International Journal of Engineering & Technology, *8 (1.7) (2019) 75-81*

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Parallel Self-Organizing Map Using Heterogeneous Uniform Memory Access

Muhammad Firdaus Mustapha[1]*, Noor Elaiza Abd Khalid[2], Azlan Ismail[3]

[1]*Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA Cawangan Kelantan, Malaysia*
[2,3]*Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA Shah Alam, Malaysia*
**Corresponding author E-mail: firdaus19@gmail.com*

Abstract

Self-organizing Map (SOM) is a useful data analysis method that consists of complex calculations. Numerous researchers have successfully improved online SOM processing speed using Heterogeneous Computing (HC). HC is a combination of Central Processing Unit (CPU) and Graphic Processing Unit (GPU) that work closely together by accessing separate memory blocks. Although the standard HC has an excellent performance, performing online SOM variant will cause computer hardware underutilized when number of neurons in SOM map is smaller than number of cores. Moreover, complexities of steps in SOM algorithm drive to increase the usage of high memory capacity. Nevertheless, this situation leads to communication latency that caused by “deep copies” method. Recently, Heterogeneous Uniform Memory Access (HUMA) platform has a noteworthy parallel processing capability. Therefore, this paper presents parallel SOM using HUMA platform and multiple stimuli approach. In this study, the main goal is to reduce computation time of SOM training via implementing parallel SOM on HUMA platform and increase GPU cores utilization. This study employed dataset from UCI repository. As a result, the proposed work was able to score up to 1.23 of speed up overall for large map size compared to standard parallel SOM. Hence, the proposed work is able to suggest a greater solution for small to medium sized of data analysis software.

Keywords: *Graphic Processing Unit; Heterogeneous Computing; Heterogeneous Uniform Memory Access; Parallel Computing; Parallel SOM.*

1. Introduction

SOM is a type of Artificial Neural Network (ANN) that is useful for solving the problem of data exploration and clustering in many domain areas [1]–[3]. It has great abilities to deal with missing values, remove noise, and outliers [4]. Although SOM has an excellent performance, it still has problems on slow processing when visualizing large map sizes [5]. This heavy workload applies to processor when it implements certain steps in SOM such as find best matching unit (BMU) and update weight [4]. In addition, dataset dimensions have a high impact on SOM processing [6]. Generally, the larger dataset dimensions, the longer time is taken to complete SOM processing.
This issue attracts many researchers to improve SOM processing by implementing SOM algorithm in parallel whether by using network (map) partitioning [7]–[9] or data (example) partitioning [10]–[12] or combining both data and network partitioning [13], [14]. Network partitioning is usually used with the original Kohonen SOM algorithm known as online SOM training in order to maintain the training sequentially with the map updates [4], [15]. The parallel execution is normally done by assigning each neuron on map per parallel thread. On the other hand, data partitioning is implemented on modified version of SOM called batch SOM training [4], [15]. In data partitioning, parallel execution is done by dividing the data amongst the individual threads. The implementation of data partitioning method was more popular among researchers. One of the reasons, it is straight forward implemention of batch SOM algorithm in parallel where each data can be mapped per processing element [12], [16], [17]. Through it, the processing speed could be assured compared to network partitioning. However, there was some issues had been reported regarding to batch training which related to quality of results [18]. Consequently, several research works attempted to combine both partitioning in order to gain the benefits of both methods [13], [14].
Meanwhile, most researchers who have suggested SOM improvements in their works are targeted to achieve scalability and efficiency. The proposed parallel SOMs that efficient should be faster in terms of processing compared to the earlier versions of parallel SOM [19]–[21]. In term of scalability, the proposed parallel SOMs should able to handle variety size of dataset and also maintain scalability of hardware. Many research works were found in the literature parallelized SOM with the interest to increase performance through increasing the utilization of hardware [22]. Greater utilization of processing elements will lead to greater speed up performance [23]. Besides, a few research works presented more inputs or stimuli on the SOM in order to improve hardware utilization [24], [25]. Therefore, this study will concern on implementing parallel SOM that combine both data and network partitioning. This method could gain benefit of original online SOM algorithm which assures in terms of accuracy of results and also gain benefit from batch SOM algorithm that has proven in terms of processing speed.
Furthermore, there were many versions of task decomposition found to parallelize SOM algorithm. Four major steps in SOM algorithm have been decomposed with the interest to run in parallel namely initialize weights, calculate distance, find BMU, and update weights. Almost all the researchers parallelized calculate distance and find BMU steps [5], [16], [22], [26], [27]. Many of them also parallelized update weights step and occasionally found in the literature that parallelized initialize weights step [22], [27], [28]. Therefore, this study

