Caspian Journal of Mathematical Sciences (CJMS)
University of Mazandaran, Iran
http://cjms.journals.umz.ac.ir
ISSN: 1735-0611

CJMS. **4**(1)(2015), 95-105

Parallel computing using MPI and OpenMP on self-configured platform, UMZHPC

V. Sabet [1] and A. Valinejad[2]
Department of Computer Science, University of Mazandaran,
P.O. Box 47416-95447, Babolsar, Iran

Abstract.

Parallel computing is a topic of interest for a broad scientific community since it facilitates many time-consuming algorithms in different application domains. In this paper, we introduce a novel platform for parallel computing by using MPI and OpenMP programming languages based on set of networked PCs. UMZHPC is a free Linux-based parallel computing infrastructure that has been developed to create rapid high-performance computing clusters. It can convert heterogeneous PCs which interconnected by using a private Local Area Network(LAN) into a high-performance computing cluster. In this operating system, you can monitor your cluster and build it utilizing low-cost hardware. In addition, programs can be run in parallel by simply booting the portable UMZHPC from fronted node by using only a CD or USB-flash drive. All the requisite configurations to build a cluster and to run your programs will be carried out automatically via UMZHPC. We made the operating system publicly for research purposes.

Keywords: Parallel computing, MPI, OpenMP, HPC.

2014 Mathematics subject classification: 65Y05, 68W10.

1. Introduction

Nowadays, parallel computing has become a significant topic of interest in complex multidisciplinary applications, in particular in computer

[1] v.sabet@stu.umz.ac.ir
[2] Corresponding author: valinejad@umz.ac.ir
Received: 21 July 2015
Revised: 23 November 2015
Accepted: 5 December 2015

sciences which enables us not only to accelerate the implementation of algorithms, but also reduce many compute-intensive processes in different sciences. Parallel computing refers to the concept of speeding up the execution of large programs by dividing them into smaller fragments that can run simultaneously [13]. UMZHPC which is the abbreviation of University of Mazandaran High Performance Computing, is a self-configured and bootable Beowulf-class [1] HPC system which provides parallel computing environment by utilizing inexpensive computational resources. It is a free GNU/Linux operating system which is originally based on Pelican HPC [4] source code that let us to set up a high-performance computing cluster by using MPI [15] and OpenMP [6] language-independent libraries. However, parallelize an algorithm requires a meticulous restructuring of existing program to achieve high-performance improvement over sequential version [5].

Building a Beowulf-class cluster requires advanced skills in operating systems and entails sophisticated software configuration which is a laborious process for beginners and non-computing-researchers.

The aims of this work is to simplify these processes by using automatic configuration and built-in softwares. UMZHPC is an improvement over Pelican HPC because it uses a monitoring system to examine cluster nodes in detail. It is customized to conventionally work with an heterogeneous set of computers and does not require to an even hard-disc drive or installed operating system because fronted node boots from a CD image or USB(Universal Serial Bus)-flash drive and computational nodes boot by PXE(Preboot Execution Environment) via LAN connection using the fronted node as a cluster server. Instead of installing on the hard-disk drive, UMZHPC operating system is installed in the RAM(Random Access Memory) which guaranteed that the installed operating system is not used. We use a tool called Ganglia [10] for monitoring our high-performance computing system which indicates what is the status of computational nodes in detail.

Due to the recent technological advances in the processors power of personal computers and network bandwidth, the propensity to parallel processing has changed from costly massively supercomputers to inexpensive cluster of PCs interconnected by using Ethernet LAN for parallel applications and large problems. Based on Live Linux operating system, we developed a special distribution which aims to focus on mathematical operations in parallel. Even though there were some research on the parallelism based on Linux operating system, the projects either ceased or the results were not desirable. For instance, KestrelHPC [9] is a set of tools to build a beowulf-class cluster based on a deb package. Chhabra et.al [3] presents a parallel computing framework which is based on the

Master-Slave computing paradigm and it emulates the parallel computing environment. birgHPC [2] is developed to create high-performance clusters for bioinformatics and molecular dynamics studies. Hai Jin .et.al [7] discussed the incentive for using clusters as well as the technologies available for building clusters. However, our system has ability to monitor cluster nodes and focused on mathematical operations. Since implementing matrix multiplication program is a computation-intensive application and can be effectively parallelized, we demonstrate the performance of our system achieved through parallel computing utilizing MPI to handle the problem. The rest of the paper is organized as follows: Section 2 describes speedup and efficiency which clarify how we can measure performance of programs. Cluster architecture and setup a cluster are in section 3. Section 4 describes parallel computing using UMZHPC . Next section 5 analyze results. Finally, section 6 concludes this paper.

