How to cite this paper:

Sajad Sajadpour, Jaspaljeet Singh Dhillon, & Abdul Rahim Ahmad. (2017). Time consumption improvement of matrix multiplication by hybrid execution of message passing interface and open multi-processing in Zulikha, J. & N. H. Zakaria (Eds.), Proceedings of the 6th International Conference on Computing & Informatics (pp 758-767). Sintok: School of Computing.

TIME CONSUMPTION IMPROVEMENT OF MATRIX MULTIPLICATION BY HYBRID EXECUTION OF MESSAGE PASSING INTERFACE AND OPEN MULTI-PROCESSING

Sajad Sajadpour, Jaspaljeet Singh Dhillon, Abdul Rahim Ahmad

College of Computer Science and Information Technology
Universiti Tenaga Nasional, Malaysia,
sajad.sajadpour@gmail.com, {jaspaljeet, abdrahim}@uniten.edu.my

ABSTRACT. The mixed-mode OpenMP and MPI programming structure in parallel programming have significant impact on multi-core clusters in terms of performance. Hybrid programming came to new step in parallel programming whereas the new architecture has been manifestation of multi-core cluster. This model of programming which is new leads this idea to the next level of research of shared memory on concepts and MP. The goal of this research is to analyzing pure MPI and hybrid code how will behave on two separated multi-core cluster architecture in use of parallel application matrix multiplication. Hybrid codes compelled to solve the lack of performance compare to the pure MPI in less amount of cores in overhead from shared memory. Although, hybrid codes efficiencies pure MPI at bigger amount of processors because of decreasing in communication overhead that leads to lower processing time.

Keywords: Hybrid programming, parallel computing

INTRODUCTION

Newly, novel trends on the way to parallel architectures have offered improved response time and performance in the case of parallel computation applications. Multi-core clusters demonstrated obvious changes in hardware architectures during HPC generations. Symmetric multi-processors (SMP shared memory systems) or single core nodes are the examples of this change. Multi-core clusters make this feasible to mix two features of parallel programming languages; message passing in shared memory and multi-cores using shared memory.

Memory latency; that is the time elapsed from point a piece of data is required until data become available, and memory bandwidth; that is the speed which data are sent from memory to processors are two main factors which effects on memory. The most general programming model, which uses message passing, is MPI. MPI programs run on shared memory nodes easily through running one or numerous MPI processes per node. MPI is a message transient interface. It can apply on programs using message passing. Programmers can use MPI to progress programs, which is using message passing (distributed memory). Furthermore, the programming by C or FORTRAN is possible.

Open MP, is defined as a standard in shared memory in terms of work sharing between threads. Start of Open MP is for parallelism and work sharing. This is more inaccessible and

movable for developers. It follows Fork-Join strategy. Open MP is programmed in C/C++ or FORTRAN programming languages. Using of a combination of MPI and Open MP are accepted method on multi-core clusters.

A comparison by Capella and Estimable on hybrid MPI/Open MP of NAS benchmarks with Pure MPI showed that the performance test result depends on numerous factors like as memory access patterns and hardware specification. Another research by Smith and Bull demonstrated that certain conditions the hybrid can attain better performance than pure MPI codes; however, it is not well matched for all applications. Built-in Open MP with further loop level and multi-threaded organization have achieved good performance on distributed computers with shared memory nodes and thousands of CPUs.

The condition of memory, has a direct influence on the performance of multi-core clusters architectures which is based on shared memory. In certain conditions hybrid code outclasses pure MPI in relations of less memory intake but outweigh by software difficulty and less know-how. MPI standard proposed functional ways for mapping procedures to processing cores. JDL file and SH, files are two required files for running MPI. The JDL file identifies amount of cores and make output file and error file. Name of SH file and source code Save from JDL file. In addition, JDL file describe, as cluster code has to run.

