ISSN 2278-3091
Volume 8, No.1.4, 2019
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse4281.42019.pdf
https://doi.org/10.30534/ijatcse/2019/4281.42019

CNDS-Performance Analysis of Parallel Programming Models for Compute-Intensive Problems in Multi-core Environment

Gamil Gardan[1], Nor Asilah Wati Abdul Hamid[1,2], Mustafa Saleh Al-Khaffaf[3]
[1]Communication Technology and Network Department, Universiti Putra Malaysia, Malaysia
[2]Institute for Mathematical Research (INSPEM), Universiti Putra Malaysia, Malaysia
[3]Computer Science and Information Technology Department, University of Kufa, Iraq
j.jar49n@gmail, asila@upm.edu.my, mustafamsk91@gmail.com

ABSTRACT

Parallel programming models have become commonplace, and these models allow developers and programmers to deal with data in many ways. There are many parallel programming models available to date, however, the current study has chosen the seven most recognized parallel programming paradigms to be compared and benchmarked, namely MPI (point-to-point and collective), OpenMP, PThreads, TBB and hybrid (MPI/OpenMP and MPI/PThreads). Besides that, the benchmark used in this study is matrix multiplication, and they are evaluated based on different matrix sizes. The execution time, speedup, and efficiency of the models are used to analyse the behaviours of these models with different number of processors and matrix sizes. The results have demonstrated that, in most cases, OpenMP and MPI (Point-to-Point) are ideal for compute-intensive problems, and they both benefit from many-core architecture. In addition, the findings have also exhibited that TBB provides good performance with low programming complexity and code changes, especially with small sized computation problems.

Key words : HPC, MPI, OpenMP, Parallel programming models, PThreads, TBB.

1. INTRODUCTION

High Performance Computing (HPC) is a practice that accumulates computing power to generate higher performance level, compared to a normal computer. The higher accumulated power is especially crucial to solve complex tasks from diverse sectors, namely engineering, business and science and business [1]. HPC is also commonly referred to as parallel computing and supercomputing [1]. In addition, the chief goal of HPC is to enhance the efficiency and performance, and this is achieved through every step of parallelization, which includes assignment, mapping, decomposition, and synchronization [2]. This development and enhancement incorporates supercomputers, data centers, and devices that run through GPU or CPU processing unit. Due to the growing needs of parallel computing, there is also a steady increase in the number of processors to conform with the demands [3]. While there are numerous parallel programming models introduced in assisting researchers, developers and programmers, four of the most widely applied are POSIX threads (Pthreads), MPI, OpenMP, and Threading Building Blocks (TBB) [4], [5].

Although there are a number of comparative studies that have been done on two or three of these selected models, namely (Openmp vs Pthreads) [6], (MPI vs OpenMP) [7, 19]. (OpenMP vs TBB), (OpenMP vs Pthreads vs TBB) or any hybrid models (combining of two models) [8], none of these studies so far have compared these five models together. Thus, in this paper, five selected parallel programming models, namely POSIX threads (Pthreads), MPI Shared Memory with two communication routines (Point-to-Point and Collective), Threading Building Blocks (TBB), OpenMP, and Hybrid(MPI/OpenMP and MPI/Pthreads) will be examined. Based on these selected models, a comparative study are conducted in compute-intensive problems to acquire and offer a lucid information to assist potential researchers in choosing the most appropriate models that fit their requirements.

The rest of the paper is organized as follows. In Section 2, a brief discussion of parallel programming models and their features. The related studies have been discussed in Section 3. The implementation of the test, applying the algorithm and the performance metrics, and other requirements have been explained in Section 4. Section 5 illustrates the results of the experiment and analysing them, and in Section 6 the findings are stated. Finally, Section 7 presents a conclusion for the study and future work.

2. PARALLEL PROGRAMMING MODELS ON MULTI-CORE SYSTEMS

After the discovery of how clock speeds are increased through the chip heat dissipations, the concept of Moore's law has been changed into add processor cores. Due to this, the manufacturing principle has changed, by equipping a single chip with more processor cores [9].

Message Passing Interface (MPI) is the most preferable approach for programming parallel applications, and most systems are using this strategy. For this reason, many MPI developers take into account of the implementation of MPI on multi-core shared memory, and try to improve the

communication with one node. Thus, the parallel programming users can run their application based on standardized programming model with a good performance [10]. MPI has an extension which uses shared-memory process model (SMPM), to deal with a shared-memory platform [7].

The second model, Open Multi-Processing (OpenMP), is a portable shared memory API and the de-facto standard for parallel programming which is a high-level, which aims to make shared memory parallel programming easier. Most of HPC programs use OpenMP [11]. Next, PThreads or Portable Operating System Interface (POSIX) Threads, is a low-level thread library and a common portable API [12].