will concern to parallelize these three steps; calculate distance, find BMU, and update weights. These steps involve a large number of repeating calculation which suitable to implement in parallel.

On the other hand, to improve the SOM algorithm, Graphic Processing Unit (GPU) variant shows better performance in terms of computation time for large data than Central Processing Unit (CPU) variant [14], [22], [29]. GPU is a many core processors comprising hundreds or even thousands of compute cores [30]. Introduction to GPU programming frameworks make GPU more popular to design parallel algorithm for high speed processing. Example of GPU programming frameworks are Open Computing Language (OpenCL) and Compute Unified Device Architecture (CUDA) [31]. Basically, GPU is an accelerator to CPU that has been widely used for processing scientific data and graphic purposes. Combination of GPU and CPU that work together produces a paradigm known as Heterogeneous Computing (HC) [32].

HC is one of the popular platforms for executing parallel SOM for both SOM variants; online SOM and batch SOM training [16], [22]. Although standard HC has an excellent performance, performing online SOM training on many cores architecture will cause underutilized computer hardware. Specifically, this condition occurs when the number of neurons on map is smaller than the number of cores [14], [22]. Therefore, several research works implemented batch SOM training by using data partitioning in order to fully utilize the cores of processor. Moreover, [6] found that when SOM processing was conducted in parallel with high dataset dimensions and larger map sizes, it would significantly slow down the SOM processing even by using combination of GPU and CPU. Numerous researchers agreed that when SOM is executing on HC, it shows a significant increase in processing speed especially for large data compared to execute on CPU only [14], [28], [29]. Additionally, due to the drawback of earlier HC architectures, many frameworks used the GPU as an accelerator that can only work under CPU control. Thus, protocol in communication between GPU and CPU is a source of high communication latency that finally causes bottleneck. This problem is resolved by using distributed memory of HC where the memory management needs to be configured manually by the programmer [33].

The latest technology on hardware design for HC system known as Heterogeneous System Architecture (HSA) is a single integrated unify GPU and CPU circuit chip [33], [34]. This offers a unified programming platform which reduces communication latency between GPU and CPU and also eliminates programmability barrier. The HSA specification has been implemented by major microprocessor manufacturers such as AMD and Intel. AMD has introduced Heterogeneous Uniform Memory Access (HUMA) technology that support HSA specification [35]. Meanwhile, Intel also has introduced HSA compliant microprocessor in their sixth-generation product line with codename Intel Skylake [36]. Moreover, the introduction of GPU programming framework like OpenCL 2.0 in 2013 that supports HSA specification become a complement to HSA enabled microprocessors which support in term of software [37]. One of the features in OpenCL 2.0 is Shared Virtual Memory (SVM). SVM allows the device and the host to share a common virtual address range [37]. SVM also can reduce overhead by removing deep copies during transferring data between host and device. Deep copies involve complete duplicating objects in the memory hence reduce redundancies [33]. Developing a computer program using both hardware and software HSA compliant is able to enhance communication between GPU and CPU by authorizing the GPU to manage its own resources and access some of the resources from the CPU.

From the above discussion, this paper presents parallel SOM using HUMA to solve two main problems. The first problem is communication latency when implementing SOM on earlier version of HC. Secondly, the problem is underutilized cores in GPU when employing online SOM training. In this study, we propose an enhanced parallel SOM architecture which combines network and data partitioning parallel methods and adapts multiple stimuli approach. The main purpose of this study is to reduce computation time of SOM training via implementing parallel SOM on HUMA platform and increase GPU cores utilization. This study conducts an experiment to evaluate the performance of the proposed work in terms of efficiency and scalability which relates to processing time. As a result, the proposed work has performed better for both efficiency and scalability compared to the previous work.