The benefits of manual mapping are mentioned as below: No source code modification; no mapping process while performance time and enhance functionality for compatibility. However, the files cannot adapt dynamically and it takes as a disadvantage because it passes factors upon execution. A function can avoid threads to be spreaded between processors and a specific number of dispensation cores will allocate to threads. The differences of two mapping files are observed while adding SMPGRANULARITY, which has the same number of processing cores. The aim of this paper is to design a hybrid open MP / MPI algorithm, also determine the performance of hybrid and pure MPI, and compare with each other in runtime perspective.

METHODOLOGY

INSPEM is one of the parallel systems of Universiti Putra Malaysia used for performance analysis of case study. It has 8 computing blades that connected by a GB Ethernet with speed of 1 GB/Per Sec. Each blade has two 6-core Intel Xeon E5-2620 with clock speed 2.0 GHz and 48 GB memory. The system software is Scientific Linux 6.2, with version OpenMPI 1.6 and OpenMP gcc-4.4 for compilation of OpenMP and MPI.

Another HPC in faculty of Mathematic (Math HPC) included three nodes, which are connected by a Gigabit Ethernet. The specifics are: eight six-core AMD Opteron processor 8435 with clock speed of 800 MHz for each node, 64 GB RAM, and Centos 5.3 as system software.

The MPI and hybrid Open MP/MPI codes were applied by two different multi-core clusters with different architectures. All code ran three times on changed range of cores and the fastest process time considered for judgment. The parallel calculation is matrix multiplication. The size of computation is 1000 squares. In order to record threads and processes to processors, some old operating systems make it practicable to perform it manually for scheduling techniques.

Manual mapping will help programmers run applications without adjusting codes. Manual mapping can be applied according to architecture of multi-core clusters to reach better performance. The essential of manual mapping is due to amalgamation of shared memory and message passing in hybrid algorithm. MPI standard offers functional techniques for mapping processes to processing cores. Running MPI task needs 2 files; JDL file and SH file. The JDL

file identify number of cores and also make output file and error file. Name of SH file and source code bring back from JDL file. In inclusion JDL file define which cluster code need to be run. These two files should pass parameters upon implementation. The pros of manual mapping are: no source code modification; no mapping process while implementation time and add functionality for compatibility. However, the files cannot modify dynamically and that take as a cons because it passes parameters upon execution. The sample of SH file for process are shown in Figure 1 and Figure 2.

```
#!/bin/sh

export PATH=$MPI_OPENMPI_PATH/bin:$PATH
export LD_LIBRARY_PATH=$MPI_OPENMPI_PATH/lib:$LD_LIBRARY_PATH

mpicc matrix.c -o matrix
mpiexec matrix
```

Figure 1. Sample SH file.

```
JobType = "Normal";
CPUNumber  = 12;
Executable = "matrix.sh";
Arguments = "";
StdOutput = "std.out";
StdEror = "std.err";
InputSandbox = {"matrix.sh", "matrix.c"};
OutputSandbox = {"std.err","std.out"};
Requirements =
Member("OPENMPI",other.GlueHostApplicationSoftwareRunTimeEnvironment)&
&(other.GlueCEInfoTotalCPUs >=
12)&&(other.GlueCEUniqueID=="ce.inspemhpc.upm.edu.my:8443/cream-pbs-
putragrid");
```

Figure 2. Sample JDL File.

For the shared memory processes is different from process mapping and it is not possible to assign it to a specific processing core. We can find a function that can be avoided the threads to be extended among processors and exact amount of processing cores will assign to threads. This thread has two fines: JDL files and SH files but the distinctions observed while adding SMPGRANULARITY which has to be same as number of processing cores. As well as, must define amount of threads in SH file that must be same as number of cores, means each core has its own thread.

Since hybrid OpenMP/MPI applications have a two-level hierarchy (processes and threads), invocation of hybrid programs follows a slightly different approach as compared to invoking pure-MPI programs. To understand this, consider the architecture of today's clusters. These clusters are built using multiple nodes (or computers) connected to each other using a high speed network such a Gigabit Ethernet, Myrinet or InfiniBand. Inside each node is a hierarchy of sockets and cores.