Another selected model, which is Threading Building Blocks (TBB), is a C++ template library developed by Intel. TBB divides a computation into tasks, to assign them to chosen core to be processed in parallel. TBB consists of Parallel algorithms and data structures and it provides scalable memory allocation and task scheduling [13]. Besides that, this study also utilizes hybrid programming, which combines two parallel programming models in order to obtain the strengths of both models. These include scalability of the distributed memory model, the ease of programming, the efficiency, and memory savings of the shared memory one. Hybrid model usually combines MPI as a distributed memory model with any shared memory model, either OpenMP, PThreads, or CUDA [11].

3. RELATED WORK

Several parallel programming models have been studied on multi-core systems using matrix multiplication algorithm as a benchmark program by researchers. While there are a number of studies done on the qualitative pros and cons of each model, there are, however, only limited studies done on their quantitative performance, which is needed to assist new researchers in choosing the most suitable model. Studies have been done by [6], [7], [14], [15] to compare a pair of parallel programming models that involve MPI, OpenMP, Pthreads, and TBB. Through the comparison between OpenMP, and MPI, openMP has been found to be providing great performance due to its ability to utilize thread level parallelism in most situations [14]. However, another study [7] concluded that, although OpenMP performed better than MPI in some matric sizes, it has been found that MPI performs significantly better in other sizes. PThreads has been found better performance against MPI, but for a lesser number of threads, MPI takes the lowest execution time [6]. Kim and Seo (2016) have found that OpenMP and TBB performed substantially faster and better, compared to the serial ones. Nevertheless, in comparison to TBB, OpenMP is able to provide better performance. However, the earlier mention studies have limited to a small number of cores which do not exceed 32 cores and some of them have a small size of matrices.

Other studies [12], [16], [17] have compared three different models together; Sharma and Soni (2014) have compared Posix Threads, OpenMP, and Microsoft Parallel Patterns libraries. The study has identified that PThreads and OpenMP provide the best speedup compared to the Microsoft Parallel Patterns libraries. Besides that, in terms of providing a higher number of matrices and better outcomes, OpenMP precedes PThreads and Microsoft Parallel Patterns Libraries. In general, OpenMP also requires less execution time compared to the other, however, PThreads is a better option for smaller matrix dimensions, with a better speedup and lower time execution. This study however, does not offer a thorough explanation of the link between the speedup with the processors' numbers. Another important point to note is that this study limits itself to only 8 processors. OpenMP, Intel cilk plus, and TBB have been compared together [12], [17]. From an experiment with 8 cores conducted by Leist and Gilman (2014), it has also been found that the task creation and scheduling overheads introduced by a very large number of small tasks, affect OpenMP more severely than they affect Cilk Plus, or TBB. In another study [17] with a higher number of cores which is 60, their results have demonstrated that TBB excels in providing better speedup for smaller sized problems, compared to the other two models, while OpenMP delivers better performance for larger sized problems.

A comparison between five models, namely OpenMPI, Thread Building Blocks (TBB), OpenMP, Pthreads, and Intel®'s CilkTM Plus has been conducted [18] in order to analyze the performance and problem complexity. It has been found that OpenMP, Cilk++, or TBB can lessen the level of the problem complexity, as they will automatically manage the threads. On the other hand, PThreads has been found to be the most complex model. In addition, this study has also identified that TBB provides a better control and performance on data-parallel loops. However, a generalization of the results cannot be made on a higher number of cores, as the experiment was performed with only 2 cores and threads. Besides that, a study [13] has also been done on six parallel programming models namely PThreads, Intel Cilk Plus, OpenMP, Intel TBB, FastFlow, and SWARM to compare them based on their performance, and programmer productivity. Based on the experiment, it has been identified that at medium or large scales matrix operations, both SWARM and OpenMP are able to provide good results. In addition, this study has also found that SWARM, OpenMP, TBB, and Cilk Plus do not demand high programming effort.

