Indonesian Journal of Electrical Engineering and Computer Science
Vol. 16, No. 3, December 2019, pp. 1577~1583
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v16.i3.pp1577-1583

Performance comparison of Java based parallel programming models

Muhammad Na'im Fikri Jamaluddin[1], Azlan Ismail[2], Amir Abd Rashid[3], Talha Takleh Omar Takleh[4]
[1,2,3,4]Department of Computer Science, Faculty of Computer and Mathematical Sciences,
Universiti Teknologi Mara (UiTM), Malaysia
[2]Knowledge and Software Engineering Research Group, Universiti Teknologi Mara (UiTM), Malaysia

Article Info

Article history:

Received Apr 1, 2019
Revised Jul 22, 2019
Accepted Jul 28, 2019

Keywords:

Hybrid
Java
Message passing
MPJ express
Multithreading
Parallel programming

ABSTRACT

Parallel programming has been implemented in many areas to solve various computational problem with the aim, to improve the performance and scalability of the software application. There are a few parallel programming models commonly used, namely, threads, and message passing (distributed) models. Furthermore, various APIs have been proposed to implement these models based on two popular languages, notably, C/C++ and Java. A few studies have been done to compare the performance of parallel programming models, specifically, pure versus hybrid model. However, most of existing comparisons targeted on MPI/OpenMP based on C/C++ language. In this paper, our aim is to explore the performance comparison between threads, message passing and hybrid model in Java, specifically using Java multithreading and MPJ Express. For this reason, we have chosen a problem called word count occurrence which is significant in Natural Language Processing and use it to design and implement the parallel programs. We then present their performance and discuss the results.

Corresponding Author:

Azlan Ismail,
Department of Computer Science, Faculty of Computer and Mathematical Sciences,
Universiti Teknologi MARA (UiTM), 40450 Shah Alam, Selangor, Malaysia.
Email: azlanismail@tmsk.uitm.edu.my

1. INTRODUCTION

The concept of parallel processing is mainly to improve the performance of a program by utilizing the resources that the computer or system has (multicore or distributed environment) [1-2]. In a serial programming model, the task of running a program will fall only to a single core. Normally, a single core can still be fast enough, but in today's technological environment which deals with massive amount of data, a single core approach is inefficient [3]. Thus, parallel programming model is more promising to efficiently compute this big data [4-5].

There are a few parallel programming models that can be applied to design and develop parallel program to solve data-intensive computational problem. Herein, we focus on three of them, namely, threads, message passing and hybrid model. Furthermore, there are standards and APIs developed to assists the implementation of these models. For instance, OpenMP and Java multithreading for the threads model, and MPI and MPJ for message passing model. Each model and its implementation has their advantages and disadvantages. Thus, a few studies have been done to provide the insights on the performance of each models from different perspectives. From these studies, the comparison of these models, notably, threads, message passing and hybrid in Java perspective is still lacking. We believe, a comparison study is significant to provide some insights for future work.

Therefore, in this paper we contribute to the design of parallel program for each model based on a problem, word count occurrence problem that is significant in Natural Language Processing. Furthermore, we

present the results of performance comparison between these parallel programming models. The next section describes the fundamental concepts and related works. We then discuss the design and implementation of the parallel programs followed by the presentation and discussion of the results. Finally, we conclude our work.

2. PRILIMINARY

In this section, we explain the scope of parallel programming models especially in Java perspective as well as related works.

2.1. Parallel Programming Models

There are a few parallel programming models commonly used to address various computational problems [6]. In this paper, we are interested in three of them, namely, the threads, message passing, and hybrid model. Each of them is briefly explained as follows.

Threads model is a kind of shared memory model that consists of threads (also known as lightweight process). Each thread can share their data through the shared space. Thus, the threads communicate between each other implicitly. Threaded programming is not new and there are a few standards and languages to support the implementation of this model that includes OpenMP and Java multithreading. This model is generally easier to be implemented. The performance overhead may come from several factors, especially the bottleneck of accessing the same space.

Message passing (distributed) model consists of multiple processes where each process resides in different machine, physically or virtually. Each process has their own memory space, and thus this model is needed when there is no global and shared space support. The data exchange has to be done through message passing. The implementation of this model has been supported by a Message Passing Interface (MPI) standard [7]. In general, this model supports scalability of parallel program. However, the implementation of this model requires additional programming effort on the communication aspect. Furthermore, the performance overhead may be caused by inappropriate design of communication approach.

