International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-1, Issue-3, June 2015
ISSN: 2395-3470
www.ijseas.com

Performance Analysis of Algorithms on Shared Memory, Message passing and Hybrid Models for Standalone and Clustered SMPs

T Satish Kumar[1], S Sakthivel[2], Durgapriya G[3]
[1]Faculty of Information and Communication Engineering, Anna University, Chennai,
Tamilnadu, India
[2]Department of CSE, Sona College of Technology, Salem, Tamilnadu, India
[3]Department of CSE, RNS Institute of Technology, Bengaluru, India

ABSTRACT

While algorithms are well-understood in its sequential form, comparatively little would be known of how to implement parallel algorithms with main-stream parallel programming platforms and run it on SMP-based mainstream systems such as multi-core clusters. The project aims at better understanding the algorithmic techniques like divide and conquer, decrease and conquer, transform and conquer parallel algorithms on parallel platforms.
In this paper we investigate four benchmark parallel algorithms such as Quick sort, Matrix Multiplication, Montecarlo, LU Decomposition on Shared memory algorithms that run on SMP-based mainstream systems with OpenMP, Message-passing model on computer clusters with Open MPI, and combined Hybrid algorithms with combination of OpenMP and MPI that run on clustered SMPs. Parallel Benchmark programs are run on a dedicated Beowulf cluster and their performance were investigated.

INTRODUCTION

Parallel computing is a form of computation that allows many instructions to be run concurrently, in parallel in a program. This can be achieved by splitting up a program into independent parts so that each processor can execute its part of the program concurrently with the other processors. This can be done on a single computer with multiple processors or with number of individual computers connected by a network or a combination of the two. A parallel programming model is often associated with one or several parallel programming languages or libraries that recognize the Parallel algorithms model that are usually formulated in terms of a particular parallel programming model.

OpenMP (Open Multi-Processing), Message Passing Interface (MPI) and Hybrid Combination consisting of OpenMP and MPI is a parallel programming model where communication among the processes is done by message interchanging. OpenMP is an Application Programming Interface (API) [12] that provides multi-platform shared memory multi-processing in C, C++ and Fortran processor architectures and operating systems including Solaris Linux, AIX, HP-UX, Mac OS X and Windows platforms.

MPI [21] is a widely accepted standard for writing message passing programs. MPI provides the user with a programming model where processes communicate with other processes by initiating library routines to send and receive messages. The advantage of the MPI programming model is that the user has complete control over data distribution and process synchronization, permitting the optimization data locality and workflow.Message Passing Interface (MPI) is the de-facto standard for programming distributed memory systems as it provides a simple communication API and simplifies the task of developing portable parallel applications.

Hybrid OpenMP+MPI programming model facilitates cooperative shared memory programming across clustered SMP nodes[6]. MPI provides communication among various SMP nodes whereas OpenMP controls the workload on each SMP node. OpenMP and MPI are used in tandem to control the overall concurrency of the application.

EXISTING SYSTEM

The OpenMP and MPI programming models can be combined into a hybrid programming paradigm to exploit parallelism ahead of a single level. The main purpose of Hybrid parallel programming paradigm[17] is to adhere process level coarse-grain parallelism, which is obtained in domain decomposition and fine grain parallelism on a loop level which is achieved by compiler directives. The Hybrid programming approach is appropriate for clusters of SMP nodes where MPI is required for parallelism across nodes and OpenMP can be used to exploit loop level parallelism within a node. Applications are designed to run on a single system. But individual systems are not capable of solving the significant problems efficiently because of their inherent complexity. Sequential programs/single threaded programs designed cannot utilize the CPU power efficiently. Hence For these computation intense applications, there is a need for requirement of clusters to compute results.

In Hybrid model where MPI programming model combines with OpenMP programming model, the memory within each node is shared by OpenMP threads, hence the total memory consumption is much fewer than using MPI ranks exclusively[6]. Scalability is also enhanced due to less MPI communication between the nodes. Also, communication among lightweight threads within a node is much quicker than MPI communication sends/receives. However, to make efficient use of hardware, mapping of threads to existing cores must be done efficiently. Also, the total number of OpenMP threads per MPI rank must be chosen and the mechanism of MPI blocking must be carefully applied since it can result to deadlock. To run applications on clusters, MPI is a de-facto standard.The limitation is that it cannot harness the capacity of a multi-core processor. Hence multi-threading the applications must be done.

