Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 70 (2015) 62 – 68

Procedia Computer Science

4th International Conference on Eco-friendly Computing and Communication Systems

A Beowulf Cluster for Teaching and Learning

Ahmad A. Datti[1], Hadiza A. Umar[1] and Jamil Galadanci[1]*

[1]*Department of Computer Science, Bayero University Kano, Nigeria*

Abstract

The use of commodity clusters in academic institutions as a cost effective solution for the study of Parallel & Distributed Computing is a well-accepted development since the success of the Beowulf Project at NASA. This paper aims explore the effects of parallel computing on some programs in a Linux based Beowulf Cluster. The research project analyses the performance of some selected parallel programs on the Cluster in an effort to provide a parallel computing system for the practical study of Parallel and Distributed computing. The process of assembling the cluster involves setting up a FastEthernet based LAN of five (5) system units and the installation of Ubuntu-server on them. Compilers were installed for program execution; MPICH for distributed processing; Secure-Shell (OpenSSH) for remote execution and Network File System (NFS) for file system sharing. For performance analysis, two sets of parallel programs were executed on the cluster with varying number of nodes and their respective performance documented. The first was a dense matrix-matrix multiplication program and the second was a program for finding the number of prime numbers in a given range. It was observed for both programs that the rate of increase of parallel speedup in these programs gets higher as the problem size increases (parallelism is more pronounced in larger problem sizes). It was also observed, in both programs, that for too small a problem size, parallelism comes with a penalty..

*Keywords:*Beowulf Cluster, Parallel Computing, Distributed Processing

* Corresponding author. Tel.: +234-803-599-3379.
E-mail address:aadatti.cs@buk.edu.ng

1. Introduction

The high cost of supercomputers has made them beyond the reach of universities and other academic institutions hence crippling the teaching and learning of Parallel and Distributed computing courses in the computer science curriculum. But thanks to the efforts of researchers at NASA Goddard space station, the Beowulf Cluster was born. A Beowulf cluster is a distributed computing system made of normal desktop computers that are connected via commodity network such as Ethernet and are being controlled by free and open source software like Linux. Since then academic institutions have been building such kinds of parallel computing systems both for teaching and research. It is within this context, that it has become imperative for universities to support the 'Parallel and Distributed Computing' courses in their Computer Science programmes with practical experiences on these affordable Beowulf Clusters as cheap alternatives to vendor custom-made Supercomputers in order to support teaching and spark interest in computational sciences.

2. Previous Works and Motivation

The lack of a parallel computing system that will enable the hands-on learning of the Parallel and DistributedComputing courses in both the Undergraduate and Postgraduate Computer Science programs in some universities isone of the detrimental factors hampering the effective teaching and learning of the courses since inception of theacademic programs.

This research was carried out to address this problem by assembling a simple 'Proof-of-Concept' Beowulf Cluster Prototype using only the available computing and networking resources at the Faculty of Computer Science and Information Technology, Bayero University Kano (FCSIT-BUK).

The first Beowulf cluster[1] (Becker & Sterling 1995) was assembled for scientific research at NASA. But the low price to performance ratio of such clusters has opened up a new means for academic institutions to build them for teaching purposes.

Adams and Vos [2] have documented a detailed experience of building a Beowulf cluster at Calvin College, USA. The cluster was meant for research on object-oriented parallel languages, recursive matrix algorithms, network protocol optimisation, graphics rendering, modelling of electron behaviour in high energy laser field, modelling of complex inorganic molecules and modelling the interactions of Saturn's ring and atmosphere. The theoretical peak performance of the Calvin College cluster was 17Gflops but team optimised and fine tuned the cluster and finally achieved an aggregate of 10.4Gflops.

Philip R. Prins[3] documented a hands-on approach in teaching concepts of parallelism using an inexpensive Beowulf Cluster. Other researchers that have worked on a similar project were Adams[4] and Brown[5].

