Jurnal Teknologi

Full Paper

THE HIGH PERFORMANCE LINPACK (HPL) BENCHMARK EVALUATION ON UTP HIGH PERFORMANCE CLUSTER COMPUTING

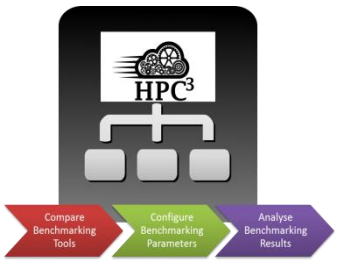
Wong Chun Shiang, Izzatdin Abdul Aziz*, Nazleeni Samiha Haron, Jafreezal Jaafar, Norzatul Natrah Ismail, Mazlina Mehat

Computer and Information Sciences Department, Universiti Teknologi PETRONAS, Bandar Seri Iskandar, 32610 Tronoh, Perak Darul Ridzuan, Malaysia

Article history
Received
1 February 2015
Received in revised form
24 March 2015
Accepted
1 August 2015

*Corresponding author
izzatdin@petronas.com.my

Graphical abstract



Abstract

Universiti Teknologi PETRONAS (UTP) has formed a high performance computing cluster (HPCC) inside the campus by utilizing most of the campus' existing resources as the computing resources. All the resources are interconnected within a same network, enabling it to be accessed by users anywhere anytime within and outside the campus. The UTP HPCC is used by researchers, including lecturers and students from UTP also from external parties to compute intensive applications. However, the UTP HPCC has never been benchmarked before, hence the performance and the ability of UTP HPCC are still unverified. The benchmarking is imperative in order to measure the true performance and the ability of UTP HPCC. This paper aims to evaluate the performance of UTP HPCC using a suitable benchmarking tool as well as to determine the optimal parameters configuration of the selected benchmarking tool. A comparative study has been done in order to select the best benchmarking tool between High performance LINPACK (HPL) and the NAS Parallel. The results show that HPL is a more suitable benchmarking tool for UTP HPCC compared to NAS Parallel. A series of experiments were carried out to select the optimum parameter of HPL configuration for UTP HPCC. The results from benchmarking show that the peak performance is achievable under the test conditions.

Keywords: Benchmarking, cluster computing, high performance LINPAC (HPL), NAS parallel

1.0 INTRODUCTION

The high performance computing (HPC) cluster is made of separate machines or servers interconnected and running intensive applications using parallel processing. By using the parallel processing, it manages to run the application efficiently, reliably and quickly. This HPC term applies especially to the system that the performance is above teraflop or 10^{12} floating-point operations per second. Mostly, the HPC users are coming from academician and scientific background. In some country they even used and rely on HPC to run some complex application for their government agency, particularly in the military.

Nowadays, there are a number of factors which result in the ubiquity of computer clusters such as the availability of high speed, computer interconnections, the reduction in the cost of components such as microprocessors, and the emergence of parallel programs or software where distributed computers can work together. So that, the benchmarking is needed in order to show the theoretical maximum performance and calculation ability of a computing cluster.

There are some different type of benchmark, which are, real program, kernel, I/O, and parallel benchmark. There are also several groups of benchmarking tools, like open source benchmarks also Microsoft benchmarks. In this paper, two of the

open source benchmarking tools are evaluated and one of it will be selected as our benchmarking tools.

The UTP's High Performance Computing Cluster (UTP HPCC) has never been benchmarked before, so that the performance and the ability of UTP HPCC are unsure. As such, this research paper aims to study the use of benchmarking software in measuring the performance of a computing cluster and implement a benchmark test on a small scale computing cluster such as UTP HPCC using the appropriate benchmark tool.

Therefore, the main objectives of this paper are to study on gauging the performance of a computing cluster through the use of benchmarking tool and also to carry out the benchmarking testing tool in a small scale computing cluster such as UTP's High Performance Computing Cluster.