Therefore, this paper is prepared as follows. Section 2 describes about the proposed work on enhanced parallel SOM using HUMA. Next, Section 3 explains about research method for this study while Section 4 elaborates about the result and discussion. Finally, Section 5 provides conclusion and future research on parallel SOM.

2. Proposed method

For the solution, this study proposes an enhanced parallel SOM architecture as depicted in Figure 1. The proposed work applies data parallelism which executing on three different steps of SOM; calculate distance, find BMU and update weights. With the interest to increase GPU utilization, this study applies task parallelism to execute duo stimuli for both calculate distance and find BMU steps. The purpose of execution of parallel duo stimuli is to reduce computation time for small map. This issue has been highlighted by the previous researchers [14], [22]. Moreover, the proposed work is based on fined-grained SVM buffers in OpenCL 2.0 platform which can reduce communication latency between GPU and CPU because they are synchronized during the execution of the SVM buffer.

In details, the proposed architecture in Figure 1 consists of two parallel kernels for calculate distance step, two parallel kernels for find BMU step and a kernel for update weight step. The design of parallel kernels is realized by applying multiple stimuli into the proposed architecture. The main purpose of running the kernels in parallel is to increase utilization of work units in GPU. OpenCL 2.0 is used in the proposed work because it provides a general purpose of parallel programming interface for supporting GPU hardware. By using OpenCL 2.0, it allows a programmer to produce many queues for execution. The queues are processed based on out-of-order execution where no guarantee that the enqueued commands will be completed in sequences [37].

The first parallel kernel comprises of Calc_Dist_Kernel_1 (CDK1) and Calc_Dist_Kernel_2 (CDK2). The function of this kernel is to calculate the distance between current input vector and neurons by using Manhattan distance calculation, *MD*. Then, the distance values are stored into an array. Figure 2 depicts algorithm for distance calculation where, x is an input vector with n data and, Wj is neurons vector with m weights. In the meantime, the second parallel kernel comprises of Find_BMU_Kernel_1 (FBK1) and Find_BMU_Kernel_2 (FBK2). This kernel searches for BMU in parallel by applying two stages of parallel reduction method. The BMU is a calculation to choose a winner neuron that has the smallest distance among the neurons. The implementation of kernels in every parallel kernel may overlap each other. Lastly, the third kernel updates weight of neurons based on neighbourhood function and learning rate. Figure 3 shows algorithm for update weight steps where W is neuron's weight and B is map length. The algorithm begins with determining the neighbourhood value which includes learning rate. The BMU and its close neighbours will be changed the most, while other neurons are changed the least. This study applies Gaussian function as a neighbourhood function as shown in Equation (2).

