one node to two nodes, the computation speed drops as shown in Figure 11. due to a delay of communication. This delay is caused by the computation problem being transmitted over an Ethernet network through Message Passing Interface (MPI) and distributed among the cluster nodes. The delay in communication increases the overhead for problem computation and can cause a slowdown in performance as shown in Figure 12.

But when the number of nodes keeps increasing, the delay in communication is compensated by the speed of processing of the cluster nodes. Thus, the speedup in processing increases until all the processors are saturated with computations (saturation point), then the increase in speed remains constant.

4.1.2 Problem Size, N

The purpose of the second testing is to determine how the problem size, N influence the speed of processing and the time taken to solve the problem. The other parameters to be tested are fixed. The results of the test are as follows:

TABLE 5. Problem Size and Speed of Processing.

Problem Size, N	Time Taken (s)	Speed of Processing (Gflops)	Speedup
5000	18.79	4.436	1.00
10000	114.68	5.815	1.31
15000	374.78	6.003	1.35
20000	812.09	6.568	1.48
25000	1605.56	6.488	1.46

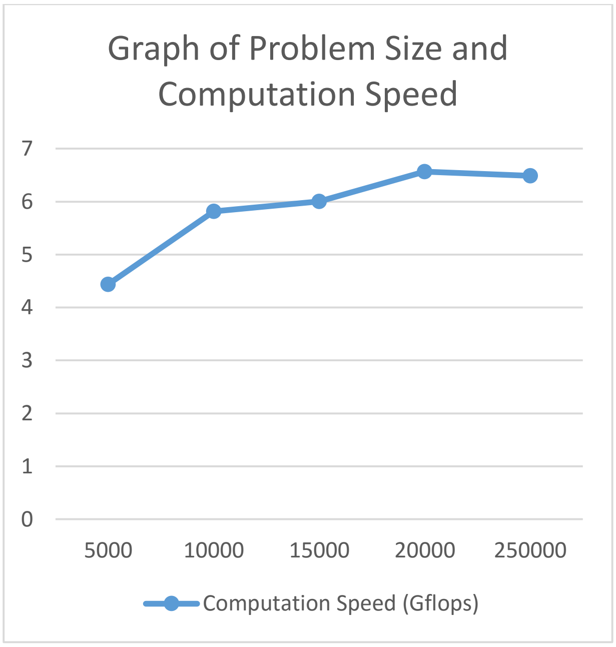


FIGURE 13. Problem Size and Speed of Processing.

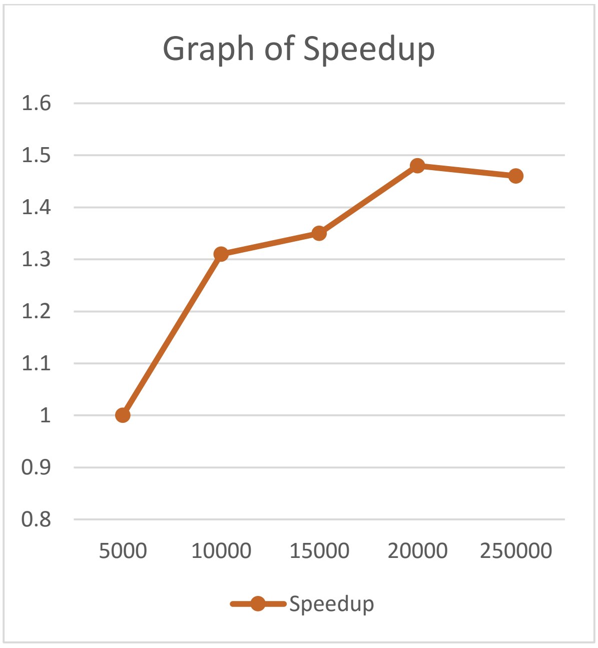


FIGURE 14. Problem Size and Speedup in Processing.