Hybrid model combines more than one of the available parallel programming models, such as, message passing and threads model. Conceptually, there are multiple processes which can perform message passing between each other. In addition, within the local environment of each process, there are multiple threads to support the parallel execution. The implementation of hybrid model is quite challenging, but in general, the performance can be potentially improved as compared to a pure/single model.

2.2. Threads in Java

Java language supports the implementation of threads model through multi-threaded programming. A thread in Java in represented as an object and has a common life cycle notably, created, started, running, waiting, and terminated. All threads share the same global memory which enable every thread to access the same space. This capability enables the threads to exchange data, whenever needed. To protect certain data from a problematic condition such as data inconsistency caused by memory interference, Java supports the implementation of synchronization. Threads have traditionally been used on single processor systems. With the advent of multicore and symmetric multiprocessor (SMP) systems, threads can be mapped to physically parallel processing hardware [8].

2.3. Massage Passing in Java

There have been several implementations of java messaging libraries especially for developing parallel program. The libraries are driven by different approaches, notably, based on Java Remote Method Invocation (RMI), based on the wrapper libraries that resort to Message Passing Standard through Java Native Interface (JNI) such as mpiJava [9], MPJ Express [10], MPJ/Ibis [11], or based on low-level Java sockets such as MPJava [12], JFS [13], and F-MPJ [14]. Each of them has their own advantages and disadvantages. In this paper, we are interested in MPJ Express.

MPJ Express is a thread-safe message passing library that provides a full implementation of the MPI in Java language. It allows application developers to write and execute parallel applications for multiple cores processors and computing clusters. The design of MPJ Express consists of several layers that includes the MPJ API layer (highest layer), collective and point to point communications API (next lower layers), and mpjdev and xdec levels (next lower layers) for actual communications and interaction with the underlying networking hardware. MPJ Express has been utilized in a few domains, such as machine translation for natural language processing [15], and clustering scalability [16].

3. RELATED WORK

Many studies have done to evaluate the performance of parallel programming models. In Rabenseifner et al. [17], they investigated the performance of pure MPI and OpenMP, and the combination of both as a hybrid model. They utilized a mapping problem of a two-dimensional Cartesian domain decomposition for implementing the parallel program. The work by Jin et al. [18] also addressed the same concern to evaluate performance of hybrid MPI and OpenMP, and the pure models. They utilized multi-zone versions of NAS Parallel Benchmarks for the evaluation. Their finding stated that the hybrid model can reduce the memory footprint and overhead associated with MPI calls and buffers and improve load balance. However, as they stated, the hybrid performance is affected due to no well-defined interaction between MPI processes and OpenMP thread. An example of hybrid (MPI and threads model) implementation can be referred to Khaitan et al. [19] that applied for massive contingency analysis in power systems. In particular, the hybrid model is used to facilitate the computation of non-blocking of work-stealing based scheduling algorithm. Baños et al. [20] investigated the implementation of MPI and OpenMP for parallelizing population-based meta-heuristics and evaluated their performances. They have shown that OpenMP is better when the number of threads are more than the available cores. Meanwhile, MPI outperforms OpenMP when the input size is sufficiently large. The work by Rao et al. [21] compared the performance of Cross Memory Attach (CMA) capable, MPI-based approach with their proposed fine-tuned multithreading approach. The study has shown that message-passing can outperform multithreading in certain scenarios. In Jiao et al. [22], they studied the performance comparison between pure MPI, pure OpenMP and hybrid (i.e. MPI + OpenMP) in addressing computational efficiency for the spherical discontinuous deformation analysis (SDDA). They have shown that the proposed hybrid model is correct and effective.

A few studies have attempted to address the performance comparison in Java environment. For instance, there are studies that evaluated the performance and scalability issues of multithreaded Java applications by focusing the impact on the microarchitecture Luo et al. [23] and on the multicores systems Kuo-Yi Chen et al. [24]. In terms of performance comparison, the work by Shafi et al. [25] compared the performance of message passing using Java (i.e. MPJ Express) and C languages (i.e.MPI) and revealed a comparable performance and they both can scale in a similar fashion. In comparison to these works, we address the performance comparison of three models, multithreading, message passing, and hybrid programming models.