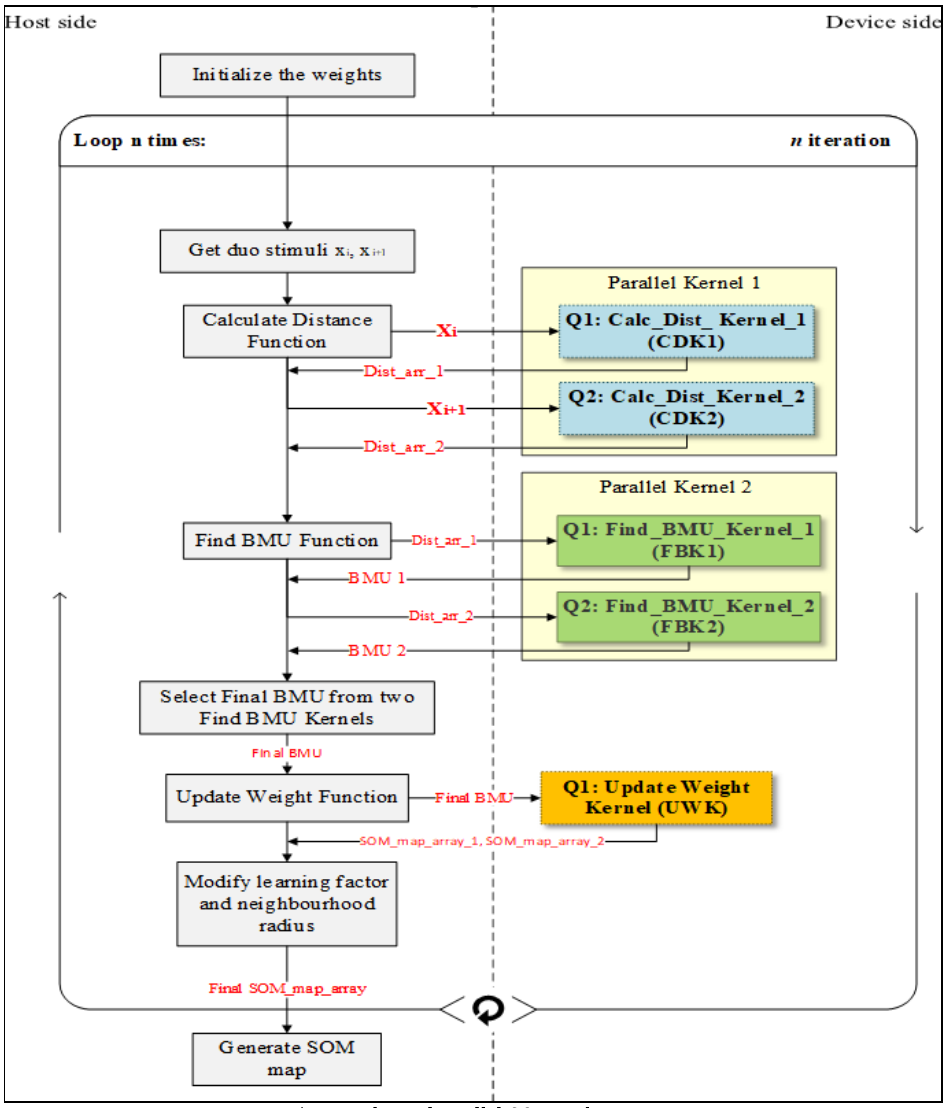
Host side
Device side
Initialize the weights
Loop n times:
n iteration
Get duo stimuli xi, xi+1
Calculate Distance Function
Xi
Dist_arr_1
Xi+1
Dist_arr_2
Parallel Kernel 1
Q1: Calc_Dist_Kernel_1 (CDK1)
Q2: Calc_Dist_Kernel_2 (CDK2)
Parallel Kernel 2
Find BMU Function
Dist_arr_1
BMU 1
Dist_arr_2
BMU 2
Q1: Find_BMU_Kernel_1 (FBK1)
Q2: Find_BMU_Kernel_2 (FBK2)
Select Final BMU from two Find BMU Kernels
Final BMU
Update Weight Function
Final BMU
Q1: Update Weight Kernel (UWK)
SOM_map_array_1, SOM_map_array_2
Modify learning factor and neighbourhood radius
Final SOM_map_array
Generate SOM map

Fig. 1: Enhanced parallel SOM architecture

```
Begin
For k=1 to Wjm do in parallel
MD(x, Wj) = (x1-Wj1) + (x2-Wj2) + … + (xn-Wjm)                (1)
dist_array[n] = MD(x, Wj )
End for
End
```

Fig. 2: Distance calculation algorithm

```
Begin
Determine neighborhood value
For $k=1$ to $B_n$ do in parallel
$W_{update} = W_{current}$ + neighborhood_function[$x(i)$- $W_{current}$]      (2)
End for
End
```

Fig. 3: Update weight algorithm

In addition, Figure 4 shows flow of the enhanced parallel SOM for better understanding. From the figure, shapes with white and blue colour are under host control. The processes that related to duo stimuli methods are coloured with blue. Meanwhile, the yellow shapes are controlled by device side. After execution of FBK1 and FBK2, GPU synchronizations are taking a place in order to make sure the executions of the both kernels are completed. After executing all the kernels, the next step is modifying learning factor and neighbourhood radius as shown in Figure 1. All the steps in the loop block will be repeated until finishing *n* iteration. Finally, the result of SOM will be generated by using map.