PROPOSED SYSTEM

Parallel programming model is a combination of the distributed memory parallelization on the node interconnects along with shared memory parallelization inside each node. The challenges and the potentials of the dominant programming models on structured hardware hierarchy is described: MPI (message passing interface), OpenMP (with distributed shared memory extension) and Hybrid OpenMP+MPI in several flavors. There are few cases where the hybrid programming model can certainly be the finer solution because of Memory consumption or improved load balance and reduced communication needs.

Hybrid programming introduces OpenMP into MPI application that makes more proficient shared memory usage on SMP nodes, thus justifying the requirement for explicit intra-node communication. Introducing OpenMP and MPI during the coding/design of a new application can maximize efficiency, scaling and performance. At the recent time, the hybrid programming model has begun to draw more consideration, for at least two reasons.

The first reason is that it is comparatively easy to pick a library/language instantiation of the Hybrid model: OpenMP plus MPI. Though there may be other approaches, they prolong as research and development projects, whereas OpenMP compilers and MPI libraries are now firm commercial products with implementations from various vendors.

The second reason is that a scalable parallel computer encourages this model. All the fastest machines virtually consist of multi-core nodes connected by a high speed network. The scheme of using OpenMP threads is to utilize multiple cores per node (with one multithreaded process per node) using MPI to communicate among the nodes.

SYSTEM ARCHITECTURE

System Architecture is shown in Fig.1. The User Interface (UI) layer consists of configuration files where the user can enter the configuration details like Number of cores, threads, hosts in the server and the input to the Benchmark program.

The next layer is the Execution layer. The various Benchmark programs are read using file handling functions and are parallelized using OpenMp, MPI and Hybrid OpenMp+MPI and evaluated.

The third layer is the communication layer. A Beowulf cluster is rigged up consisting of 8 nodes which can be scaled up as per the requirement. Open MPI is used to communicate between the master and the slave nodes. For this to happen, same piece of code has to reside on all the machines. Network File System (NFS) is used to share the common folder containing the source code. Open MPI uses SSH to communicate within nodes. So open SSH has to be installed and the password authentication has to be removed on all nodes. TCP/IP protocol is used for the communication.

EXPERIMENTAL SET UP

Table I shows the system configuration used in setting up the Beowulf cluster. All experiments undertake in this research uses a set of 1, 2, 4 and 8 nodes respectively.

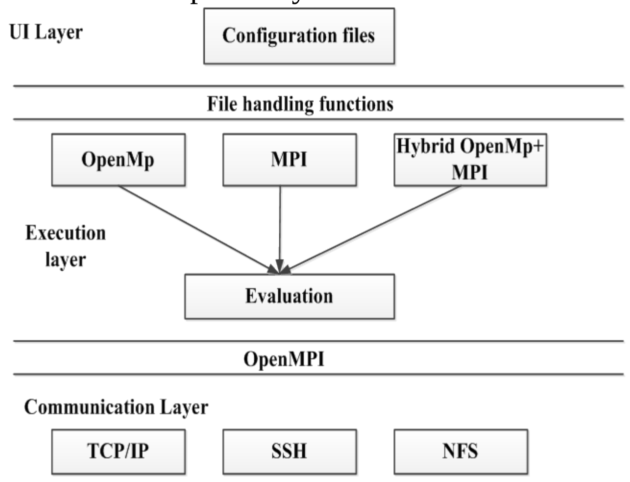

Fig.1: System Architecture

Table I: Target Architecture

# of nodes	1-8
# of cores per node	4
Memory per node	4GB
Open MPI version	1.4.2
Make	Dell 390 Optiplex
Processor	Intel core i5
OS	Ubuntu 12.04LTS
Clock Frequency:	2.5GHz

RESULTS

The performance of Quick sort, Matrix Multiplication, Montecarlo, LU Decomposition on the proposed architecture is as follows:

Fig. 2 to 5 presents performance of each programming model (OpenMP, MPI and Hybrid) for the Benchmark program discussed.