In the final study [8], the researchers have implemented and analysed Sparse matrix vector multiply (SpMVM) algorithms on multi-core architectures through the utilization of a hybrid parallel paradigm between OpenMP, and MPI. The comparison is made between SpMVM and the previous implementations, by measuring the resource usage on supercomputers with CPU core hours metric, and run on two large clusters. They have found that the usage of the selected hybrid parallel paradigm helps to markedly reduce data movement overheads and generate the best value in terms of the CPU core hours metric.

memory, 1 TB SATA HardDisk and is running CentOS Linux

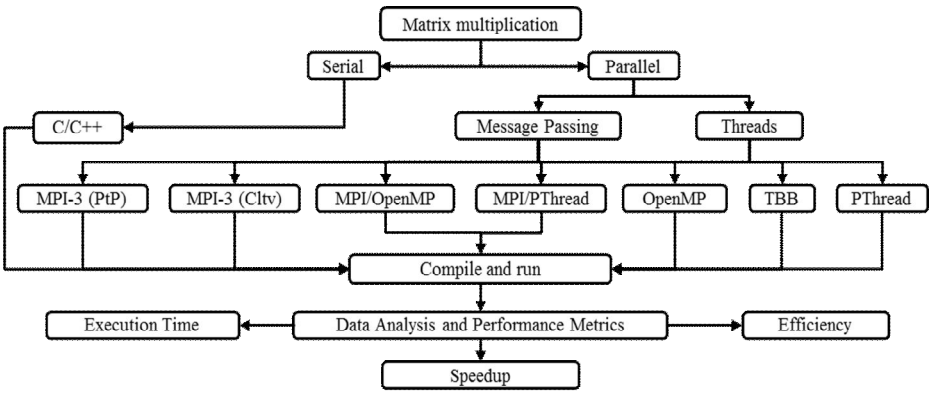

Figure 1: The experiment Framework of the study that shows how the algorithm applied over the parallel models.

4. TEST SETUP

In the current study, seven selected parallel programming paradigms, namely MPI (point-to-point and collective), OpenMP, PThreads, TBB and hybrid (MPI/OpenMP and MPI/PThreads) are compared using matrix multiplication algorithm. In order to measure the programs' performance, a significant number of experiments were performed with various sets of matrices, namely 64, 128, 256, 512 and 1024 (small sized matrix) and for large sized matrices are 2048, 4096 and 5120, also with different number of core processors *P*, namely 2, 4, 8, 16, 32, 48 and 64. To obtain the final results, the average of both matrix sizes are then calculated. Besides that, on the same platform, the test is also performed in both parallel and sequential execution.

4.1 Algorithm

To conduct the experiment, matrix multiplication has been applied as the algorithm, which is a binary operation that consists of a pair of matrices (A and B), that later generates another matrix (C). Three nested loops; an algorithm which has the highest amount of implementation in solving this problem is demonstrated in (1). Each i, j entry is given by multiplying the Aik entries (across row i of A), by the entries Bkj (down column j of B), for k = 1,2,..., m, and summing the results over k.

$$AB_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj} \quad (1)$$

The time required for this algorithm is *O(nmp)*, in asymptotic notation. However, to fit the needs of the study, the algorithm is simplified for the purpose of algorithms analysis, by assuming that the inputs are entirely square matrices with the size of $n \times n$, and run time of $O(n^3)$.

4.2 Experiment platform

The HPC machine used is equipped with an AMD Opteron™ 6272 (4 CPU sockets × 16-core), each working at 2.1 GHz with 16 MB of L3 cache. The machine has 4 × 16 GB of RAM 6.9 with kernel version 4.8. GCC version, 4.8.5, supports the OpenMP 3.1, and the MPI implementation is MPICH3 and TBB 4.4.

4.3 Test framework

In order to compare between the selected parallel programming models, the matrix multiplication has been applied in a sequential way on C/C++ language, which is then ran as the benchmark program. As demonstrated in Figure 1, the benchmark program has been injected and enveloped before being performed on the selected models mentioned earlier. Performing multiply function of matrices in a parallel manner is also the prime component of the parallel operation.

4.4 Performance metrics

Execution time: Execution time carries the meaning of the amount of time used to solve matrix multiplication problem. It is measured by calculating the time period between the program's start and end. Parallel execution time, on the other hand, can be calculated by calculating the time period between the start and end of the parallel computation. Besides that, the average of an enormous number of experiments are calculated and taken as execution time. Execution time also has been taken for sequential program which denoted by TS, while the parallel is denoted by TP.

Speedup: In parallel programming, the speedup of a program is defined to be the proportion of the rate at which work has been done on N processors, to the rate of work which has been done by only one processor as shown in (2) [3]. Efficiency is calculated using the formula based on Admahl's law. In the formula, E represents efficiency, SP for speedup and P is the number of cores used in parallel as we can see in (3) [3].

$$Speedup\ (SP) = \frac{Sequential\ execution\ \ time\ T_s}{Parallel\ execution\ \ time\ T_p} \quad (2)$$

$$Efficiency\ (E) = \frac{SP}{P} \quad (3)$$

5. RESULTS AND ANALYSIS

In this section, we present the experiments results for the performance of the selected parallel programming models. The tested results have been divided into two parts; the sequential implementation and parallel implementation, which the latter is based on the selected performance metrics.