Problem size dictates the size of the matrix to be decomposed. A larger problem size engages more processing power in finding the solution, and thus resulting in a higher computation speed as shown in Graph 13.

However the increase in computation speed or speedup will only keep increasing until a saturation point, where all the processors are being used to solve the problem, then the increase in speed stabilises. This is due to the maximum effectiveness and efficiency of processing power had been reached.

4.1.3 Distribution Size, NB

The purpose of the third testing is to determine how the distriution size, NB influence the speed of processing and the time taken to solve the problem. The other parameters to be tested are fixed. The results of the test are as follows:

TABLE 6. Distribution Size and Speed of Processing.

Distribution Size, NB	Time Taken (s)	Speed of Processing (Gflops)
100	19.36	4.307
125	17.86	4.667
150	20.99	3.973
175	18.54	4.497
200	23.17	4.406
225	22.30	3.738
250	21.30	4.706

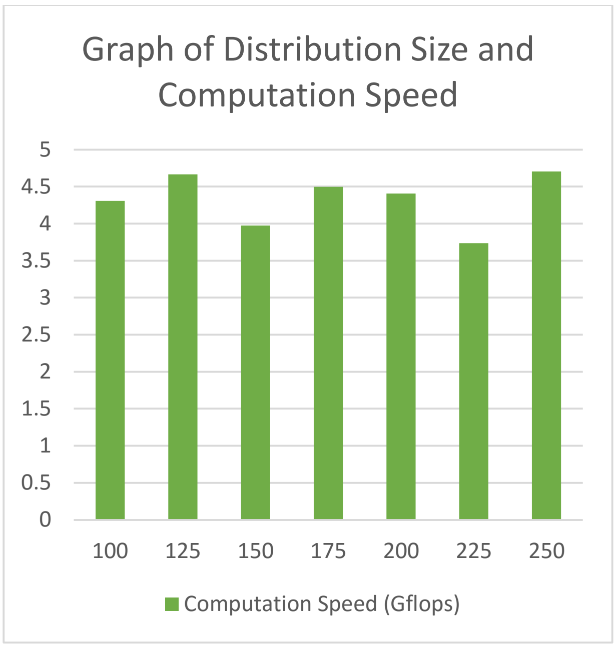


FIGURE 15. Distribution Size and Speed of Processing.

The distribution size dictates the block size of the problem to be decomposed and distributed among the nodes. The optimal distribution sizes will vary depending on computational performance and network configuration.

We have tested distribution sizes ranging from 100 to 250 in increments of 25, and the results from the test, as shown in Figure 15, found that block size 250 provides the best performance in terms of computation speed. A bigger block size means less messages to be sent over the network, hence less communication delay.

4.1.4 Panel Factorization, PFACT

The purpose of the fourth testing is to determine how the selection of panel factorization algorithm, PFACT and recursive panel factorization, RFACT influence the speed of processing and the time taken to solve the problem. The other parameters to be tested are fixed. The results of the test are as follows:

TABLE 7. Panel Factorization and Speed of Processing.

PFACT	Time Taken (s)	Speed of Processing (Gflops)
Left	18.33	4.549
Crout	17.20	4.847
Right	18.19	4.583