The paper presents the benchmarking results and is structured as follows: Section 2 presents the brief explanations on the computer cluster and the benchmarking tools. Section 3 presents the proposed design and methodology of the benchmarking process. Section 4 contains the results and explanation of our findings. Finally, Section 5 concludes this paper and presents a brief outline of the research perspectives.

2.0 LITERATURE REVEW

2.1 Computer 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 [1]. 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 [2]. A cluster is normally enclosed within a room and handles as a single computer system [3]. 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 [3], with each node running its own instance of an operating system [1]. 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 [2].

The UTP HPC cluster for the UTP campus comprises of 60 cluster nodes. Each of the nodes contains 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 hardware specification of the Cloud Nodes is as follows: AMD FX 3.1 GHz processor, 32GB DDR3 RAM, AMD 7970 graphics card and Ethernet interconnection. The suite of software running on the nodes includes 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).

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 [4]. 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 synchronization of a central mechanism [4]. 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 [5]. 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:

$$\mathbf{T}(\boldsymbol{n}) = \mathbf{T}(\mathbf{1})\left(\boldsymbol{\beta} + \frac{\mathbf{1}}{\boldsymbol{n}}(\mathbf{1}-\boldsymbol{\beta})\right)$$

And the improvement in computation time, also known as speedup, S(n) is calculated by [5]:

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

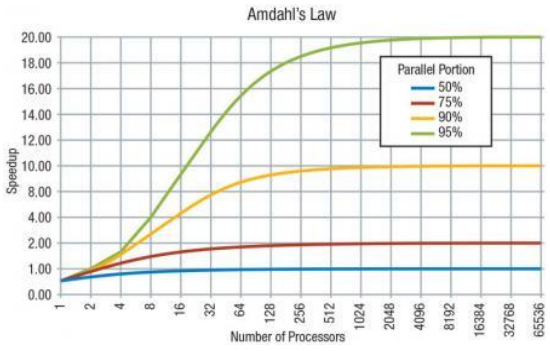


Figure 1 Graph of Amdahl's Law

Figure 1 illustrates the improvement in speed (or speedup) of a process relative to the percentage of portion in the process that can run in parallel. The speedup in processing time is significantly increase with the percentage of parallel portion in the process increases along with the number of processors. 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.

2.1 Benchmarking

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 can also show how different configurations may affect the performance of the computation.

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

2.1.1 High Performance LINPAC (HPL)

The HPL benchmarking tool is a portable application which works across various platforms and is written in C [6]. The LINPAC benchmark was primarily an auxiliary program which is developed from the LINPAC package. The LINPAC 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 [7].

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:

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

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) [8].

The decomposition of the dense matrix A is then commenced and the final outcome of the calculation is 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 unpiloted part of the matrix is the lower triangular matrix *L* and is not returned to the calculation [9]. 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 width and height of the matrix [9].

The data is distributed onto a two-dimensional P-by-Q grid of processes according to the block-cyclic scheme to ensure "good" load balance as well as the scalability of the algorithm. The n-by-n+1 coefficient matrix is first logically partitioned into NB-by-NB blocks, that are cyclically "dealt" onto the P-by-Q process grid. This is done in both dimensions of the matrix [9].

The right-looking variant has been chosen for the main loop of the LU factorization. This is mean that each of iteration of the loop a panel of NB columns is factorized, and the trailing sub matrix is updated. Note that this computation is thus logically partitioned with the same block size NB that was used for the data distribution [9].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 sub matrix is applied. The identical block size NB that was intended for distribution of data is used to logically divide the computation into partitions [9].

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 factorize. 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) [10]. 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 [9].

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 sub matrix 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 [9].

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 look ahead in normal situations, which means the panel presently being broadcast will affect the following sub matrix. 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 [9].

2.1.2 NAS Parallel Benchmark

The NAS Parallel Benchmark (NPB) suite is a set of computer programs. The benchmark is designed for testing parallel computer clusters in order to gauge the performance of parallel computer clusters [11]. 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 LINPAC benchmark and the NAS Kernels are more suited to evaluate vector supercomputers, and not the highly parallel machines popularly used nowadays [11].