5.1 Sequential implementation

From the results, it has been found that sequential implementation is in its ideal execution time when the matrix size is small (64:1024). In contrast, testing large size of matrix (2048:5120) gives undesirable results that take longer execution time, as demonstrated in Figure 2. This is due to the large size of the matrix which requires more effort to be executed by a sequential program. Thus, the performance of the program is clearly degraded.

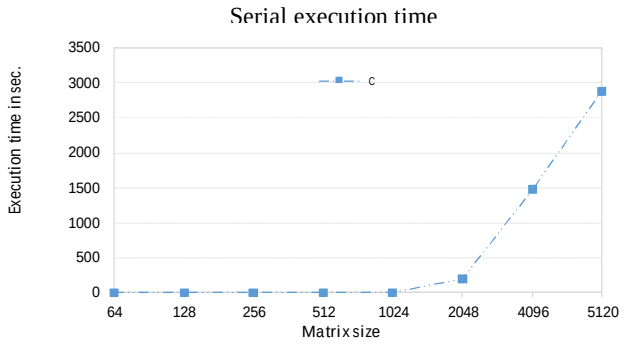

Figure 2: The sequential result of matrix multiplication of matrices from 64 to 5120.

5.2 Parallel implementation

A. Execution time

As can be seen from Figure 3, TBB provides the best performance among all the sizes and it maintains the optimal performance level while increasing the number of processors. Similarly, PThreads also provides efficient performance, and is able to maintain it while increasing the number of threads. Moreover, this study has also identified that MPI (Point-to-Point), and OpenMP perform best with small matrices and both consume lesser time than PThreads. However, the performance of both MPI-Ptp and OpenMP drop with the increasing number of processors or threads. Besides that, from our experiments, none of these models, namely MPI-collective, MPI-OpenMP, and MPI-PThreads are found to be performing well with small matrices, as their overall performance decline with the increasing number of processors. It is also important to note that there are no significant differences that have been found between parallel performance and sequential performance for very small compute sized problems, such as multiplying matrices that are lesser than 128.

B. Speedup

The speedup shows how the parallel computing can minimize the execution time to solve a problem that has high level of complexity [3]. Figure 4 illustrates the speed up of selected parallel programming models with all sizes.

TBB speedups are more prominent from other six models in most cases for all different matrix sizes. OpenMP algorithm behaves similarly to MPI(Point-to-Point) counterparts, as they did not generate steady speedup. Both of them have demonstrated good speedup particularly when the number of processors are less than 32, however, their speedup rate decline with the increasing number of processors and threads. Hybrid (MPI-OpenMP) demonstrated the lowest speedup when compared with the other models in most cases. Besides that, Pthreads speedup has been found to be the best for small matrices, as it can speed the performance up, and is able to outperform TBB when the matrices sizes are 512 and 1024. This is due to the independency of threads management and code. The drawback of parallel programming with small sized problems is that the speedup growth cannot continue with the increasing number of processors.

As shown in Figure 4, the graphical relationship between seven parallel programming models to solve large size problems (2048:5120) using different processors. The speedups continue to rise while increasing the number of processors. From graphs, it's clear that OpenMP and MPI(Point-to-Point) behave similarly and perform well by providing high speedups which are near to the theoretical speedups. TBB and PThreads provide almost same and worse speedups with large size of problems. Hybrid (MPI-PThreads and MPI-OpenMP) are performing well with large size of matrices when the number of processors less than eight and this is because processors' architecture which may affect modules performance. Many things that can affect the performance of TBB and PThreads, cache misses during the experiments, the time taken for creating and terminating threads and tasks.

C. Efficiency

Even though all models have outperformed the sequential execution, their performances are not equal. Thus, we don't know if they are efficient compared with the supplied resources [3]. The communication overhead which may happen during the increasing number of processors affects parallel models' efficiency. This demonstrates that parallelizing of small size problems is not efficient. Both MPI (Point-to-Point) and OpenMP performed better than other models, which leads to their higher level of efficiency with large problem sizes. The efficiency of Hybrid (MPI-PThreads and MPI-OpenMP) models is better with small number of processors, and the overall efficiency is moderate. As demonstrated in Figure 5.