2. Speedup and Parallel efficiency

We design parallel programs in the hope of speeding them up and they run faster in comparison with sequential execution. This can be done by dividing a program into multiple fragments and each one executes on its own processor concurrently. Hence the time taken to solve our problem can reduce drastically.

Definition 2.1. The parallel speedup is a ratio between sequential execution time and parallel execution time:

$$S_p = \frac{T_s}{T_p} \tag{2.1}$$

Which T_s is sequential execution time and T_p is parallel execution time.

Definition 2.2. The efficiency of a parallel program is obtained through dividing the speedup by the number of processors:

$$E_p = \frac{T_s}{P * T_p} \tag{2.2}$$

Which T_s is sequential execution time, P is number of processors and T_p is parallel execution time [8, 12]. By these definitions, we test our system to show how parallel execution over UMZHPC with different number of processors has an effect upon the speedup and efficiency of a program.

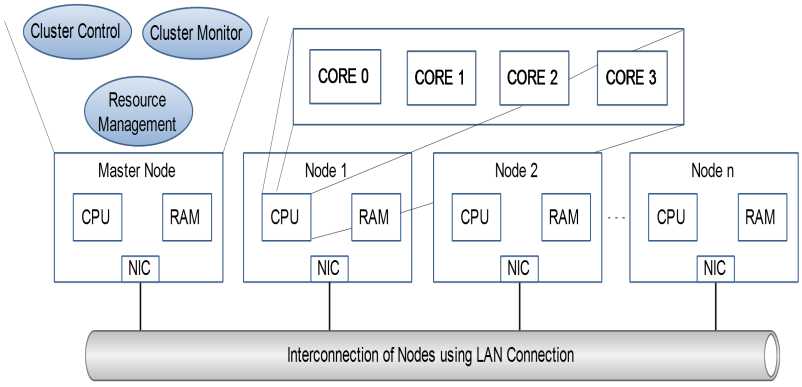

FIGURE 1. Architecture of UMZHPC. Master node control the whole cluster.

3. System design

UMZHPC is designed to facilitate parallel programing and to achieve high-performance computing. In this system, master node is developed in order to conduct entire cluster management and control connection among slaves. At first, it prepares some requisite services for the cluster and then scans the network and identifies available slaves. In the next stage, a evaluation script will be accomplished which indicates the approximate power of the cluster. Finally, your cluster is ready to implement parallel programs written in MPI and OpenMP.
On the other hand, slave node is developed to receive the problem or sub-problem from the master node and then computes its proportion of the problem. When the slave node prepares the computed sub-problem, it will send the result back to the master node and as a consequence, it shows the outcome.

UMZHPC Cluster architecture and setup: The main feature of UMZHPC is its capability to convert PCs interconnected using Ethernet LAN into a high-performance computing cluster. PCs in computer laboratories are often worthless after work-hours and during non-work days. Such resources can offer a significant computational power and can be established as a HPC cluster within minutes by using UMZHPC. To setup the cluster, one of the computers must be master and other computers will be slaves as the computational nodes. If some nodes have multi-core CPUs, each core in the CPUs can act as a slave node in the cluster. The computer that act as the master node first boot from the CD or USB-flash drive to manage other nodes and then the computing nodes will be boot via LAN connection. Figure 1 shows the architecture of UMZHPC.

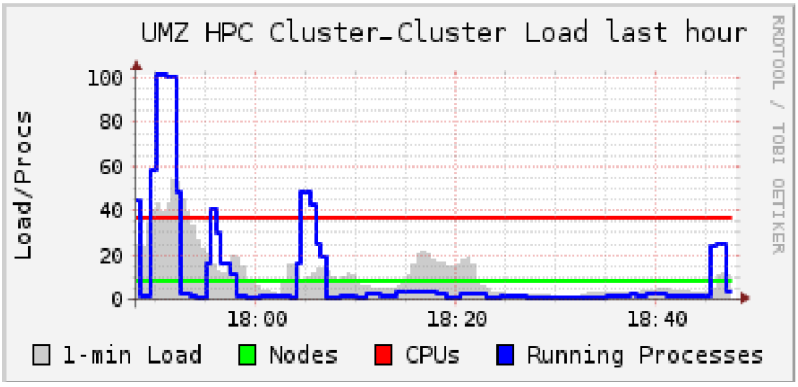

FIGURE 2. cluster usage status using Ganglia monitoring system.