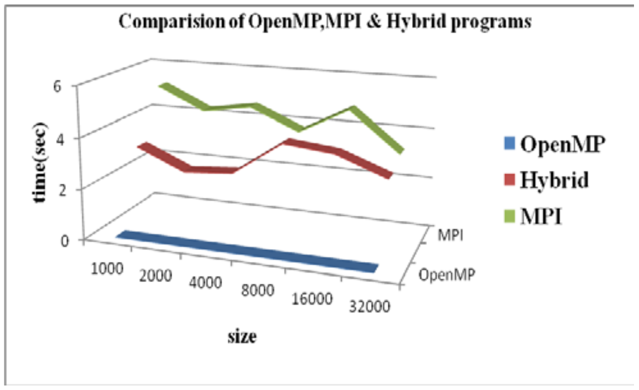

Fig.2: Comparison of OpenMP, MPI and Hybrid programs for Quick sort

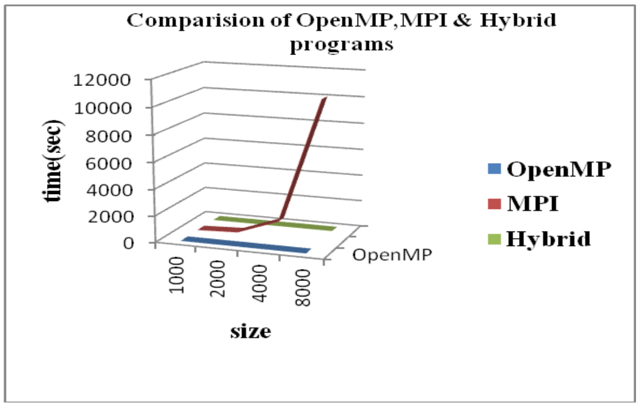

Fig.3: Comparison of OpenMP, MPI and Hybrid programs for Matrix Multiplication

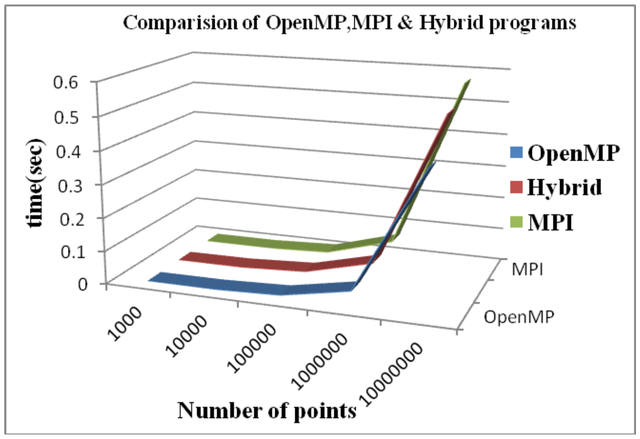

Fig.4: Comparison of OpenMP, MPI and Hybrid programs for Montecarlo

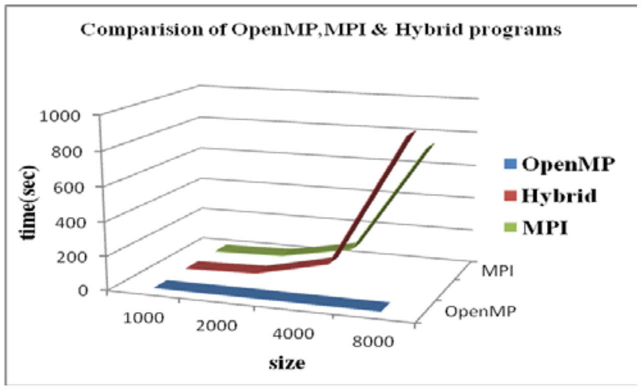

Fig.5: Comparison of OpenMP, MPI and Hybrid programs for LU Decomposition

Fig. 6 to 9 shows the average performance increase of MPI and Hybrid with respect to OpenMP programming. Clearly Hybrid outperform the MPI implementation.

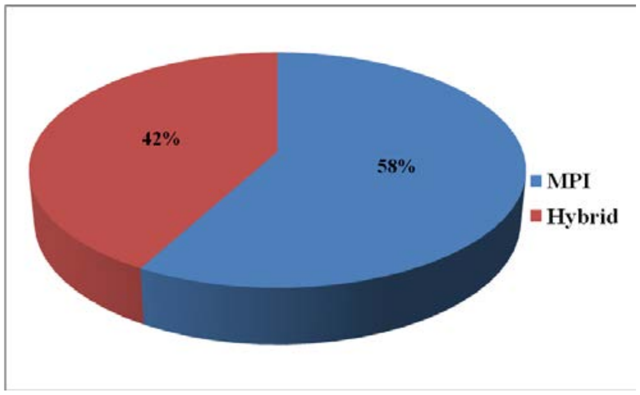

Fig.6: Time Enhancement Of hybrid and MPI with respect to OpenMP parallel programming model for Quick sort