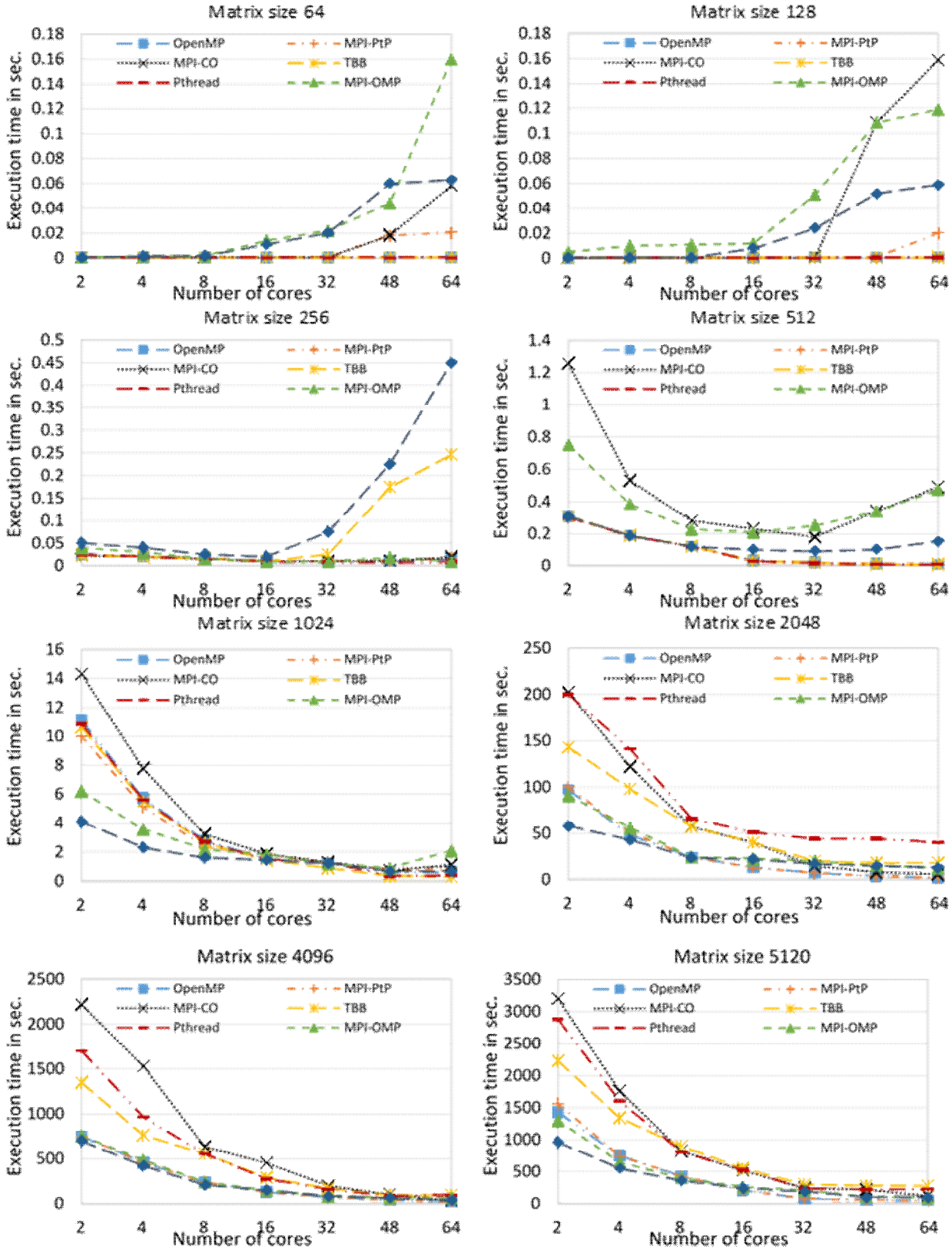

Figure 3: The parallel execution time of the seven paradigms for all chosen sizes of matrix.