For instance, INSPEM each node has 12 cores in which included with two sockets, each socket has 6 cores. Considering the typical design of clusters as mentioned above, the best

way to run hybrid MPI/OpenMP programs would be to have 1 MPI task per node and an equal number of OpenMP threads per task as the number of cores per node. The hybrid and MPI OpenMP/MPI codes has been applied in 2 different multi-core clusters with different architecture as we discussed lately. All of the codes has been run three times on different span if cores and the fastest process time allocated for comparison.

The research for parallel computation is matrix multiplication and the computation sizes are 1000/2000 Sq. In order to run and test of MPI performance, amount of processes defined as number of cores. To run hybrid code, this work exert one MPI processor core that works as a master to spawn threads between cores and finally all data will synchronize toward MPI tasks. The OMP_NUM_THREADS environment variable used to spawn threads in the node which given number of cores defined as (MPI processes) * (OpenMP threads) per node. To measure process time in MPI and hybrid code, MPI_Wtime() used as a method to measure wall time process. As Figure 3 shows, matrix multiplication ran in two different methods on multi-core clusters are described below:

- Pure MPI- One MPI process ran on each core in the node without OpenMP threads
- Hybrid Open MP/MPI- One MPI process started on each node and other cores covered by Open MP threads.

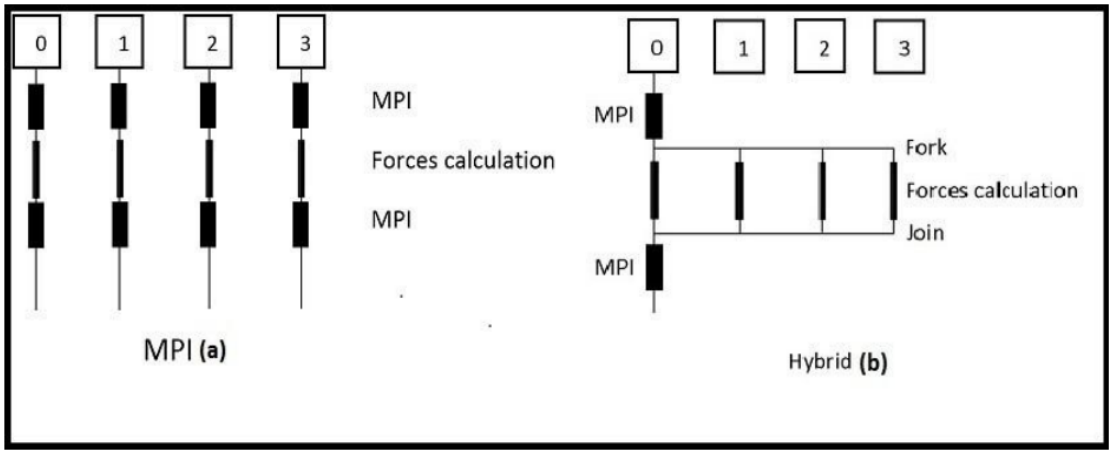

Figure 3. MPI and Hybrid Open MP/MPI.

Hybrid Open MP/MPI algorithm has three main parts which twisted together in order to deliver high performance and lowest process time. MPI structured main body that starts communications and finalize processes at end, parallelism inner part of MPI with process rank zero.

The algorithm structured as three parts which first part broadcast data to processes, broadcast will optimize looping and rolling between ranks, collective communications is faster and efficient, also apply optimization over range of messages size, common sizes and specific rank layout. After broadcast, MPI_Isend method sends interval of data to workers in order to get processed. At final all calculated date receive from worker by MPI_Irecv.

Second part has structured to parallel calculation by Open MP that #pragma omp parallel used as main region of parallelism and main parallel region has two parts which first one parallel interval by #pragma omp for schedule, this method has two parameters which are static and chunk. Static parameter do block iteration which defined by chunk size 10 to each thread. In order to parallel calculation of remainder data used #pragma omp for schedule as well.

Third part has to organize calculation and processes of data when root rank is something different of zero so all processes passed to rest MPI process on other nodes, this part is complete version of first and second parts in which third part parallelism happen by #pragma omp parallel for workers processes.