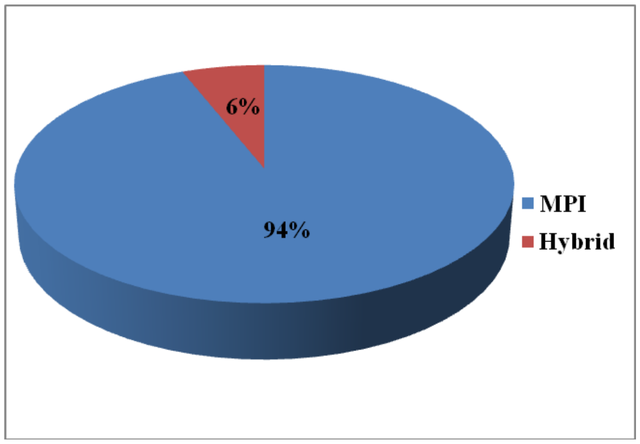

Fig.7: Time Enhancement Of hybrid and MPI with respect to OpenMP parallel programming model for Matrix Multiplication

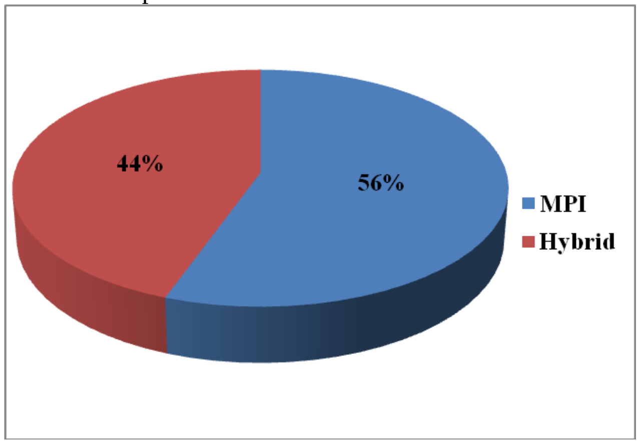

Fig.8: Time Enhancement of hybrid and MPI with respect to OpenMP parallel programming model for Montecarlo

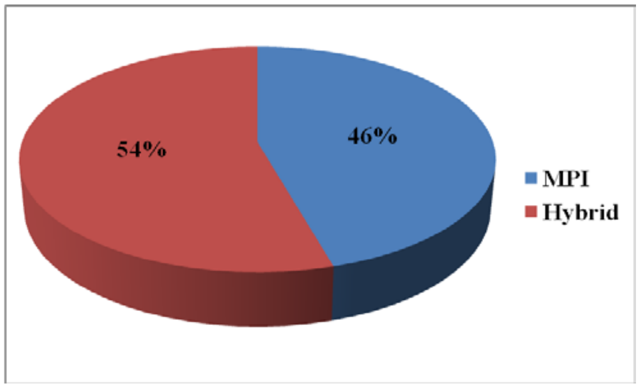

Fig.9: Time Enhancement Of hybrid and MPI with respect to OpenMP parallel programming model for LU Decomposition

Fig. 10 to 13 outlines the performance of Hybrid and MPI with respect to OpenMP for 1,2,4 and 8 cores. The evidence provides the hybrid model having an upperhand for all the Benchmark programs.