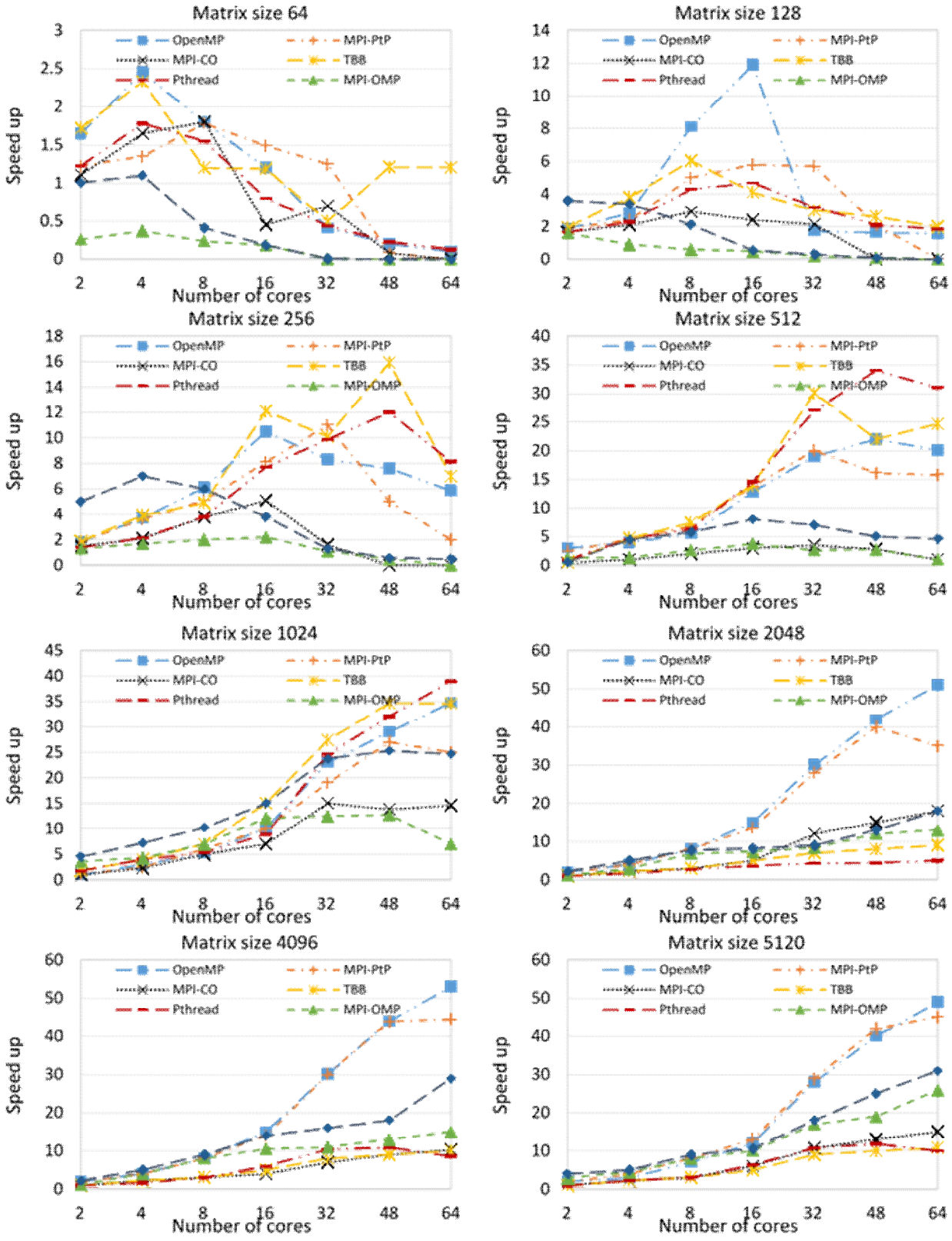

Figure 4: The speedup of the seven paradigms for all chosen sizes of matrix.