4. RUNNING EXAMPLE

In this paper, we utilize the word count occurrence problem as the running example to evaluate the parallel programs' performances as shown in Figure 1. The aim of problem solving is to count the frequency of occurrence of each word in a document. The counted values are one of many tasks for supporting different kind of text analysis, such as sentiment analysis [24-26]. In this section, we only focus on the word count occurrence algorithm and presented it as a serial prog ram. The program consists of three core tasks, namely:

a. Load and store (TS1) – The program loads or reads data from the external source (e.g. text file) into the memory. In particular, it reads line-by-line as a String and store them into an array of list.
b. Split/chunk and store (TS2) – The program then split each line of String into a chunk of words and store them into another memory, a hash map. The reason of utilizing the hash map (i.e. key and value) is to keep the frequency of occurrence for each word, where key refers to the extracted word and value is its frequency.
c. Aggregate result (TS3) – The program aggregates the total count for each similar word. Finally, the total count for each word is displayed.

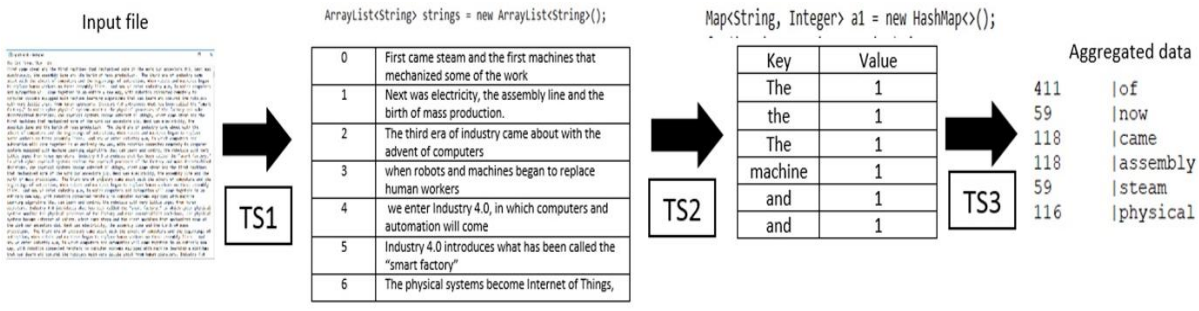

Figure 1. A Process Flow of Word Count Occurrence

5. RESEARCH METHOD

The comparison of these models is done through an experimentation. For this reason, we design and implement three parallel programs which can solve the word count occurrence mentioned earlier. Thus, in this section, we present our design of the parallel programs.

5.1. Multithreading Program

The design of this program follows master-slave approach which involves two types of threads, the main thread and the job threads, as shown in Figure 2(A). The main thread is responsible to perform TS1 and TS3. As for the parallelism, it also partitions the input file (i.e. into a series of lines that contains multiple words), coordinate the life cycle of job threads, and assign the required job. The job thread is responsible to execute TS2. The data sharing between the main and job threads is done through the global memory. The construction of this program is based on Java multithreading. We utilized *Thread* class for the threads implementation.

5.2. 5.2 Message Passing Program

The design of this program follows peer-to-peer approach, where there is no main coordinator/controller as shown in Figure 2(B). In this case, each process executes TS1 and TS2 on a region from the document. For this reason, we define a partitioning function that determines the region for each process. Furthermore, only one process (i.e. P0) executes TS3. As there is no global memory, explicit communication has to be implemented as each process needs to send their results of TS2 to P0. We utilize high level communication API of MPJ Express, called *MPI.COMM_WORLD.Reduce()* to communicate and aggregate the results.

5.3. 5.3 Hybrid Program

The design of this program combines the previous two designs based on hierarchical approach as shown in Figure 2(C). The higher level refers to message passing model, while the lower level refers to threads model. Furthermore, the higher level processes are meant to execute TS1 and TS3. They take their own region of data from the document as defined in the main partitioning formula. In addition, they partition and map their region into multiple threads. The threads are required to execute TS2. Each higher level process has a global memory to be shared among threads. The construction of this program is based on the combination of MPJ Express and Java multithreading.