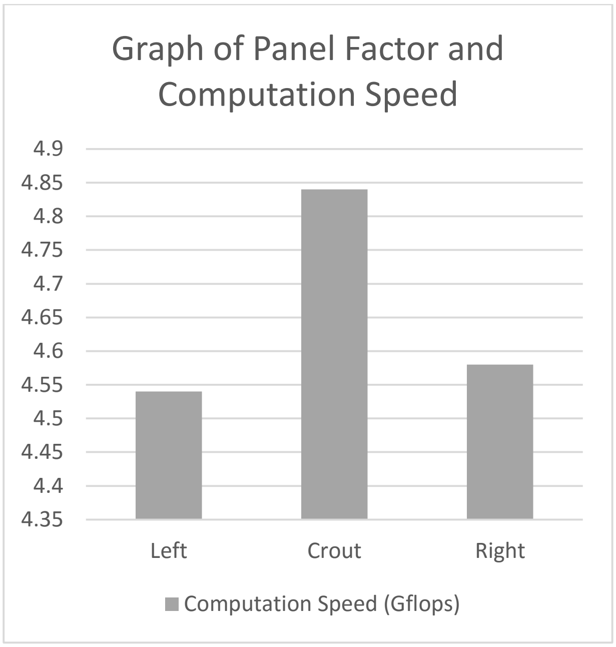


FIGURE 16. Panel Factorization and Speed of Processing.

Panel factorization dictates the type of panel factorization algorithm to be employed in solving the matrix decomposition problem. There are three types of algorithm: the left looking, right looking and Crout algorithms, which are three different ways of solving the computation problem.

The results of the test, as shown in Figure 16, indicate that the Crout algorithm in panel factorization increases the computation performance of the cluster as compared to other algorithm.

4.1.5 Recursive Panel Factorization, RFACT

The purpose of the fifth testing is to determine how the selection of recursive panel factorization, RFACT influence the speed of processing and the time taken to solve the problem. The other parameters to be tested are fixed. The results of the test are as follows:

TABLE 8. Recursive Panel Factorization and Speed of Processing.

RFACT	Time Taken (s)	Speed of Processing (Gflops)
Left	18.46	4.516
Crout	17.93	4.649
Right	19.97	4.175

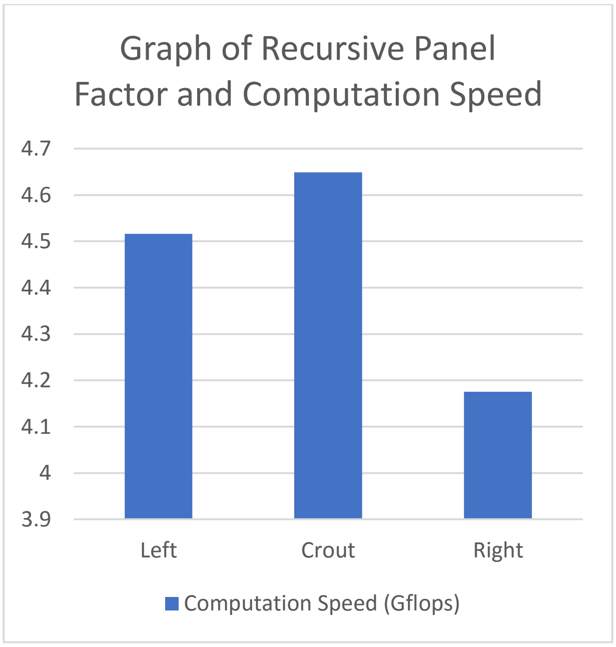


FIGURE 17. Recursive Panel Factorization and Speed of Processing.

Recursive panel factorization dictates the type of recursive panel factorization algorithm to be employed in solving the matrix decomposition problem. There are three types of algorithm: the left looking, right looking and Crout algorithms, which are three different ways of solving the computation problem.

The results of the test, as shown in Figure 17, indicate that the right looking algorithm in recursive panel factorization has the highest computation performance as compared to the other algorithms.

4.1.6 Broadcast Parameter, BCAST

The purpose of the sixth testing is to determine how the selection of broadcast algorithm, BCAST influence the speed of processing and the time taken to solve the problem. The other parameters to be tested are fixed. The results of the test are as follows:

TABLE 9. Broadcast Parameter and Speed of Processing.