Explicitly, this study combines network partitioning by using online SOM training and data partitioning by using batch SOM training. The batch learning process is implemented into the proposed work by adapting duo stimuli that targeted to reduce half of the training cycle. The proposed work is essentially implementing batch learning processing to execute Parallel Kernel 1 and Parallel Kernel 2. For an example, the execution of CDK1 and CDK2 is considered as executing two different tasks. Furthermore, the detail algorithm for this proposed work has been explained in our previous paper [39]. The algorithm is expected to reduce the computation time and maintain the final results as good as the original online SOM training.

3. Research method

This study conducts an experiment to evaluate algorithm of the proposed enhanced parallel SOM architecture known as e-FGPSOM. The experiment is done to evaluate the efficiency and scalability of e-FGPSOM over larger dataset size and larger map size compared to algorithm for standard parallel SOM on HUMA platform known as FGPSOM. e-FGPSOM applies combination of online and batch SOM while FGPSOM applies online SOM for training type. Besides, e-FGPSOM uses five kernels while FGPSOM uses three kernels. However, both FGPSOM and e-FGPSOM are executed on OpenCL 2.0. The experimental design for this study is summarized in Table 1.

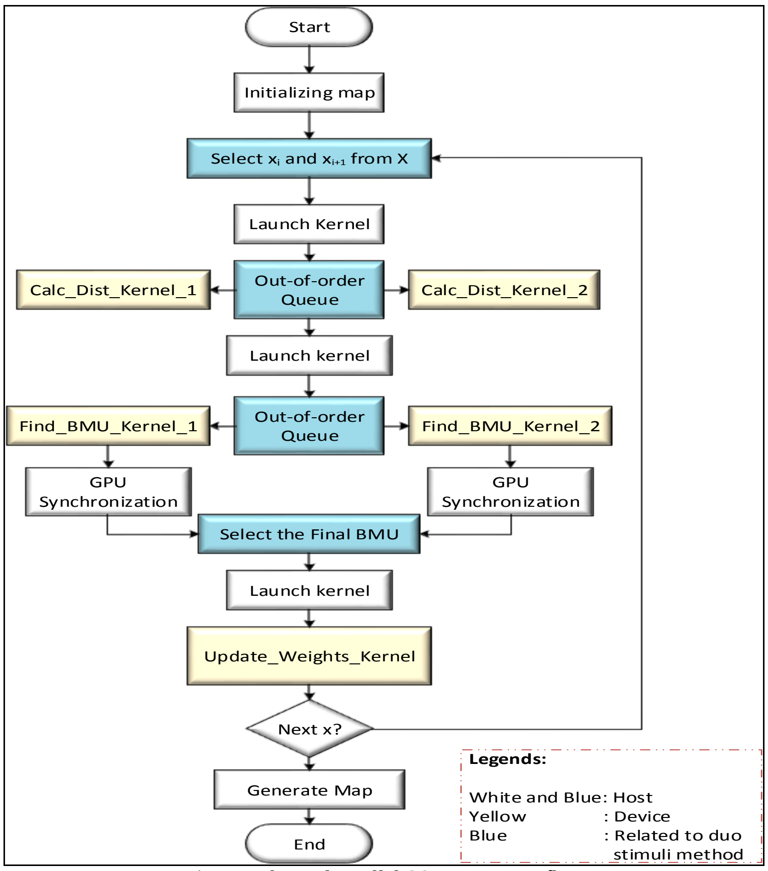

Fig. 4: Enhanced parallel SOM execution flow

Table 1: Experimental Design

Objectives	To evaluate efficiency and scalability
SOM Training Type	e-FGPSOM (Online + Batch SOM) FGPSOM (Online SOM)
No. of Kernel	e-FGPSOM (5 kernels) FGPSOM (3 kernels)
Dataset Sizes	10000, 20000, 30000, 40000
Map Sizes	50x50, 80x80, 100x100
No. of Parameters	3
Iterations (Epoch)	30
Quality Measurements	Time (s), Speed Up