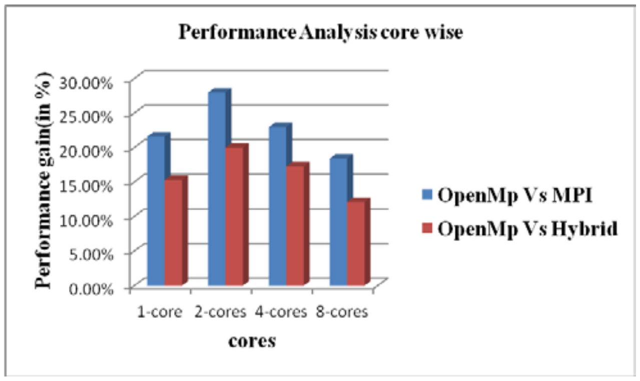

Fig.10: Performance Analysis core wise for Quick sort

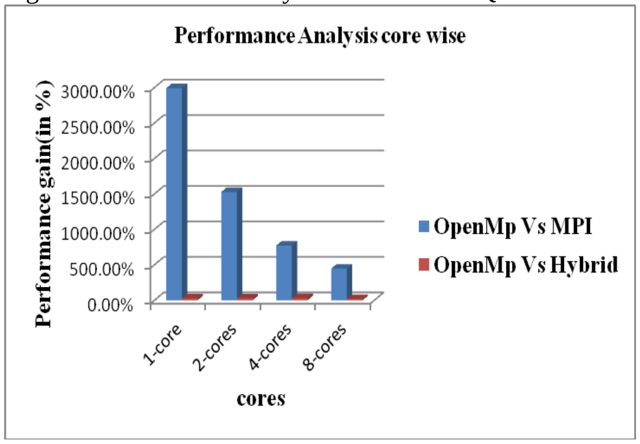

Fig.11: Performance Analysis core wise for Matrix Multiplication

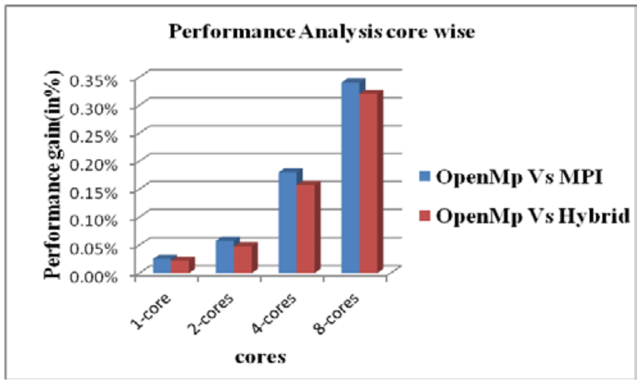

Fig.12: Performance Analysis core wise for Montecarlo

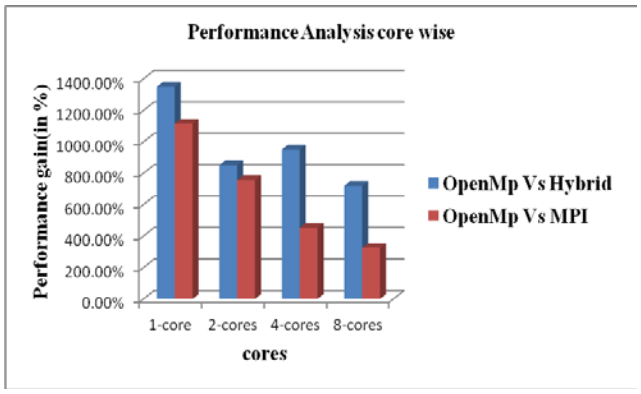

Fig.13: Performance Analysis core wise for LU Decomposition

CONCLUSION

The three parallel versions of benchmark programs: shared memory (with OpenMP), message-passing (with MPI) and Hybrid (with MPI and OpenMP) is been introduced.
While others have developed algorithms with either multi-threading or message-passing, this paper work offers efficient hybrid implementations. The use of hybrid implementation is straightforward; it has led us to a better speedup, to the readily existing high-performance instances. The overall performance of hybrid programming model results in an increase of 30%.

The work presented here is been extended to other benchmarks at larger scale and to begin developing various benchmarks specialized for the Hybrid approach. It is also observed that for certain benchmarks programs MPI proves to be better due to the fact that they are totally computation intensive application and spent lesser time on communication. One such benchmark used in the project was LU-Decomposition. To further increase the throughput of the Hybrid system better networking technologies may be used like InfiniBand, OpticFibre etc. MPI runtime parameters may be tuned for better performance by hybrid model.

REFERENCES

[1] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford, Introduction to Algorithms (3rd ed.), MIT Press, 2009.

[2] LaMarca, Anthony; Ladner, Richard, The influence of caches on the performance of sorting. Proc. 8th Ann. ACM-SIAM Symposium on Discrete Algorithms (SODA97), 370–379.

[3] R.Biswas and R. C. Strawn,A new procedure for dynamic adaption of three dimensional unstructured grids. Applied Numerical Mathematics, 13:437–452, 1994.

[4] Whaley R. C., Petitet A., Dongarra J. J., "Automated empirical optimizations of software and the ATLAS project" Parallel Computing 27, 1.2 (2001), 3-35.