The computing nodes have to support PXE (Preboot Execution Environment) boot feature and if any of them does not support PXE feature, a specify implementation of PXE called gPXE [16] will be useful. At first, the umz_setup script is used to configure the cluster. The umz_restart_hpc will be run when additional nodes are added or removed from the existing cluster. All settings which are needed for running MPI and OpenMP programs is configured automatically via UMZHPC. Status of the cluster can be monitored utilizing Ganglia monitoring system (Figure 2).

After configuration of cluster, a directory will be share among the nodes by using NFS (Network File System) protocol to run jobs. Users must put their files in /home/user directory located in the master node and in order to run programs on the cluster nodes, you can refer to the nodes' list in /home/users/tmp/bhosts file which consists of cluster node IPs and number of available slots per node.
Since UMZHPC is loaded into RAM, all the user files in the shared directory will be lost when fronted node is rebooted. You can mount a peripheral storage device to save your files or as a second solution, you can connect the master node to the internet via a subsidiary network card and send the result to your e-mail.

4. Parallel computing using UMZHPC

To evaluate UMZHPC performance, we chose matrix multiplication program as a sample problem which has great capability for parallelism and can be divided into sub-problems and each sub-problem can be computed simultaneously. For simplicity, we assume that the matrices used

in multiplication are square in shape. We used C++ programming language for the implementation since this language use row-major placement approach [14], we further assume that three matrices are stored in row-major order, as shown in Figure 3. Consider two n*n matrices A and B which their multiplication will generate matrix C. The master node divides matrix A into set of rows depending upon the number of available slave nodes and send each row proportion to the slaves. In addition, the entire matrix B is sent to all slave nodes. Each slave node computes its proportion and sends back a set of rows of resultant matrix to the master. Then the master node gathers the result's portions from each slave node and put them into pertinent order in resultant matrix. Figure 4 illustrates the parallel Matrix multiplication procedure.

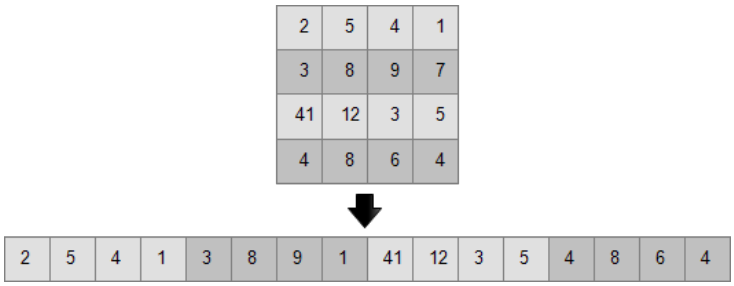

FIGURE 3. Layout of a 4*4 matrix in row major order.

5. Results and Discussion

In this section, we present experimental results of parallel computing using UMZHPC. We analyzed the performance of parallel method against serial method of matrix multiplication problem. The test performed on the following system configuration for all slave nodes:

- Processor: Intel Core i5-4460 (3.20 GHz)
- RAM: 4GB
- Network: LAN connection using 1000 Mb FULL DUPLEX switch

Considering these configuration, calculation of the problem performed on 7 PCs and overall, 25 cores. Figure 5 illustrates workload status per slave node.

As can be observed in Figure 5, The master node excluded from the problem calculation; because we have empirically discovered that when we use the master node for solving problems beside controlling the cluster, the calculation process takes more time in comparison with when master node is omitted. As a result, the overall performance will drop considerably. Table 1 indicates calculation time to solve the problem by

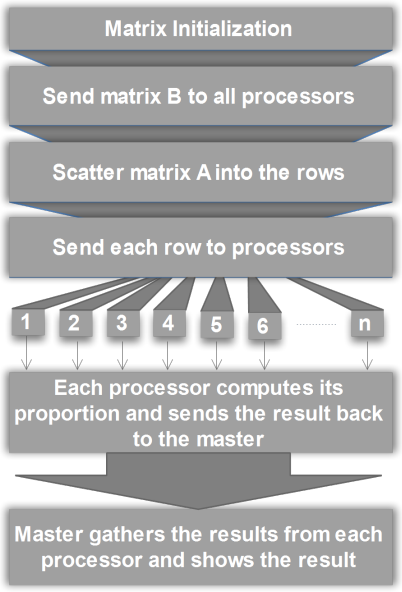

FIGURE 4. Parallel Matrix Multiplication procedure

using different number of processors. The performance of the problem has compared to different number of processors in the cluster and the speedup and efficiency, which are illustrated in Figures 6 and 7 respectively, have calculated according to aforementioned formulas in section 2. Analyzing the execution of matrix multiplication problem on UMZHPC reveals a clear fact that the more computational nodes exist in your cluster, the better speed and performance will be achieved.