Alfonso and Muttoni[6] observed that computing clusters were mainly based on UNIX workstations and Linux PCs but different implementations of message passing systems were made available also for Microsoft Windows recently. They tested the performance of two implementations of MPI for Windows platforms, and compared the results with those obtained from Linux systems. They gathered that Windows performs better than Linux (in terms of Mop/s) in the implementation of a cluster for scientific HPC but Linux offers more stability and simplicity of cluster management. For instance, to obtain the said data, more than six months of work were necessary to complete the executions under Windows, while to obtain all the data under Linux one week was sufficient and it never crashed.

Stavrakas et al [7] discussed the use of Beowulf clusters in higher education - a case study of the Department of Electronics Engineering of the Technological Educational Institute of Athens. They discussed their design methodologies, the performance measurements and the experiments. Their aim was to enable undergraduate and postgraduate students to study parallel computing.

Adams and Brom [8] developed Microwulf, a Beowulf cluster that cost just $2470 to build,but provides 26.25 GFlops of measured performance, making it the first Beowulf with a price/performance ratiobelow $100/GFlops (for double-precision operations), an attractive design for most computer science departments.

Ayanda and Adejumo [9] gave attention to commodity cluster computing in public research institutions in Nigeria by proposing a prototype of Beowulf cluster for Obafemi Awolowo University.

Georgi et al [10] observed that due to a steady increase in the complexity of parallel computer systems there is an industrial need for employees with skills in practical experiences in the design and administration of HPC systems in addition to the theoretical fundamentals. But practical approaches were lacking in current curricula. For this reason, the *Technische Universitat Dresden*, developed and introduced a course "Linux Cluster in Theory and Practice" into their Computer Science programme so as to provide background knowledge about the design and administration of large-scale parallel computer systems and the practical implementation on the available hardware. In their paper, they analyzed the current variety of courses in the area of parallel computing systems, described the structure and implementation of LCTP and provided conclusions and an outlook on possible further developments.

Czarnul [11] presented motivations and experiences from using the BeesyCluster middleware for teaching high performance computing at the *Gdansk University of Technology*. They pointed out features of BeesyCluster well suited for conducting courses which include: easy-to-use web interface for application development and running hiding queuing systems, publishing applications as services and running in a sandbox by novice users, team work and workflow management environments.

Frinkle and Morris [12] described an approach to designing and implementing a High Performance Computing class focused on creating competency in building, configuring, programming and benchmarking HPC clusters. By coordinating with campus services, they were able to avoid additional costs to the students or the university. Their students built three twelve-unit independently-operating clusters. For evaluation purposes, they illustrated through pre- and post-course surveys that students gained substantial knowledge in fundamental aspects of HPC through the hands-on approach of creating their own clusters.

3. System design and Implementation

The building of the B0 Beowulf cluster basically involved the following steps:

i. Physical packing of all nodes into a compact portable form. This was to minimize the space occupied by these kinds of clusters.
ii. Installation of Ubuntu Server 12.04.2 with having Linux 3.5 kernel at its core on allnodes,
iii. Networking of the all nodes using Fast Ethernet followed by proper configuration of the operating system,
iv. Installation and configuration of additional software packages:
    a. OPENSSH-SERVER and OPENSSH-Client. For Remote Connection to enable MPI run the program in multiple Nodes.
    b. NFS-Common.: To enable all compute nodes make use of shared programs and shared files without the need for duplication on all nodes.
v. Installation and configuration of C, C++ and Fortran Compilers for coding and compilation of programs.
vi. Installation and configuration of MPICH2 as a Message Passing Utility that enables distributed cluster computing.
vii. Testing the Cluster with simple MPI-Hello World Program
viii. Assessing the performance of a sample parallel program (matrix-matrix multiplication) on the Cluster.

4. Experimental Result

For the performance analysis, two sets of parallel programs were executed on the cluster with varying number of nodes and varying problem sizes and their respective performance documented.