[5] Ayguade, E. Copty, N., Duran, A., Hoeflinger, J. The Design of OpenMP tasks‖, IEEE Transactions on Parallel and Distributed systems, volume: 20, Issue: 3, pp. 404-418, June 2008.

[6] Early Experiments with OpenMP/MPI Hybrid Programming Model Ewing Lusk and Anthony Chan, Mathematics and Computer Science Division Argonne National Laboratory, ASCI FLASH Center, University of Chicago.

[7] Daniel Lorenz, Peter Philippen, Dirk Schmidl and Felix Wolf:"Profiling of OpenMP Tasks with Score-P". In *Proc. Of the 41st International Conference on Parallel Processing Workshops(ICPPW), Workshop on Parallel Software Tools and Tool Infrastructures(PSTI), Workshop on Parallel Software Tools and Tool Infrastructures(PSTI),* pages 444-453, September 2012

[8] Dixie Hisley, Gagan Agrawal, Punyam Satya-narayana, and Lori Pol-lock, Porting and performance evaluation of irregular codes using OpenMP. In In proceesing of First European Workshop on OpenMP (EWOMP 1999), Lund, Sweden, pages 47–59, 1999.

[9] Timothy G. Mattson. How good is openmp. Scientific Programming, 11(2):81–93, 2003.

[10] Milind Kulkarni, Keshav Pingali, Bruce Walter, Ganesh Ramanarayanan, Kavita Bala, and L. Paul Chew. Optimistic parallelism requires abstractions. In Proceedings of the 2007 ACM SIGPLAN con-ference on Programming language design and implementation (PLDI 2007), San Diego, CA, USA, pages 211–222, 2007

[11] Jarek Nieplocha, Andres` Marquez,´ John Feo, Daniel Chavarr´ıa- Miranda, George Chin, Chad Scherrer, and Nathaniel Beagley, Evaluating the potential of multithreaded platforms for irregular scientific computations. In Proceedings of the 4th international conference on Computing frontiers (CF 2007), Ischia, Italy, pages 47– 58, 7–9 May 2007.

[12] OpenMP,The OpenMP API specification for parallel programming http:// openmp .org/wp/, 1998.

[13] Simone Secchi, Antonino Tumeo, and Oreste Villa. A bandwidth optimized multi-core architecture for irregular applications. In Proceed-ings of 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, (CCGrid 2012), Ottawa, ON, Canada, pages 580–587, 13–16 May 2012.

[14] D.S. Henty, Performance of hybrid message-passing and shared-memory parallelism for discrete element modeling. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 10, Dallas, Texas, United States, 2000.

[15] T. Satish Kumar, S Sakthivel, Sushil Kumar S, May 2014. Optimizing code by selecting Compiler flags using Parallel Genetic Algorithm on Multicore CPUs, International Journal of Engineering and Technology, Vol 6, No 2 .

[16] Rabenseifner, R., "Hybrid Parallel Programming: Performance Problems and Chances", Proceedings of the 45th Cray User Group Conference, Ohio, May 12-16, 2003.

[17] Y. He and C. H. Q. Ding. MPI and OpenMP paradigms on cluster of SMP architectures: the vacancy tracking algorithm for multi-dimensional array transposition. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, Baltimore, Maryland, USA, IEEE Computer Society Press,2002.

[18] The OpenMP specification for parallel programming. Retrieved on March 1, 2011 from http://openmp.org.

[19] Radenski, A. Shared Memory, Message Passing, and Hybrid Merge Sorts for Standalone and Clustered SMPs. Proc. PDPTA'11, the 2011 International Conference on Parallel and Distributed Processing Techniques and Applications,

CSREA Press (H. Arabnia, Ed.), pp. 367 – 373,2011.

[20] Aaftab Munshi. The opencl specification. Khronos OpenCL Working Group, 1:l1–15, 2009.

[21] T.Satish Kumar,S Sakthivel and Manjunatha Swamy M, “Optimizing MPI Communication using Heuristic Algorithms” Asian journal of information technology 13(11),700-706.

[22] R. Eigenmann and B.R. de Supinski (Eds): IWOMP 2008, LNCS 5004, pp. 36–47, 2008. Springer-Verlag Berlin Heidelberg, 2008.

[23] John J. Buoni,Paul A. Farrell and Arden Ruttan, ”Algorithms for LU decomposition on a shared memory Multiprocessor”,1993, 925-937.