The experiment uses Bank Marketing dataset because this dataset is considered as a huge and multivariate dataset [40]. It consists of 21 numbers of attributes and there are no missing values. The dataset is tested using three different map sizes in order to find the best map sizes which are 50x50, 80x80, and 100x100. This experiment also uses four different dataset sizes; 10000, 20000, 30000, 40000 with the same number of parameters and iterations. Moreover, the performance measurement is based on time in seconds and speed up. Four kinds of time have been calculated from the execution of each kernel and the total time.
Besides, this study also considers suitable hardware and software. Thus, the experiment is executed on a laptop that equipped with HSA enabled processor, Intel i7-6700HQ processor, 16 Gigabyte (GB) of Random-Access Memory (RAM) and built in Intel® HD Graphics 530. This processor belongs to the Skylake family which supports the OpenCL 2.0 and lower version. It is equipped with four CPU cores and 24 numbers of Execution Units (EUs) placed at GPUs. The reason of employing this processor is to implement SOM algorithm on HUMA platform. 16 GB of memory or RAM is applied due to increase processing of the processor for the reason that the memory is shared between CPU and GPU. The one Terabyte (TB) of storage is sufficient for this study.
Moreover, the software section consists of Microsoft Windows 10 for Operating System (OS), Microsoft Visual Studio 2013 for Integrated Development Environment (IDE), and GPU programming framework. C++ programming language has been used to write the coding for parallel SOM algorithm. Moreover, the most important software in this study is Intel OpenCL Software Development Kit (SDK). It is installed to accelerate the programming in a GPU environment. By using Intel® SDK for OpenCL™ Applications 2015 that supports the features in OpenCL 2.0 and lower versions, the SOM processing can be implemented in parallel. OpenCL 2.0 is used to implement enhanced parallel SOM on HUMA platform.

4. Result and analysis

The analysis of results for performance comparison between FGPSOM and e-FGPSOM are illustrated in Figure 5 and 6 respectively. From the graph, it can be seen that e-FGPSOM is able to reduce the computation time compared to FGPSOM for every dataset sizes and map sizes. Both solutions have similar trend for kernel processing where calculate distance and update weight kernels consume least computation time compared to the find BMU kernel. The computation time of update weight kernel of e-FGPSOM increases compared to FGPSOM due to the enhanced version includes array copying process in order to update both SOM map array for preparing the next iteration of processing. Meanwhile, the computation time of find BMU kernels is seen decreasing although it has included two synchronization points.