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 in the Table 1.

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.

Table 1 Operations of NAS Modules

Benchmark Module	Operations
Multi Grid	Employs the V-cycle multi grid 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 sub procedure 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 [12].
Embarrassingly Parallel	Employing the Marsaglia polar process to produce independent Gaussian random variates.
Block Tri-diagonal, Scalar Penta-diagonal, Lower-Upper symmetric Gauss-Seidel	Find the answer to a nonlinear PDEs synthetic system using three different processes of calculations involving block tri-diagonal, scalar penta-diagonal 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.

Table 2 Comparative Study between HPL and NAS Benchmarks

Categories	Types of Benchmark	
	High Performance LINPAC (HPL)	***NAS Parallel Benchmarks***
Problem solving	Solves a dense matrix problem using LU factorization	Solves calculations involving simulations.
Modules	Single (LINPAC)	Multiple (CG, MG, FT, IS, EP, BT, SP, LU, UA, DC, DT)
Suited for Parallel Computation	Designed for Parallel Computers	Designed for Parallel Computers
Used widely as standard	Yes (TOP 500 list)	No

3.0 METHODOLOGY

In the first part of methodology, the problem is defined and clearly detailed. 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. One of the benchmark configurations needed in running the benchmark is to setup the High Performance LINPAC 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 sub procedures 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 utilized in creating software tools which need to perform linear algebra, such as LINPAC and LAPACK, as they have high efficiency, are transferrable across platforms and have many open source implementations [13].

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 [13]. The LINPAC 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 [13]. 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 [13]. 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 [13]. The ATLAS library will be used as BLAS library for the HPL benchmark.

In the following phase, the parameters to HPL.dat are 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 [9]. 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 utilized, with the remaining 20% left for other uses.

The processor grid (P x Q) is a 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, 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 [10]. This finding is a little contrasting with the recommended processor grid configuration, and will need to be checked out. 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 [9], 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 [9]. Research has suggested that the Modified Increasing-ring algorithm is one of the best in efficiency [14]. 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 [10]. These findings will be tested when benchmarking the UTP HPC cluster.

4.0 RESULTS AND DISCUSSION

Seven test runs are conducted on the HPC cluster based on the seven main parameters to be configured in the HPL benchmark. The results of the test to determine how the number of nodes, P x Q, influence the speed of processing and the time taken are as shown in Table 3.

Table 3 Results of Test based on Number of Nodes

Number of Nodes	Time Taken (s)	Computation Speed (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

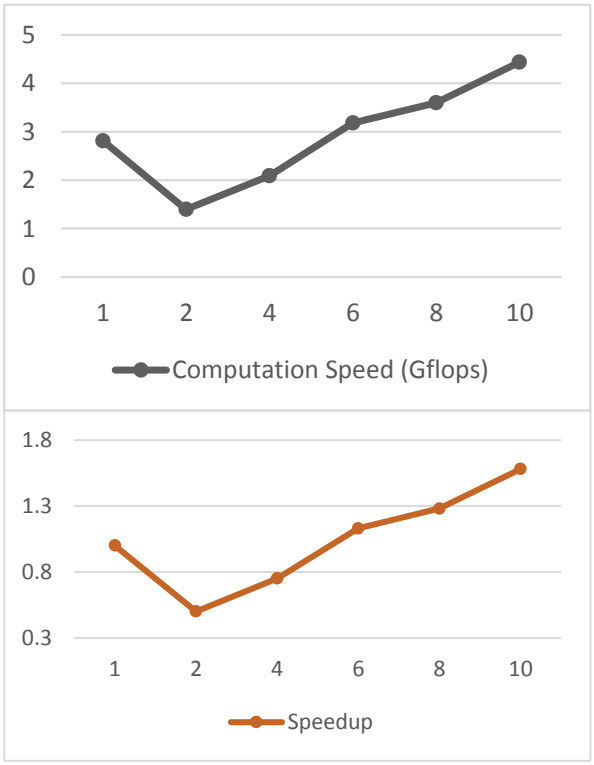


Figure 2 Computation Speed and Speedup Results (based on number of nodes)

As the number of nodes increases from one node to two nodes, the computation speed drops as shown in Figure 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 2. 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.