Generally program acceleration is our anticipation of parallel execution. For instance, when we use n CPUs in our cluster, we expect that our program speedup being n times faster than serial execution. However, the result of parallel execution of matrix multiplication problem with 1000 elements in each row and column with 25 processors shown in the Table 1 depicts that the speedup is about 18 times faster. It is because of the communication latency and overhead [11] that prevent us from a perfect achievement.

6. CONCLUSION

This article demonstrates a platform which enables us to facilitate parallel programing and to gain high-performance computing achieved through parallel computing. UMZHPC is an operating system based on

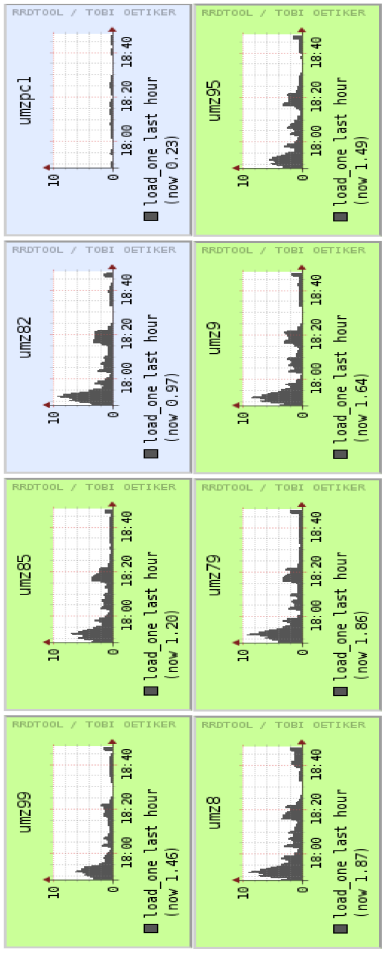

FIGURE 5. cluster's nodes load status. umzpc1 is the master node which has excluded from the computation.

Linux Debian Live CD which originally derived from PelicanHPC source code. It is used to convert PCs interconnected via LAN connection into a high-performance computing cluster which aims to simplify building a Beowulf-class cluster within minutes. All the requisite configurations to build a cluster and to run your programs will be carried out automatically via UMZHPC. This system allows researchers to implement their parallel algorithms utilizing MPI and OpenMP programming languages and has capability for monitoring systems on the cluster in detail. Even though UMZHPC is a platform to speed up computation-intensive programs, programming skills and level of expertise are not trivial to

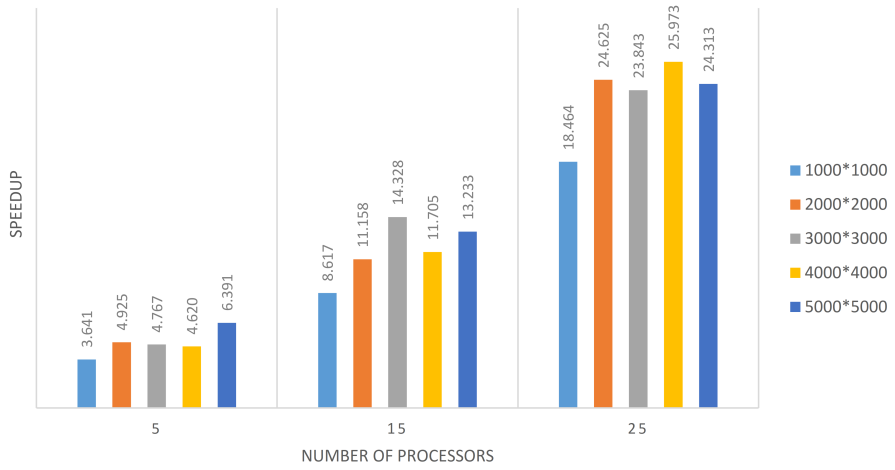

FIGURE 6. Parallel speedup of matrix multiplication problem with different number of elements and processors over UMZHPC