RESULTS

As results of analysis that we get from performing of hybrid codes tested by two principal parameters; the problem with size and the number of cores that we have used. At lower amount of processors, pure MPI in some cases captured more efficient performance than hybrid, but when it goes to bigger amount of processors hybrid outperformed pure MPI.

The architecture of multi-core clusters has been made an important role that affect the performance of Pure MPI and hybrid OpenMp/MPI. Pure MPI faced with overhead communication between nodes mostly on INSPEM multi-core cluster because of architecture. There are 8 nodes in INSPEM with bigger amount of communications occurred during processes, otherwise when MPI is running on Math HPC, it did not face with communications overhead because it only has three nodes.

Hybrid OpenMP/MPI has been solved the problem on INSPEM also on Math HPC, in the step of process time of parallel application matrix multiplication. after while comparison of table 1 in pure MPI performance on INSPEM and Math HPC, it can be resulted that math HPC performance is better while increasing number of cores also with lower CPUs clock speed because, when amount of processors increase on INSPEM the processes need to communicate between nodes. But then Math HPC nodes with 48 cores and less amount of nodes did not deal with communications overhead and process time decreased drastically.

The runtime process for INSPEM measured till 96 cores and for Math HPC till 144 cores. According to table 2, after exerting hybrid code on Math HPC and INSPEM, hybrid code performs better than pure MPI code in terms of communication which decreased number of communication times that result in better process time and performance.

Table 1. Pure MPI Runtime of INSPEM and Math HPC.

Core	Pure MPI (INSPEM)	Pure MPI (Math HPC)
1	7.455	22.685
2	5.075	12.478
4	3.852	7.311
8	3.067	5.723
12	2.698	3.788
16	2.456	3.214
24	2.533	3.442
32	4.231	2.884
48	3.114	3.533
64	4.856	3.423
96	5.983	3.495
144	--	3.678

The result of process time that we get from pure MPI on Math HPC, which is more stable and predictable than INSPEM, largely and decreasing in process time and at the same time shown in Figure 4 increasing the amount of cores after reaching to cores 48. Pure MPI graph of Math HPC shows slightly increment that is owing to pass messages from node 1 to 2. The processes time from core 48 until 144 is still stable only with a little millisecond changes. In opposite, INSPEM process time from core 1 until core 12 show us better stability because all of the message passing are in 1 node. When it is reaching to core 16, the graph increasing slightly that shows nature of communications in 1st and 2nd nodes. On core 24 processing time reduces a little which is happen on core 48. Core 32 in incrementing similar to core 16 but the processing time increasing with stability because message passing need to be done between 8 nodes.

Table 2. Pure MPI and Hybrid Runtime of INSPEM.

Core	Hybrid (INSPEM)	Pure MPI (INSPEM)
12	2.987	2.964
24	2.468	2.533
48	2.586	3.611
96	3.524	5.728

Table 3. Pure MPI and Hybrid Runtime of Math HPC.

Core	Hybrid (INSPEM)	Pure MPI (INSPEM)
24	3.655	3.389
48	3.035	3.225
96	3.085	3.945
144	2.598	3.876