BCAST	Time Taken (s)	Speed of Processing (Gflops)
1 ring	17.58	4.783
1 ring (modified)	18.35	4.542
2 ring	16.78	4.967
2 ring (modified)	16.77	4.972
Long message	19.94	4.181
Long message (modified)	19.43	4.291

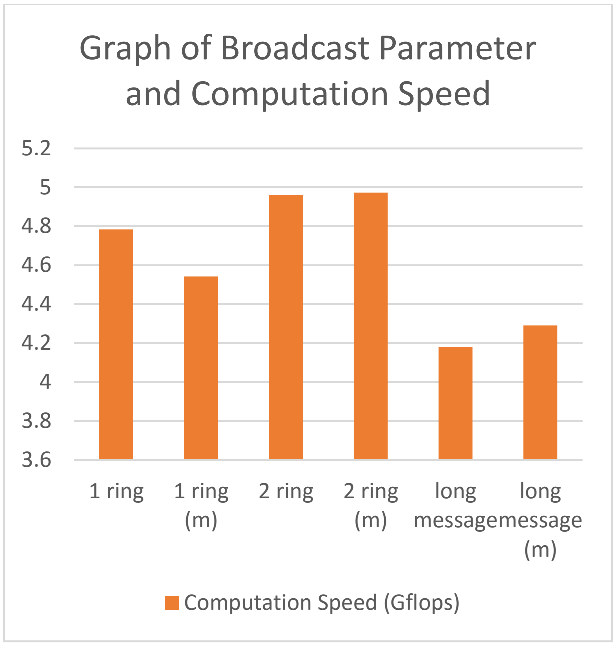


FIGURE 18. Broadcast Parameter and Speed of Processing.

The broadcast parameter dictates the type of panel broadcast algorithm to be utilised in distributing messages to other processes. Six types of broadcast algorithm are available for the user to select: the increasing-1-ring, the increasing-1-ring (modified), the increasing-2-ring, the increasing-2-ring (modified), long (bandwidth reducing) and long (bandwidth reducing modified) algorithms, in which are the varied ways of message exchange between the processes, which affects the time taken to process and also the computation speed, according to the results in Table 9.

The results of the test, as shown in Figure 18, confirms that the increasing-2-ring (modified) panel broadcast algorithm provides the best performance in terms of the computation performance of the cluster.

4.1.7 Look-ahead Depth, DEPTH

The purpose of the seventh testing is to determine how the selection of look-ahead depth, DEPTH influence the speed of processing and the time taken to solve the problem. The other parameters to be tested are fixed.

TABLE 10. Look-ahead Depth and Speed of Processing.

DEPTH	Time Taken (s)	Speed of Processing (Gflops)
0	19.10	4.365
1	16.21	5.143

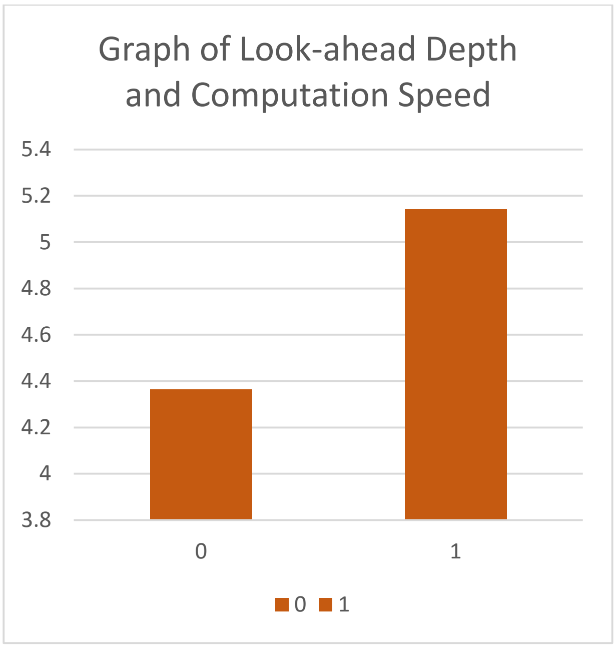


FIGURE 19. Look-ahead Depth and Speed of Processing.