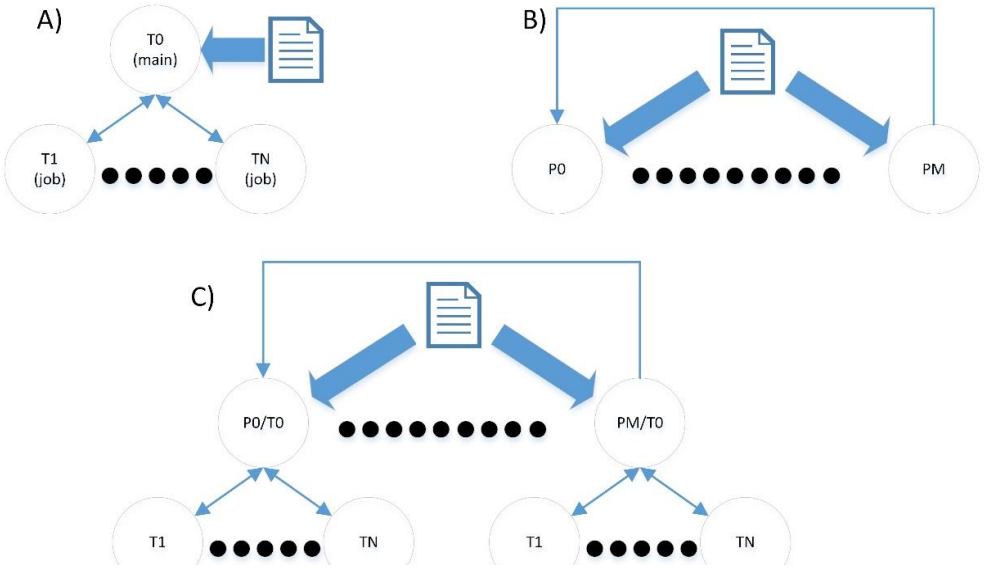

Figure 2. Illustration of multithreading (A), message passing (B), and hybrid implementation (C)

6. RESULTS AND DISCUSSION

In this section, we begin with the description of our experiment setup followed by the presentation and discussion of the results.

6.1. Experiment Setup

Our experiments were conducted on a Linux-based virtual machine with 10 Cores and 9GB of memory. We utilized eclipse Oxygen 2 to run the programs. We used three different size of input files, 1MB, 5MB, and 10MB for each parallel program. Furthermore, we focused on 2, 4 and 8 number of threads/processes. In the case of hybrid program, we set for 2, 4, and 8 threads for each 2, 4, and 8 processes. Based on these choices, we executed 3 groups of possible configurations as shown in Table 1. We run 10 times of each configuration to obtain the mean value. We logged the execution time of each program for presenting the results.

Table 1. The configuration used for the experiment

Conf.	File size	Thread num.	Process num.	Process + thread num.
1	1MB	2t	2p	2p+2t, 4p+2t, 8p+2t
		4t	4p	2p+4t, 4p+4t, 8p+4t
		8t	8p	2p+8t, 4p+8t, 8p+8t
2	5MB	2t	2p	2p+2t, 4p+2t, 8p+2t
		4t	4p	2p+4t, 4p+4t, 8p+4t
		8t	8p	2p+8t, 4p+8t, 8p+8t
3	10MB	2t	2p	2p+2t, 4p+2t, 8p+2t
		4t	4p	2p+4t, 4p+4t, 8p+4t
		8t	8p	2p+8t, 4p+8t, 8p+8t

6.2. Performance Comparison

In general, all programming models illustrate a performance improvement with increasing number of computing node (i.e. from 2 to 8 nodes), especially for the threads and message passing model. The threads model outperforms the other models when the input file size is 1MB. However, message passing model outperforms the others when the file size is increased. The fact that the threads model is degrading when the file size is increasing may be caused by the master-slave concept, where the main thread has to do the partitioning and distribution, as well as aggregating the results. Meanwhile, the good performance of message passing model may be due to peer-to-peer concept, where each computing node obtains the same file and only read the required region (i.e. based on predefined partition) assigned to them. The performance of hybrid model is slower for 1MB and 5MB of file sizes. However, the hybrid model with 8 process and 2 threads outperforms the other models. Having said that, the hybrid model does not show a significant difference with increasing number of computing node, especially when the file size is 10MB. This result may be caused by two layers partitioning of the input data. In addition, it also shows that increasing the computing node of the second layer (the thread number) does not give much benefit to the hybrid model. Figure 3 show the performance comparison based on 1MB, 5MB and 10MB.