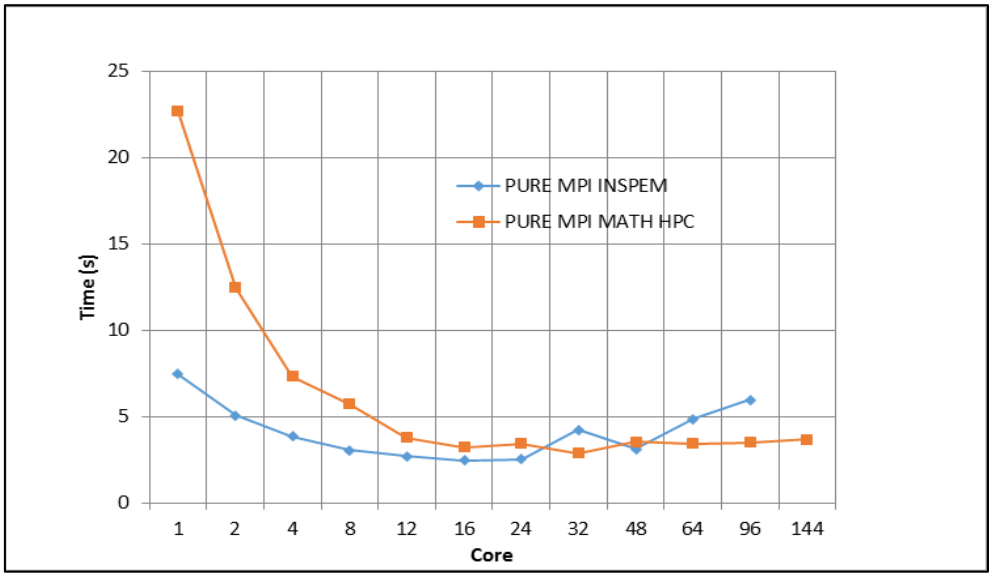

Figure 4. Runtime Pure MPI of INSPEM and Math HPC.

Math HPC memory's node and INSPEM memory's node is different from memory of each node on INSPEM, INSPEM memory is 48 GB but in MATH HPC is 64 GB. It will effects of processing time in bigger amount of cores. The process time from 48 until 96 cores are more stable and less than INSPEM. It shows remarkable increase because the memory in each node is lower. When memory of node is in higher stage, MPI task in bigger amount of cores takes more processing data and process perform is more stable. In Figure 5, as we can see that the speedup on INSPEM and Math HPC. Pure MPI speedup was computed up to 48 cores on both INSPEM and Math HPC.

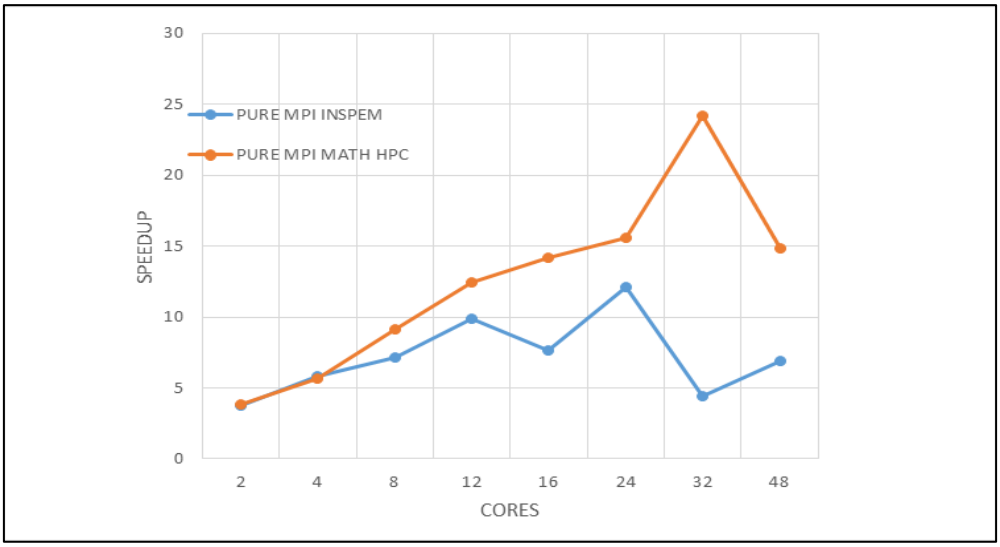

Figure 5. Speedup pure MPI on Both System.

Figure 6 and Figure 7 show runtime for Table 2 and Table 3 respectively, that hybrid code outperforms pure MPI at higher number of cores on both multi-core clusters.