Look-ahead depth dictates if the benchmark changes the order of the operations so that less efficient operations will run together will more efficient operations. If the look-ahead depth is greater than zero, the benchmark will “look ahead” by storing the panels being factorized in memory and uses up more memory in exchange for a better performance.

The results of the test, as according to Figure 19, indicate that when the look-ahead depth is one, the computation performance of the cluster is better than when there is zero look-ahead depth.

4.2 Final Test Run Results

The optimised parameters are configured on the HPL benchmarking tool based on the results from the test runs, which are as follows:

TABLE 11. Optimised Parameters for HPL based on HPC cluster

Parameters	Optimised Parameter Value
Number of Nodes, P x Q	10
Problem Size, N	125,000
Block Size, NB	250
Panel Fact, PFACT	Crout
Recursive Panel Fact, PFACT	Crout
Look-ahead Depth, DEPTH	1
Broadcast Parameter, BCAST	2-ring (modified)

For the problem size, the calculation is based on the recommended 80% of the square root of total memory size in bytes as written in the Literature Review. The formula of calculation is as follows:

$$N = \sqrt{\frac{Number\ of\ Gigabytes\ \times 1024^3 \times Number\ of\ Nodes}{8}} \times\ 0.8$$

$$= \sqrt{\frac{32\ \times 1024^3 \times 10}{8}} \times\ 0.8$$

$$\approx 165{,}800$$

However, when this problem size is tried in the HPC cluster, a segmentation fault error was given. Thus, the problem size is reduced to 60% to avoid the error, which is approximated to 125,000.

Another test will also be carried out using random and un-optimised parameters. The results of the test is as follows:

TABLE 12. Test Results with Optimised Parameters

Parameters	Time Taken (s)	Speed of Processing (Gflops)
Optimised	12830.65	23.78
Random	13978.47	21.16

.

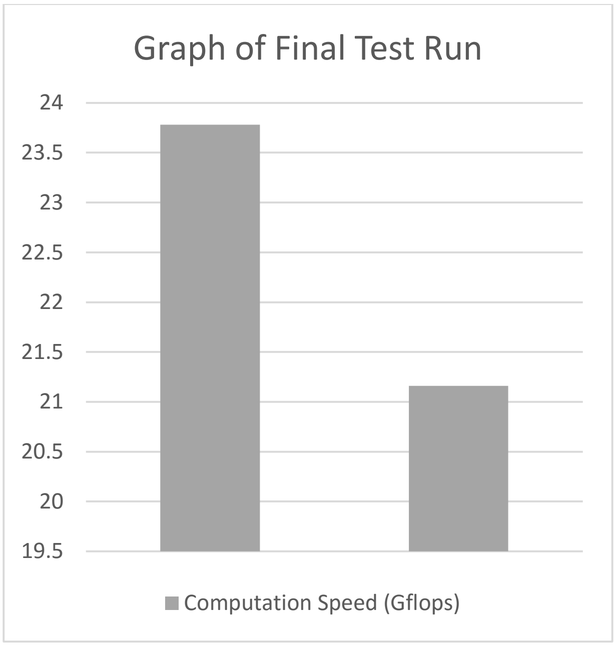


FIGURE 20. Test Results with Optimised Parameters.

As shown in Figure 20, the best results for using HPL benchmarking tool on HPC cluster is approximately 24 Gigaflops, while using random parameters, the computation speed is around 21 Gigaflops, a reduction of approximately 12 percent.

CHAPTER 5
CONCLUSION AND RECOMMENDATION

Through this research, the UTP HPC cluster has been benchmarked using the HPL benchmarking tool. The results from benchmarking also shows the peak performance achievable under the test conditions. The factors which can affect the implementation of HPL have also been discussed.

A few conclusions can be drawn from the findings obtained in this research. One is that a lot of factors and parameters need to be taken in account in running the HPL benchmark process tool. The kind of interconnection system employed, such as Gigabit Ethernet, InfinitiBand and Myrinet, can influence the effectiveness of the cluster and in turn the HPL benchmark result (Yeo et al., 2006). A better interconnection layout with higher bandwidth and lower latency will improve the maximum performance of the cluster.