The first is a dense matrix-matrix multiplication program. The dimensions of the matrix were 500X500, 1000X1000, 1500X1500, 2000X2000, 2500X2500, and 3000X3000. The execution time (for each matrix dimension) was observed for the program when executed in parallel on 5 Nodes, 4 Nodes, 3 Nodes and 2 Nodes and then serially on 1 Node.

The second is a program for finding the number of prime numbers in a given range. The ranges were four powers of 2 as follows: 214 (16,384), 215 (32,768), 216 (65,536) and finally 217 (131,072). The program was executed seriallyfor each range above and then in parallel on 5 nodes, 4 nodes, 3 nodes, and finally 2 nodes. Execution time was observed and reported below.**NB**: All timings were taken using the Linux time command

4.1. Matrix-Matrix Multiplication

Table 1:Average execution time in (seconds) for thedifferent matrix sizes.

	500X500	1000X1000	1500X1500	2000X2000	2500X2500	3000X3000
Serial	0.340	2.145	9.074	16.279	45.190	82.291
2 Nodes	1.235	4.526	12.191	26.523	51.387	86.900
3 Nodes	0.969	3.097	7.550	16.172	29.998	54.237
4 Nodes	0.876	2.599	6.030	13.091	21.613	36.962
5 Nodes	1.034	2.607	5.855	11.720	19.744	33.283

Table 1 above shows the speedup obtained from the matrix multiplication program while figure 1 is the graph of the parallel speedup against number ofprocessing elements for the different problem sizes.Note thatideal speedup is obtained when **Speedup = Number of Nodes**. When running an algorithm with linearspeedup, doubling the number of processors doubles the speed.

Table 2:Speedup for different matrix sizes

Nodes	Ideal	500X500	1000X1000	1500X1500	2000X2000	2500X2500	3000X3000
Serial	1	1	1	1	1	1	1
2 Nodes	2	0.275	0.474	0.744	0.614	0.879	0.947
3 Nodes	3	0.351	0.692	1.202	1.007	1.506	1.517
4 Nodes	4	0.388	0.825	1.505	1.244	2.091	2.226
5 Nodes	5	0.329	0.823	1.550	1.389	2.289	2.472

Figure 1: Graph depicting speedup for the different matrix sizes

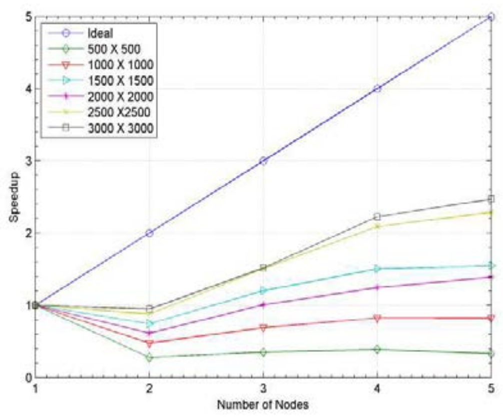
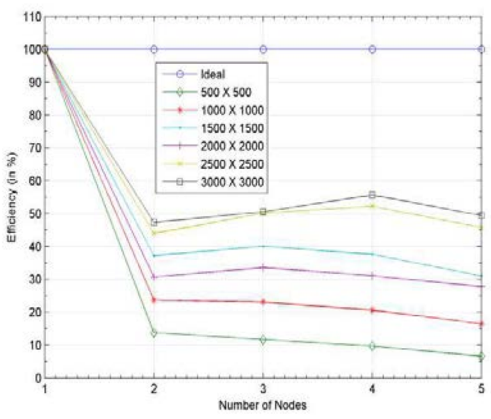


Figure 2: Graph depicting efficiency for the different matrix sizes



From the Speedup graph in Figure 1, it could be observed that problem sizes of 500x500 and 1000x1000 were found to perform slower when parallelized while those showing improvement due to parallelization were of 2000x2000, 1500x1500, 2500x2500 and 3000x3000. It can be generalized that the rate of increase of parallel speedup in this matrix-matrix multiplication program gets higher as the problem size increases. In plain words, improvement due to parallelism is more pronounced for larger problems.