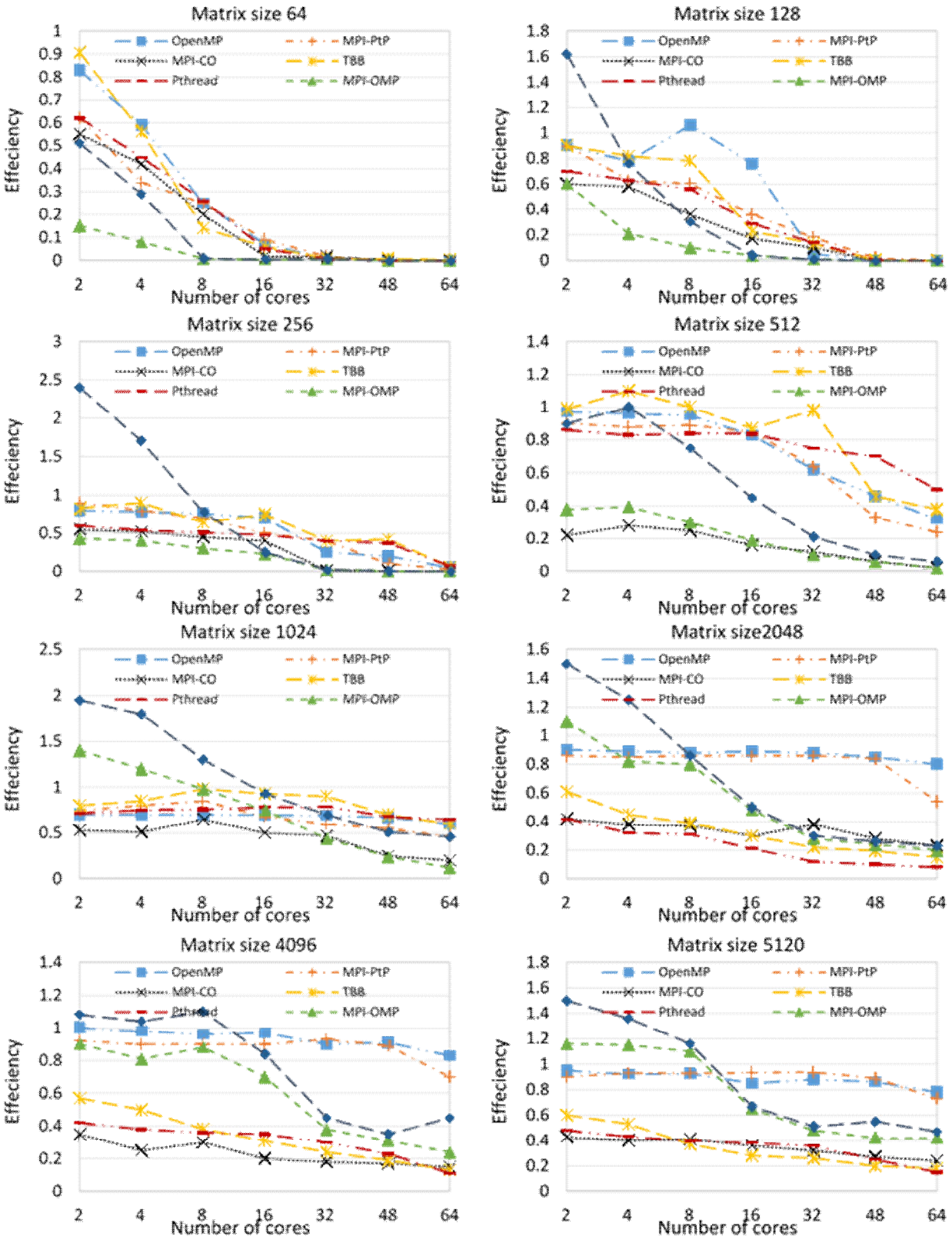

Figure 5: The efficiency of the seven paradigms for all chosen sizes of matrix.

6. FINDINGS

It has been identified that TBB and PThreads are the preferred models to solve small sized problems (matrices), particularly with large number of processors. However, it is impractical to parallelize problems with small sizes, as each model will not be able to perform better than the serial method. In addition, MPI(Point-to-Point) and OpenMP have also been found to be the best models to solve large sized problems (matrices).

7. CONCLUSION AND FUTURE WORK

In this paper, we analysed the performance of seven parallel programming paradigms by exploring the past related works. We then implemented matrix multiplication on the seven selected paradigms as a compute-intensive problem. HPC server with 4 CPU sockets * 16 cores has been used to implement the experiment with different sizes of matrices. The execution time, speedup and efficiency have been recorded for each model as metrics for the test. In addition a computational quantitative comparison of those models has been done in order to determine the best parallel programming model for compute-intensive problems. Based on the previous performance and comparison, it can be concluded that in most cases, OpenMP and MPI(Point-to-Point) are the ideal models to enable compute-intensive problems to benefit from multi-core architecture. Both models provide significant speedup results over other models and sequential implementation. On the other hand, TBB exhibits good performance with low complexity programming and code changes when compared to other models, especially with smaller sized computation problems.

For the future work, there are a number of ways to further extend the current study, namely implementing it on other environments with HPC machine that has higher number of cores. Besides that, it would also be beneficial to further widen the number of parallel programming models and adding additional benchmark algorithms, to further assist future researchers or programmers in choosing the most efficient models for their needs. In addition, future researchers could also examine the factors that affect the performance of TBB and PThreads performance with larger sized problems to further improve the algorithms to raise their performance.

ACKNOWLEDGEMENTS

This research is fully funded by the Universiti Putra Malaysia under the Grant Putra, GP/2017/9569600.