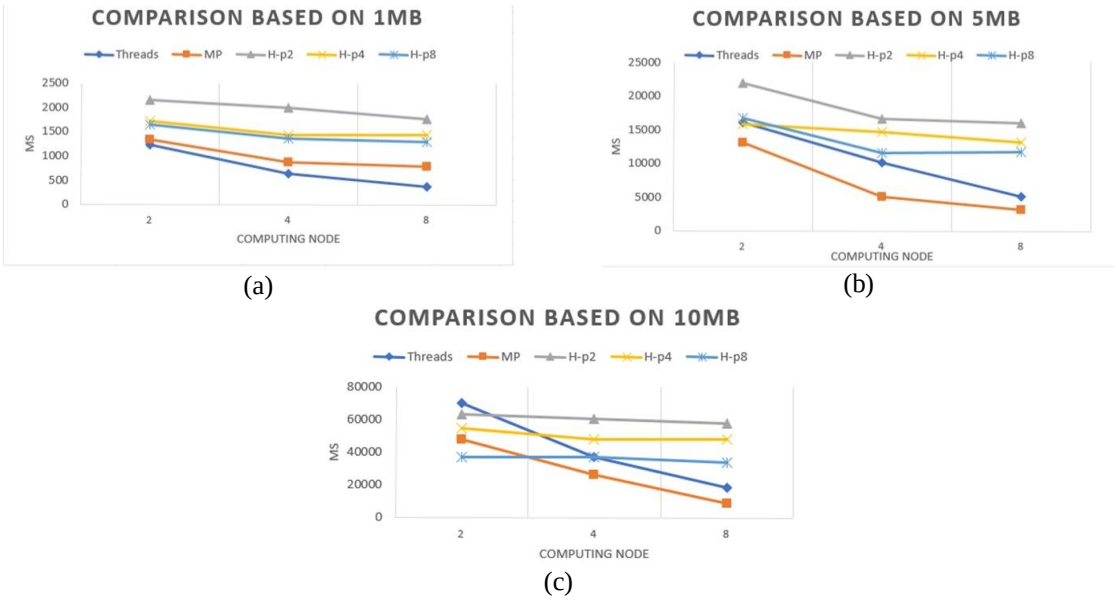

Figure 3. Performance comparison: based on (a) 1MB, (b) 5MB and (c) 10MB

7. CONCLUSION

In this paper, we have presented the design of parallel programs based on three types of parallel programming models, namely, threads, message passing and hybrid model. We also have implemented these models using MPJ Express and Java multithreading and combination both. The implementation aimed to solve the word count occurrence problem that is significant in NLP research area. We have conducted a series of experiments to evaluate the performance of each program. We made a comparison of their execution time in relation to the size of input files and the number of threads and/or processes. The results have shown that the threads model is better when the input size is smaller, whilst the message passing model outperformed the others when the file size is doubled. The hybrid model is a bit slower and does not show a significant increase in terms of performance. For future work, we plan to use different computational problem that requires heavier computation with massive data as well as GPU support.

ACKNOWLEDGEMENTS

This research has been supported by a Research Grant, 600-IRMI/ PERDANA 5/3 BESTARI (048/2018), funded by Universiti Teknologi MARA, Shah Alam, Selangor, Malaysia.