The results of the test to determine how the problem size, N, influence the speed of processing and the time taken are as shown in Table 4.

Table 4 Results of Test based on Problem Size

Problem Size, N	Time Taken (s)	Computation Speed (Gflops)	Speed up
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

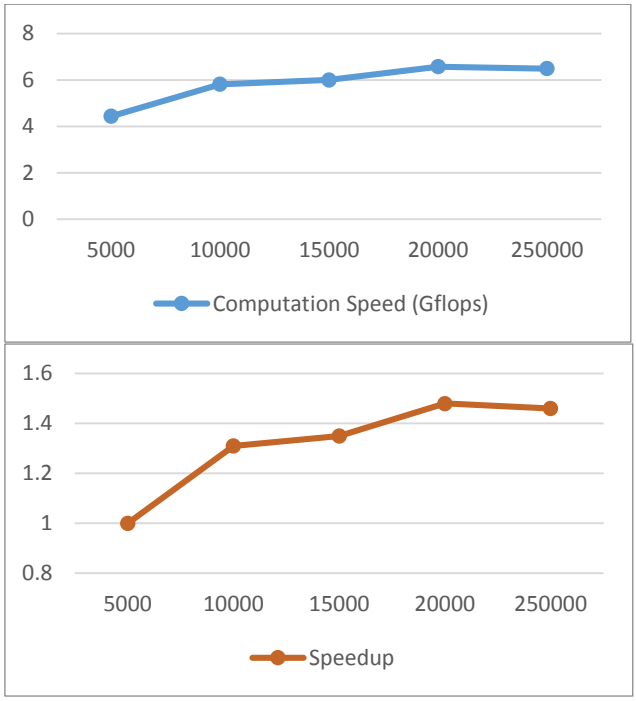


Figure 3 Computation Speed and Speedup Results (based on problem size)

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 Figure 3. However the increase in computational 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 stabilizes. This is due to the maximum effectiveness and efficiency of processing power had been reached.

The results of the test to determine how the distribution size, NB, influence the speed of processing and the time taken are as shown in Table 5.

Table 5 Results of Test based on Distribution Size

Distribution Size, NB	Time Taken (s)	Computation Speed (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

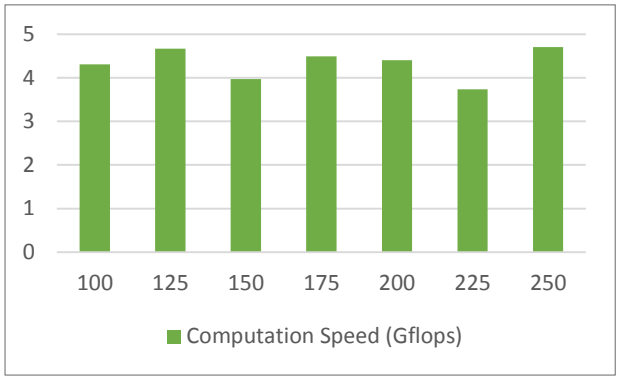


Figure 4 Computation Speed Results (based on distribution size)

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 found that block size 250 provides the best performance in terms of computation speed. A bigger block size means fewer messages to be sent over the network, hence less communication delay.

The results of the test to determine how the panel factorization, PFACT, influence the speed of processing and the time taken are as shown in Table 6.