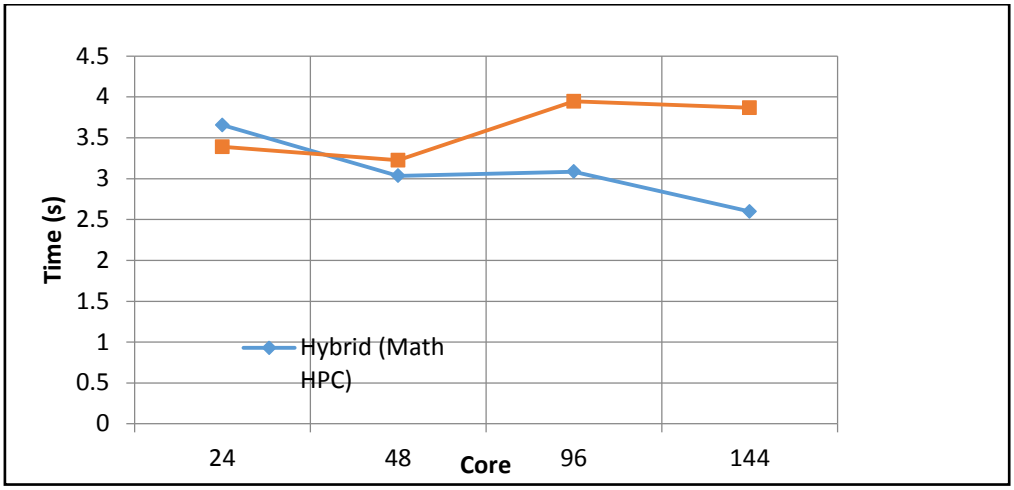

Figure 6. Runtime Pure MPI and Hybrid of Math HPC.

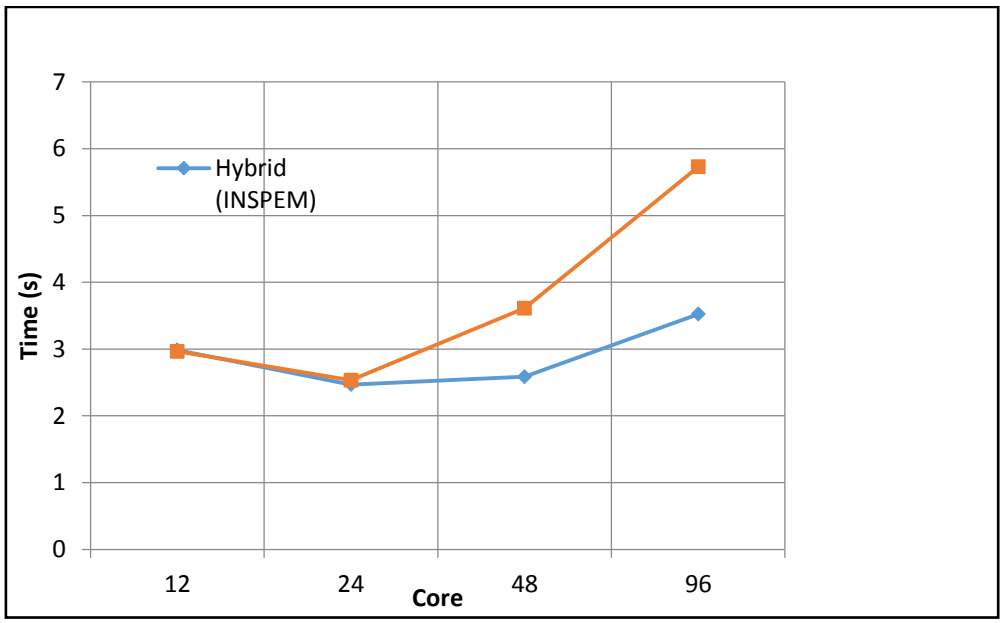

Figure 7. Runtime Pure MPI and Hybrid of INSPEM.

CONCLUSION

Pure MPI and hybrid's performance, depends on few factors which are multi-core cluster architecture, system interconnect and number of cores. OpenMP allocated for MPI to design hybrid codes in matrix multiplication, it has been run in 2 multi-core clusters with separated architectures. The analysis of the performance shows that pure MPI outcome hybrid OpenMP/MPI on both multi-core clusters on lower number of core. The hybrid has been showed the lack of performance on both systems due to overhead happened in shared memory by OpenMP but in bigger amount of cores performed better than pure MPI because of decreasing of communication. Memory of node has affected the stability and process time merely in bigger number of cores that higher amount of data has to be processed.