Parameters of the HPL.dat and the type of BLAS library utilised can also affect the benchmark result (Imran et al., 2013). With different configurations of HPL parameters or even different BLAS libraries employed, a different result will be obtained.

For future work, a degree of optimisation should be employed for the HPL benchmark, by changing parameters off the benchmark for better results.

References:

Bader, D. A., & Pennington, R. (2001). "Cluster Computing: Applications''. *The International Journal of High Performance Computing, 15*(2), 181-185.

Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., . . . Schreiber, R. S. (1991). "The NAS parallel benchmarks". *International Journal of High Performance Computing Applications, 5*(3), 63-73.

Bakery, M., & Buyyaz, R. (1999). "Cluster computing at a glance". *High Performance Cluster Computing: Architectures and Systems, 1*, 3-47.

Barney, B. (2010). *"Introduction to parallel computing".* Lawrence Livermore National Laboratory. Retrieved from https://computing.llnl.gov/tutorials/parallel_comp/

Dongarra, J., Luszczek, P., & Petitet, A. (2001). "The LINPACK Benchmark: Past, Present and Future": University of Tennessee.

Dongarra, J. J. (1990). "The LINPACK benchmark: an explanation". In S. Aad van der (Ed.), *Evaluating supercomputers* (pp. 1-21): Chapman &amp; Hall, Ltd.

Dongarra, J. J., Bunch, J. R., Moler, C. B., & Stewart, G. W. (1979). *"LINPACK users' guide"* (Vol. 8): Siam.

El-Rewini, H., & Abd-El-Barr, M. (2005). *"Advanced Computer Architecture and Parallel Processing"*: John Wiley & Sons Inc.

Grün, T., & Hillebrand, M. A. (1998). *NAS Integer Sort on multi-threaded shared memory machines.* Paper presented at the Euro-Par'98 Parallel Processing.

Imran, M. J., Nor, A. W. A. H., & Othman, M. (2013). *"The High Performance Linpack (HPL) Benchmark on the Khaldun Sandbox Cluster".* Universiti Teknologi Mara, Universiti Teknologi Mara (Perak) Kampus Tengah, Perak Darul Ridzuan, Malaysia.

Microsoft. (n.d.-a). "Building and Measuring the Performance of Windows HPC Server 2008-Based Clusters for TOP 500 Runs". Retrieved 24 February, 2015, from http://go.microsoft.com/fwlink/?LinkId=134483

Microsoft. (n.d.-b). Load-Balanced Cluster. Retrieved 24 Feb, 2015, from https://msdn.microsoft.com/en-us/library/ff648960.aspx

Petitet, A., Whaley, R. C., Dongarra, J., & Cleary, A. (2004). "HPL-a portable implementation of the high-performance Linpack benchmark for distributed-memory computers". Retrieved 18 February, 2015, from www.netlib.org/benchmark/hpl/

Red Hat, I. (n.d.). "Cluster Basics". Retrieved 27 February, 2015 from https://www.centos.org/docs/5/html/Cluster_Suite_Overview/s1-clstr-basics-CSO.html

Rodgers, D. P. (1985). Improvements in multiprocessor system design. *SIGARCH Comput. Archit. News, 13*(3), 225-231. doi: 10.1145/327070.327215

Sterling, T. (2001). "An introduction to PC clusters for high performance computing". *International Journal of High Performance Computing Applications, 15*(2), 92-101.

Strohmaier, E., Dongarra, J. J., Simon, H., Meuer, M., & Meuer, H. (2015). The Linpack Benchmark, Top 500 Supercomputer Sites. Retrieved 18 February, 2015, from http://www.top500.org/

Yeo, C. S., Buyya, R., Pourreza, H., Eskicioglu, R., Graham, P., & Sommers, F. (2006). "Cluster computing: high-performance, high-availability, and high-throughput processing on a network of computers" *Handbook of nature-inspired and innovative computing* (pp. 521-551): Springer.