Table 6 Results of Test (Panel Factorization)

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

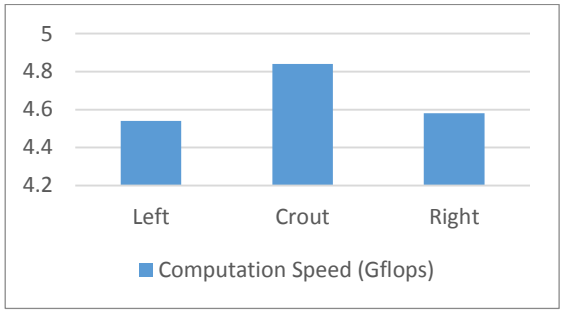


Figure 5 Computation Speed Results (based on panel factorization)

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 5, indicate that the Crout algorithm in panel factorization increases the computation performance of the cluster as compared to other algorithms.

The results of the test to determine how the recursive panel factorization, RFACT, influence the speed of processing and the time taken are as shown in Table 7.

Table 7 Results of Test (Recursive Panel Fact.)

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

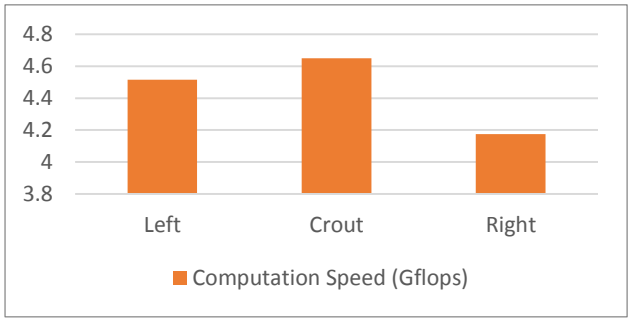


Figure 6 Computation Speed Results (based on recursive panel factorization)

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 computational problem. The results of the test, as shown in Figure 6, indicate that the right looking algorithm in recursive panel factorization has the highest computation performance as compared to the other algorithms.

The results of the test to determine how broadcast algorithms, BCAST, influence the speed of processing and the time taken are as shown in Table 8.

Table 8 Results of Test (Broadcast)

BCAST	Time (s)	Speed (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

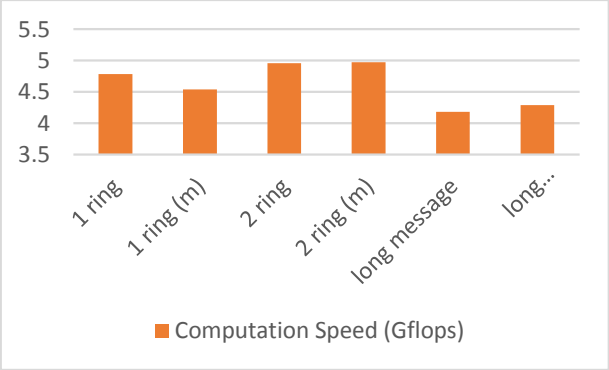


Figure 7 Computation Speed Results (based on broadcast algorithms)

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 Figure 7. The results of the test, as shown in Figure 8, confirms that the increasing-2-ring (modified) panel broadcast algorithm provides the best performance in terms of the computational performance of the cluster.

The results of the test to determine how look-ahead depth, DEPTH, influence the speed of processing and the time taken are as shown in Table 9.

Table 9 Results of Test (Based on Depth)

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

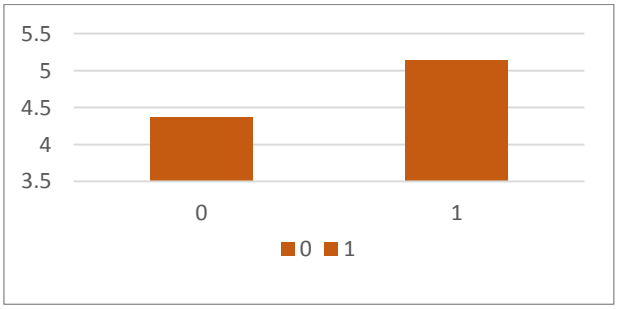


Figure 8 Computation Speed Results (based on look-ahead depth)

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 9, 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.