It can also be observed that for all problem sizes, the speedup is below 'ideal'. In other words, speedup is sub linear. Texts on parallel computing have identified the following to be reasons for sub-linear speedup in parallel algorithms[13]:

i. Using more than one processor necessitates communication, which is overhead that was not part of the original serial computation.
ii. Secondly, if the processors do not have exactly the same amount of work to do, they may be idle part of thetime, lowering the attained speedup.
iii. Finally, program code may have sections that are inherently serial (the famous Amdahl's law).

4.2. Efficiency for the different Matrix Sizes

Table 3 is a graph of the efficiency against number of processing elements obtained for the different problem sizes.

NOTE: 100% Efficiency is achieved only when speedup is ideal.

Table 3: Efficiency (in percentage) for the different matrix sizes

P	Ideal	500X500	1000X1000	1500X1500	2000X2000	2500X2500	3000X3000
1	100.000	100.000	100.000	100.000	100.000	100.000	100.000
2	100.000	13.767	23.693	37.214	30.689	43.970	47.348
3	100.000	11.691	23.083	40.058	33.554	50.214	50.575
4	100.000	9.699	20.626	37.619	31.088	52.272	55.660
5	100.000	6.575	16.454	30.993	27.780	45.775	49.450

Figure 2 shows the parallel efficiency for all permutations of matrix sizes and number of nodes. It wasobserved from the graph of that the permutations of matrix dimension and number of nodes that exhibits the highestefficiency (49.5%) was found to be when multiplying a 3000X3000 matrix on 4 Nodes.

4.3. Prime number generation

Table 4: Average execution time (in secs) for prime generation

	131072	65536	32768	16384
Serial	12.768	3.470	0.961	0.305
2 Nodes	24.858	4.592	0.834	0.833
3 Nodes	16.226	1.976	1.088	1.129
4 Nodes	11.920	1.512	0.888	0.887
5 Nodes	6.738	1.411	0.939	0.945

Table 5: Speedup for the prime generation program ($T_{serial}/T_{parallel}$)

	Ideal	131072	65536	32768	16384
Serial	1	1	1	1	1
2 Nodes	2	0.514	0.756	1.153	0.366
3 Nodes	3	0.787	1.756	0.883	0.270
4 Nodes	4	1.071	2.295	1.082	0.344
5 Nodes	5	1.895	2.459	1.023	0.323

Table 6:Parallel efficiency table for the prime generation program

	Ideal	131072	65536	32768	16384
Serial	100	100	100	100	100
2 Nodes	100	25.682	37.787	57.630	18.307
3 Nodes	100	26.230	58.545	29.442	9.005
4 Nodes	100	26.780	57.370	27.050	8.598
5 Nodes	100	37.897	49.174	20.456	6.456

From figure 3, it could be observed that largest problem sizes exhibited a more normal speedup curve that progressively increases with increase in number of nodes while the two smaller problem sizes show curves that generally decrease with increase in number of nodes. This indicates that parallelism is more pronounced in larger problem sizes and that for too small a problem size, parallelism comes with a penalty. Secondly, it could be observed that all problem sizes exhibit a sub-linear speedup.

4.4. Efficiency for prime number generation

Table 6 is detailing the computed efficiency of the Prime generation program for all permutations ofnumber of nodes and range while figure 4 is a graph of the efficiency against number of nodes.

From figure 4, it could be seen that all efficiencies are below the 'perfect' 100% level and that the only problem size that shows a progressive increase in efficiency with increase in number of nodes is for 131,072 (217). The efficiency of all other problem sizes decreases with increase in number of nodes. Highest efficiency (58.5%) was achieved when finding primes between 2 and 65536 on 3 nodes.