REFERENCES

1. M. U. Ashraf, F. A. Eassa, A. A. Albeshri, and A. Algarni. **Performance and Power Efficient Massive Parallel Computational model for HPC Heterogeneous Exascale Systems**. *IEEE Access*, vol. 6, pp. 1–1, 2018. https://doi.org/10.1109/ACCESS.2018.2823299
2. D. E. Culler, J. P. Singh, and A. Gupta. ***Parallel computer architecture: a hardware/software approach*.** *San Francisco: Morgan Kaufmann Publishers*, 1999.
3. C. A. Navarro, N. Hitschfeld-Kahler, and L. Mateu. **A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures**. *Communications in Computational Physics*, vol. 15, no. 2, pp. 285–329, Feb. 2014. https://doi.org/10.4208/cicp.110113.010813a
4. H. Asaadi, D. Khaldi, and B. Chapman. **A Comparative Survey of the HPC and Big Data Paradigms: Analysis and Experiments.** in *Proc. IEEE International Conference on Cluster Computing*, 2016, pp. 423–432. https://doi.org/10.1109/CLUSTER.2016.21
5. S. Salehian, J. Liu, and Y. Yan. **Comparison of Threading Programming Models**. in *Proc.2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017, pp. 766–774. https://doi.org/10.1109/IPDPSW.2017.141
6. A. Asaduzzaman, F. N. Sibai, and H. El-Sayed. **Performance and power comparisons of MPI Vs Pthread implementations on multicore systems**. in *Proc. 2013 9th International Conference on Innovations in Information Technology (IIT)*, 2013, pp. 1–6. https://doi.org/10.1109/Innovations.2013.6544384
7. Z. Krpic, G. Martinovic, and I. Crnkovic. **Green HPC: MPI vs. OpenMP on a shared memory system**. in *Proc. 2012 MIPRO of the 35th International Convention*, 2012, pp. 246–250.
8. D. Oryspayev, H. M. Aktulga, M. Sosonkina, P. Maris, and J. P. Vary. **Performance analysis of distributed symmetric sparse matrix vector multiplication algorithm for multi-core architectures**. *CONCURR COMPUT-PRACT EXP*, vol. 27, no. 17, pp. 5019–5036, 2015. https://doi.org/10.1002/cpe.3499
9. C.-Y. Chou and K.-T. Chen. **Performance Evaluations of Different Parallel Programming Paradigms for Pennes Bioheat Equations and Navier-Stokes Equations**. in *Proc. 2016 International Computer Symposium (ICS)*, Chiayi, Taiwan, 2016, pp. 503–508. https://doi.org/10.1109/ICS.2016.0106
10. 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 COMPUT*, vol. 37, no. 9, pp. 562–575, Sep. 2011. https://doi.org/10.1016/j.parco.2011.02.002
11. J. Diaz, C. Munoz-Caro, and A. Nino **A Survey of Parallel Programming Models and Tools in the Multi**

and Many-Core Era. *IEEE TRANS PARALL DISTRIB SYS*, vol. 23, no. 8, pp. 1369–1386, Aug. 2012. https://doi.org/10.1109/TPDS.2011.308

12. A. Leist and A. Gilman. **Comparative Analysis of Parallel Programming Models for C++.** in *Proc. The Ninth International Multi-Conference on Computing in the Global Information Technology*, 2014.
13. P. D. Michailidis and K. G. Margaritis. **Scientific computations on multi-core systems using different programming frameworks**. *Applied Numerical Mathematics*, vol. 104, pp. 62–80, Jun. 2016. https://doi.org/10.1016/j.apnum.2014.12.008
14. M. Silven. **Evaluation and Comparison of Programming Frameworks for Shared Memory Multicore Systems,** M.S. thesis, Dept. Comput, Linkopings Univ., Linköping, Sweden 2014.
15. C. G. Kim and Y.-H. Seo. **Parallel JPEG Color Conversion on Multi-Core Processor**. *IJMUE*, vol. 11, pp. 9–16, 2016. https://doi.org/10.14257/ijmue.2016.11.2.02
16. M. Sharma and P. Soni. **Comparative Study of Parallel Programming Models to Compute Complex Algorithm**. *International Journal of Computer Applications*, vol. 96, no. 19, pp. 9–12, Jun. 2014. https://doi.org/10.5120/16900-6961
17. A. Tousimojarad and W. Vanderbauwhede. **Comparison of Three Popular Parallel Programming Models on the Intel Xeon Phi**. in *Proc. Euro-Par 2014: Parallel Processing Workshops*, 2014, pp. 314–325. https://doi.org/10.1007/978-3-319-14313-2_27
18. E. Ajkunic, H. Fatkic, E. Omerovic, K. Talic, and N. Nosovic. **A Comparison of Five Parallel Programming Models for C++**. in *Proc. 2012 MIPRO of the 35th International Convention*, 2012, pp. 1780–1784.
19. Amjad Kotobi, Nor Asilah Wati Abdul Hamid, Mohamed Othman, and Masnida Hussin. **Performance Analysis of Hybrid OpenMP/MPI Based on Multi-Core Cluster Architecture**. In Compu tational Science and Technology (ICCST), 2014 International Conference on, pages 1–6. IEEE, 2014.