REFERENCES

[1] O. Edelstein, et al., "Multithreaded Java program test generation," *IBM Syst. J.*, vol. 41, pp. 111-125, 2002.
[2] Z. N. Rashid, et al., "Distributed Cloud Computing and Distributed Parallel Computing: A Review," *2018 International Conference on Advanced Science and Engineering (ICOASE)*, pp. 167-172, 2018.
[3] D. L. Bruening and J. Chapin, "Systematic Testing of Multithreaded Java Programs," *Dep. Electr. Eng. Comput. Sci.*, pp. 150, 1999.
[4] A. Ahmad, et al., "Multilevel Data Processing Using Parallel Algorithms for Analyzing Big Data in High-Performance Computing," *Int. J. Parallel Program.*, vol. 46, pp. 508-527, 2018.
[5] C. Shen, et al., "Performance prediction of parallel computing models to analyze cloud-based big data applications," *Cluster Comput.*, vol. 21, pp. 1439-1454, 2018.
[6] Blaise Barney, "Introduction to Parallel Computing," 2018. Available: https://computing.llnl.gov/tutorials/parallel_comp/#MemoryArch.
[7] E. Lusk, et al., "MPI: A message-passing interface standard," *Int. J. Supercomput. Appl.*, vol. 8, pp. 623, 2009.
[8] D. G. Waddington, et al., "Dynamic Analysis and Profiling of Multithreaded Systems," *Designing Software-Intensive Systems*, IGI Global, pp. 290-334, 2009.
[9] M. Baker, et al., "mpiJava: An object-oriented java interface to MPI," Springer, Berlin, Heidelberg, pp. 748-762, 1999.
[10] M. Baker, et al., "MPJ Express: Towards Thread Safe Java HPC," *2006 IEEE International Conference on Cluster Computing*, pp. 1-10, 2006.
[11] M. Bornemann, et al., "MPJ/Ibis: A Flexible and Efficient Message Passing Platform for Java," Springer, Berlin, Heidelberg, pp. 217-224, 2005.
[12] W. Pugh and J. Spacco, "MPJava: High-Performance Message Passing in Java Using Java.nio," Springer, Berlin, Heidelberg, pp. 323-339, 2004.
[13] G. L. Taboada, et al., "Java Fast Sockets: Enabling high-speed Java communications on high performance clusters," *Comput. Commun.*, vol. 31, pp. 4049-4059, 2008.
[14] G. L. Taboada, et al., "F-MPJ: scalable Java message-passing communications on parallel systems," *J. Supercomput.*, vol. 60, pp. 117-140, 2012.
[15] A. Tomar, et al., "Parallel implementation of machine translation using MPJ Express," *2013 National Conference on Parallel Computing Technologies (PARCOMPTECH)*, pp. 1-5, 2013.
[16] D. Pettinger and G. Di Fatta, "Scalability of efficient parallel K-Means," *2009 5th IEEE International Conference on E-Science Workshops*, pp. 96-101, 2009.
[17] R. Rabenseifner, et al., "Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes," *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pp. 427-436, 2009.
[18] H. Jin, et al., "High performance computing using MPI and OpenMP on multi-core parallel systems," *Parallel Comput.*, vol. 37, pp. 562-575, 2011.
[19] S. K. Khaitan and J. D. McCalley, "SCALE: A hybrid MPI and multithreading based work stealing approach for massive contingency analysis in power systems," *Electr. Power Syst. Res.*, vol. 114, pp. 118-125, 2014.
[20] R. Baños, J. Ortega, C. Gil, F. de Toro, and M. G. Montoya, "Analysis of OpenMP and MPI implementations of meta-heuristics for vehicle routing problems," *Appl. Soft Comput.*, vol. 43, pp. 262–275, Jun. 2016.
[21] D. M. Rao and D. M., "Performance comparison of Cross Memory Attach capable MPI vs. Multithreaded Optimistic Parallel Simulations," *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation - SIGSIM-PADS '18*, pp. 37-48, 2018.
[22] Y. Y. Jiao, et al., "A hybrid MPI/OpenMP parallel computing model for spherical discontinuous deformation analysis," *Comput. Geotech.*, vol. 106, pp. 217-227, 2019.

[23] Y. Luo, et al., "Workload Characterization of Multithreaded Java Servers," *IEEE Int. Symp. Perform. Anal. Syst. Softw.*, pp. 128-136, 2001.
[24] K. Y. Chen, et al., "Multithreading in Java: Performance and Scalability on Multicore Systems," *IEEE Trans. Comput.*, vol. 60, pp. 1521-1534, 2011.
[25] A. Shafi, et al., "A comparative study of Java and C performance in two large-scale parallel applications," *Concurr. Comput. Pract. Exp.*, vol. 21, pp. 1882-1906, 2009.
[26] B. Liu, "Sentiment Analysis and Opinion Mining," *Synth. Lect. Hum. Lang. Technol.*, vol. 5, pp. 1-167, 2012.

BIOGRAPHIES OF AUTHORS

Muhammad Na'im Fikri Bin Jamaluddin received his Bachelor Degree in Netcentric Computing in 2017 from Universiti Teknologi Mara (UiTM), Malaysia. He continued his study in Master of Computer Science in UiTM, Shah Alam. His research interests are in the area of computer architecture, machine learning, problem solving algorithm and parallel programming.

Azlan Ismail is currently working as a senior lecturer at Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Shah Alam, Selangor. He received his PhD in Informatics in 2012 from University of Wollongong (UoW), Australia. His research interests are related to cloud computing, service oriented computing, parallel computing and self-adaptive systems. He is a member of Knowledge and Software Engineering research group (UiTM), IAENG and SIGSOFT.

Amir Abd Rashid recieved his Bachelor Degree in Intelligent System in 2017 from Universiti Teknologi MARA (UiTM), Malaysia. He is currently hold position as software engineer at one of international company Experian Malaysia. He also has obsession in AI technology and continiously doing RnD as his personal interest.

Talha Takleh Omar Takleh received his Bachelor Degree in Electrical Engineering in 2014 from Purdue University, Indiana, USA. He is currently a graduate research assistant and postgraduate student pursuing Master's in Computer Science in Faculty of Computer and Mathematical Sciences (FSKM), Universiti Teknologi MARA (UiTM), Shah Alam, Selangor, Malaysia. He is currently involving in a project that studies the simultaneous localization and mapping (SLAM) for UiTM autonomous vehicle project.