Appendices

Appendix A Steps Taken to Configure and Run HPL Benchmark in Linux

APPENDIX A

Steps to Take to Configure and Run HPL Benchmark in Linux

1. Making sure that necessary libraries/dependencies are installed, i.e. BLAS libraries (ATLAS/LAPACK/gotoBLAS etc), MPI (openMPI, mpich etc). Clusters with Intel processors can use a different BLAS called Intel Math Kernel Library.
2. Copy down the filepaths to the libraries of BLAS and MPI.
3. Create a home directory for HPL and goto directory.

   mkdir hpl

   cd hpl/
4. Download the source code for HPL. (Current version is 2.1)

   wget http://www.netlib.org/benchmark/hpl/hpl-2.1.tar.gz
5. Untar file.

   tar -zxvf hpl-2.1.tar.gz
6. Goto the new HPL directory

   cd hpl-2.1
7. There is a file named INSTALL, which contain instructions

   more INSTALL
8. Goto setup directory
   - cd setup
9. Create a generic template

   sh make_generic

   cp Make.UNKNOWN ../Make.Linux
10. Modify the makefile using favourite text editor (vi, emacs, nano etc)

    vi Make.Linux

    ARCH = Linux

    TOPdir = /home/(fill in user’s home directory)/hpl/hpl-2.1

    MPdir = /usr/local/mpi/mvapich/intel/1.1 (example filepath to MPI dir)

    MPlib = -L$(MPdir)/lib (example filepath to MPI library directory)

    MPinc = -L$(MPdir)/inc (example filepath to MPI include directory)

```
LAdir = /usr/local/atlas-lib/ (example filepath to BLAS directory)
LAlib = -L$(LAdir)/lib (example filepath to BLAS library directory)
```

11. If Infiniband is used, symbolic links need to be created or install "libibverbs-devel" and "libibumad-devel". Root may be required.
12. Build the makefile using make command

    ```
    make arch=Linux
    ```

13. If error encountered:
    - copy Make.Linux somewhere else

      ```
      cp Make.Linux /tmp
      ```

    - go up parent directory

      ```
      cd ..
      ```

    - delete hpl directory

      ```
      rm –rf hpl-2.1
      ```

    - untar file again (step 5)
    - goto directory (step 6)
    - copy Make.Linux back into directory

      ```
      cp /tmp/Make.Linux
      ```

    - clean before build

      ```
      make arch=Linux clean_arch_all
      ```

14. Build successful when xhpl binary is created.

    ```
    cd bin/Linux/
    ls
    ```

    (output) **HPL.dat xhpl**

15. Create a file containing all the names of the cluster nodes.

    ```
    vi allnodes (example of filename)
    compute-1 (insert name of node)
    compute-2
    …
    ```

    After all is finished, save and quit. Alternatively, use bash commands to create file.

16. Modify the desired parameters in HPL.dat (can check with online sources to see which to modify, such as HPL Tuning at netlib)

    ```
    vi HPL.dat
    ```

17. Create ssh-keygen to generate rsa key pair if nodes require ssh logins. Enter required passphrase to login.

    ```
    ssh-keygen
    ```

18. Copy the public key to remote host using ssh-copy-id. Do for every node.

    ```
    ssh-copy-id –i ~/.ssh/id_rsa.pub (name of node here)
    ```

19. Add ssh-agent to login only once to all the nodes.

    ```
    eval `ssh-agent`
    ssh-add
    ```

    Enter required passphrase.

20. Run the HPL benchmark using mpirun command.

    mpirun –np **80** –hostfile **allnodes** ./xhpl | tee HPL.out

    (highlighted: np refers to number of processor cores to utilise, hostfile refers to the filename of file containing the names of the nodes)