System interconnected will impact on pure MPI performance directly. Using faster Infini-Band interconnect makes pure MPI better in performing than hybrid because communications process is faster and become competitive with hybrid codes. On both systems run hybrid and pure MPI, it has same system interconnect. Higher amount of nodes on INSPEM cause lack of performance in compare with HPC on higher amount of cores even with bigger clock speed CPU. Math HPC has 3 nodes and that will lead MPI toward less communication on higher amount of cores.

REFERENCES

A. Rane, "A STUDY OF THE HYBRID PROGRAMMING PARADIGM," 2009.

B. Barney, "POSIX threads programming," National Laboratory. Disponível em:< https://computing. llnl. gov/tutorials/pthreads/> Acesso em, vol. 5, 2009.

F. Cappello and D. Etiemble, "MPI versus MPI+ OpenMP on the IBM SP for the NAS Benchmarks," in Supercomputing, ACM/IEEE 2000 Conference, 2000, pp. 12-12.

F. Leibovich, M. Naiouf, L. De Giusti, F. G. Tinetti, and E. De Giusti, "Hybrid Algorithms for Matrix Multiplication on Multicore Clusters," ed: Julio, 2012.

G. Krawezik, "Performance comparison of MPI and three OpenMP programming styles on shared memory multiprocessors," in Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures, 2003, pp. 118-127.

H. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang, and B. Chapman, "High performance computing using MPI and OpenMP on multi-core parallel systems," Parallel Computing, vol. 37, pp. 562-575, 2011.

H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," *Dr. Dobb's journal*, vol. 30, pp. 202-210, 2005.

M. Feldman, "Our manycore future," 2007.

J. M. Bull, J. Enright, X. Guo, C. Maynard, and F. Reid, "Performance evaluation of mixed-mode OpenMP/MPI implementations," *International journal of parallel programming*, vol. 38, pp. 396-417, 2010.

K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," in *Job Scheduling Strategies for Parallel Processing*, 1998, pp. 62-82.

Kotobi, A., Hamid, N. A. W. A., Othman, M., & Hussin, M. (2014, August). Performance analysis of hybrid OpenMP/MPI based on multi-core cluster architecture. *In Computational Science and Technology (ICCST), 2014 International Conference* on (pp. 1-6). IEEE.

L. Adhianto and B. Chapman, "Performance modeling of communication and computation in hybrid MPI and OpenMP applications," *Simulation Modelling Practice and Theory*, vol. 15, pp. 481-491, 2007.

M. J. Berger, M. J. Aftosmis, D. Marshall, and S. M. Murman, "Performance of a new CFD flow solver using a hybrid programming paradigm," *Journal of Parallel and Distributed Computing*, vol. 65, pp. 414-423, 2005.

M. Kosiedowski, C. Mazurek, and M. Stroinski, "PROGRESS–Access Environment to Computational Services Performed by Cluster of Sun Systems," in *Proceedings of the 2nd Cracow Grid Workshop, Cracow Poland*, 2002, pp. 45-56.

X. Wu, V. Taylor, C. Lively, and S. Sharkawi, "Performance analysis and optimization of parallel scientific applications on CMP cluster systems," in *Parallel Processing-Workshops*, 2008. ICPP-W'08. International Conference on, 2008, pp. 188-195.

X. Wu and V. Taylor, "Using processor partitioning to evaluate the performance of MPI, OpenMP and hybrid parallel applications on dual-and quad-core Cray XT4 systems," in the *51st Cray User Group Conference (CUG2009)*, 2009, pp. 4-7.

X. Wu and V. Taylor, "Performance Characteristics of Hybrid MPI/OpenMP Scientific Applications on a Large-scale Multithreaded BlueGene/Q Supercomputer," in *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2013 14th ACIS International Conference on, 2013, pp. 303-309.