Based on the results, these are the optimized parameters:

Table 10 Optimised HPL Parameters

Parameters	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)

The results of the final test run are as follows:

Table 11 Results of the Final Test Run

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

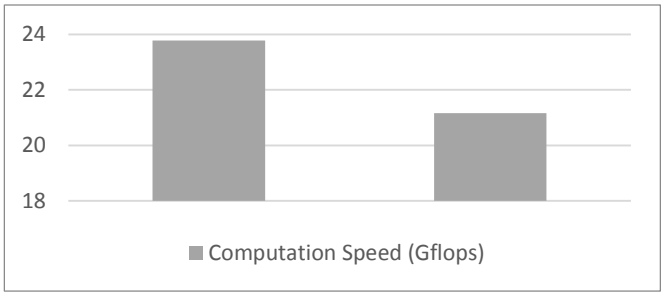


Figure 9 Final Test Results

As shown in Figure 9 and Table 11, 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. This shows that having optimized parameters can increase the performance of the cluster.

5.0 CONCLUSION

Through this research, the UTP HPC cluster has been benchmarked using the HPL benchmarking tool. The results from benchmarking also show 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, Infiniti Band and Myrinet, can influence the effectiveness of the cluster and in turn the HPL benchmark result [15]. 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 [10]. 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

[1] El-Rewini, H. and Abd-El-Barr, M. 2005. *Advanced Computer Architecture and Parallel Processing.* John Wiley & Sons Inc.

[2] Sterling, T. 2001. An Introduction To PC Clusters For High Performance Computing. *International Journal of High Performance Computing Applications*. 15(2): 92-101.

[3] Bakery, M. and Buyyaz, R. 1999. Cluster Computing At A Glance. *High Performance Cluster Computing: Architectures and Systems*. 1: 3-47.

[4] Barney, B. 2012. Introduction To Parallel Computing, Lawrence Livermore National Laboratory.

[5] Rodgers, D. P. 1985, June. Improvements In Multiprocessor System Design. In *ACM SIGARCH Computer Architecture News*). IEEE Computer Society Press. 13(3): 225-231.

[6] Dongarra, J. J., Bunch, J. R., Moler, C. B. and Stewart, G. W. 1979. *LINPACK Users' Guide*. Siam.

[7] Dongarra, J. J. 1988. The LINPACK Benchmark: An Explanation. *Supercomputing*. Springer Berlin Heidelberg. 456-474.

[8] Dongarra, J. J., Luszczek, P. and Petitet, A. 2003. The LINPACK Benchmark: Past, Present And Future. *Concurrency And Computation: Practice And Experience*. 15(9): 803-820.

[9] Petitet, A., Whaley, R. C., Dongarra, J. and Cleary, A. 2005. HPL–A Portable Implementation Of The High–Performance Linpack Benchmark For Distributed–Memory Computers, 2008. Available from Internet:< http://www. netlib. org/benchmark/hpl.

[10] Jelas, I.M., Hamid, N.A.W.A. and Othman, M., 2013. The High Performance Linpack (HPL) Benchmark on the Khaldun Sandbox Cluster.

[11] Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S. and Simon, H. D. 1991. The NAS Parallel Benchmarks. *International Journal of High Performance Computing Applications*. 5(3): 63-73.

[12] Grün, T. and Hillebrand, M. A. 1998, September. NAS Integer Sort On Multi-Threaded Shared Memory Machines. In *Euro-Par'98 Parallel Processing*. Springer Berlin Heidelberg. 999-1009.

[13] Strohmaier, E., Meuer, H. W., Dongarra, J. and Simon, H. D. 2015. The TOP500 List and Progress in High-Performance Computing. *Computer*. (11): 42-49.

[14] Microsoft. (n.d., 24 February). *Building and Measuring the Performance of Windows HPC Server 2008-Based Clusters for TOP 500 Runs*. Available: http://go.microsoft.com/fwlink/?LinkId=134483.

[15] Yeo, C. S., Buyya, R., Pourreza, H., Eskicioglu, R., Graham, P. and 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*. Springer US. 521-551.