5. Conclusions and recommendations

Parallel and Distributed computing is presently the solution to the ever increasing demand for High Performance Computing Systems. The study of this field in academic disciplines is only going to be complete and meaningful with practical heavy sessions where students will physically experience the concepts and ideas and will develop skills that will prepare them for careers in design and management of High Performance Computing systems.

Researchers are also going to benefit from a parallel computing system for running computation intensive simulations.

Beowulf Clusters are the cheapest kind of parallel computing systems because they can be built at minimal or zero cost especially at institutions where computers have been in use for long. It will also help institutions to recycle their out of used desktop computers.

5.1. The Cluster

This work has successfully built a low cost Beowulf cluster that can be used for the study of Parallel and Distributed Computing courses at both the undergraduate and postgraduate levels of computer science.

5.2. Performance of the program

i. From observations, it can be concluded that Parallelism is more pronounced for larger problem sizes and it comes with a penalty for problem sizes that are too small.
ii. It can also be observed that for all problem sizes, the speedup is sub linear.
iii. The size of the problem is a factor of the resource efficiency of the cluster.

For improvements regarding the performance of B0, interested parties should consider using a Gigabit switch instead of a Fast Ethernet switch. This is because the network of commodity clusters is always the bottleneck for achieving high performance regardless of the speed of the processor and size of memory.

Secondly, replacing the current nodes in this cluster with faster systems will also significantly improve on the raw performance of the whole cluster.

For performance analysis, the Linpack benchmark should be run so as to get the actual FLOPs of the cluster. This will provide a yardstick by which the cluster could be compared to other computing systems.

References

1. Becker, D. J., Sterling, T., Savarese, D., Dorband, J. E., Ranawak, U. A., & Parker, C. V. (1995). Beowulf: A Parallel Workstation for Scientific Computation. International Conference on Parallel Processing.
2. Adams, J., & Vos, D. (2002). Small-College Supercomputing: Building a Beowulf Cluster at a Comprehensive College. *ACM SIGCSE Bulletin, 34* (1), 411-415.
3. Prins, P. R. (2004). Teaching parallel computing using Beowulf clusters: a laboratory approach. Journal of Computing Sciences in Colleges, 20 (2), 55-61.
4. Adams, J. C. (2011, March). A cluster for CS education in the manycore era. Proceedings of the 42nd ACM technical symposium on Computer science education, 27-32.
5. Brown, R. G. (2007). Engineering a Beowulf-Style Compute Cluster (0.1 ed.). Durham, North Carolina, United States of America: Robert G. Brown - Duke University Physics Department.
6. Alfonso, G., & Muttoni, L. (2004). Performance Evaluation of NT based PC Cluster for high performance computing. *Journal of Systems Architecture, 50* (6), 345-359
7. Stavrakas, I. K. (2005). Beowulf Clusters for Parallel Programming Courses. The International Conference on Computer as a Tool. 1, pp. 791-794. IEEE.
8. Joel C. Adams and Tim H. Brom (2008). A Beowulf cluster for every desk. Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education
9. Ayanda, D. &. Adejumo (2011). A Prototype Model of High Performance Computing Using Beowulf Cluster. International Journal of Emerging Sciences, 1 (4), 696-705.
10. Georgi, A., Höhlig, S., Geyer, R., & Nagel, W. E. (2011). Linux cluster in theory and practice: A novel approach in teaching cluster computing based on the Intel atom platform. *Procedia Computer Science,* (pp. 1917-1926)
11. Czarnul, P. (2014). Teaching High Performance Computing Using BeesyCluster and Relevant Usage Statistics*. *Procedia Computer Science, 29,* pp. 1458-1467
12. Frinkle, K., & Morris, M. (2015). Frinkle, K., & Morris, M. (2015). Developing a Hands-On Course around Building and Testing High Performance Computing Clusters. *Procedia Computer Science,, 51,* pp. 1907-1916
13. Eijkhout, V., Chow, E., & Geijn, R. v. (2011). Introduction to High Performance Scientific Computing (1st Edition ed.). Texas: The Saylor Foundation.