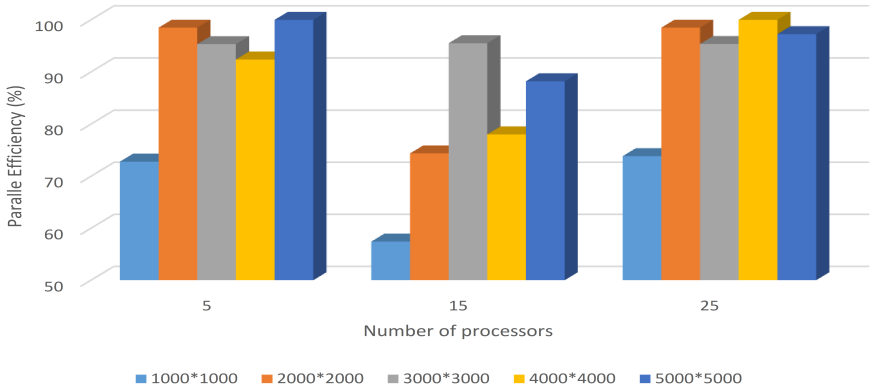

FIGURE 7. The efficiency of parallel execution of matrix multiplication problem with different number of elements and processors over UMZHPC

achieve desirable performance. We made the operating system publicly for research and academic purposes.

7. AVAILABILITY

Project name: UMZHPC
Project homepage: http://hpc.math.umz.ac.ir
Operating system: self-determining Platform

TABLE 1. Execution time of Matrix Multiplication problem on the UMZHPC in seconds

Type of execution	Number of cores	Number of elements in each Row and Column				
		1000*1000	2000*2000	3000*3000	4000*4000	5000*5000
Serial	1	5.17	64.27	250.59	627.26	1467.31
MPI	5	1.42	13.05	52.57	135.78	229.59
	15	0.6	5.76	17.49	53.59	110.88
	25	0.28	2.61	10.51	21.11	60.35

Provided programming language: MPI and OpenMP
License: GNU GPL
Price: Free

REFERENCES

[1] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawak, and C. V. Packer, BEOWULF: A parallel workstation for scientific computation, *In Proceedings, International Conference on Parallel Processing,* **95***(1995).*

[2] T. H. Chew, K. H. Joyce-Tan, F. Akma, and M. S. Shamsir, birgHPC: creating instant computing clusters for bioinformatics and molecular dynamics. *Bioinformatics, 27**(9)**(2011), 1320-1321.*

[3] A. Chhabra, and G. Singh, A Cluster Based Parallel Computing Framework (CBPCF) for Performance Evaluation of Parallel Applications, *International Journal of Computer Theory and Engineering, 2***(2)***(2010), 226*

[4] M. Creel, PelicanHPC tutorial, 2008.

[5] D. D'Agostino, A. Clematis, E. Danovaro, E. Jeannot, and J. Zilinskas, Heterogeneous Parallel Computing Platforms and Tools for Compute-Intensive Algorithms: A Case Study, *High-Performance Computing on Complex Environments, (2014), 193-213.*

[6] L. Dagum, and R. Enon, OpenMP: an industry standard API for shared-memory programming, Computational Science and Engineering, *IEEE 5**(1)**(1998), 46-55.*

[7] H. Jin, R. Buyya, and M. Baker, Cluster computing tools, applications, and Australian initiatives for low cost supercomputing, *MONITOR Mag.(The Institution of Engineers Australia), 25**(4)**(2001).*

[8] G. E. Karniadakis? and R. M. Kirby, Parallel scientific computing in C++ and MPI: a seamless approach to parallel algorithms and their implementation, Cambridge University Press, (2003).

[9] KestrelHPC: Simple Diskless Clustering, Accessed 2 February 2014, from http://kestrelhpc.sourceforge.net.

[10] M. L. Massie, B. N. Chun, and D. E. Culler, The ganglia distributed monitoring system: design, implementation, and experience, *Parallel Computing. **30(7)**(2004), 817-840.*

[11] R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson, Effects of communication latency, overhead, and bandwidth in a cluster architecture, *ACM, 25***(2)***(1997), 85-97.*

[12] J. Q. Michael, Parallel Programming in C with MPI and OpenMP, Dubuque, IA: McGraw-Hill, 2003.

[13] W. P. Petersen, and P. Arbenz, Introduction to parallel computing, Oxford University Press, 2004.

[14] J. G. Siek, and A. Lumsdaine, The matrix template library: A generic programming approach to high performance numerical linear algebra, Computing in Object-Oriented Parallel Environments, *Springer Berlin Heidelberg, (1998), 59-70.*

[15] M. Snir, MPI–the Complete Reference: The MPI core, Vol. **1**, MIT press, 1998.

[16] M. Steggink, Reliable network booting of cluster computers, 2008.