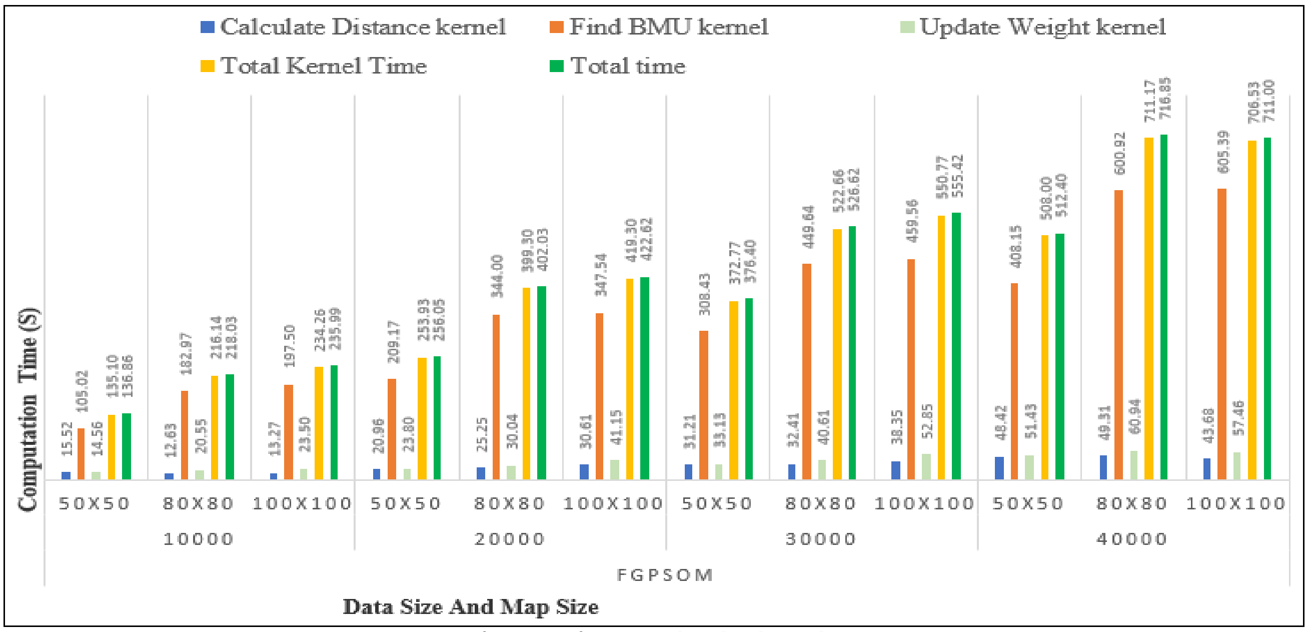

Fig. 5: Performance of FGPSOM based on larger dataset sizes

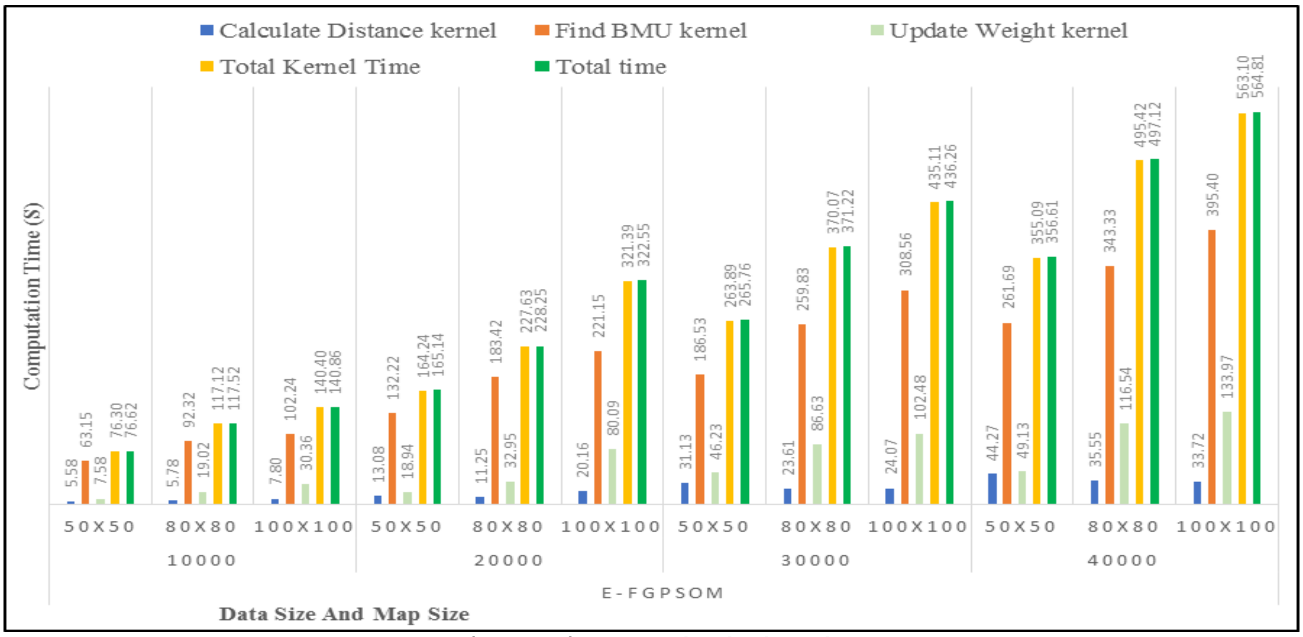

Fig. 6: Performance of e-FGPSOM based on larger dataset sizes

Overall, the performance of e-FGPSOM outperforms FGPSOM. However, the averages of speed up enhanced are decreasing over larger dataset sizes and map sizes. For example, the values of speed up enhanced of e-FGPSOM are between 1.675 to 1.855 and 1.310 to 1.761 for executing 10000 and 20000 datasets respectively. In addition, the values of speed up enhanced gradually decreased for processing 30000 and 40000 datasets with scores between 1.273 to 1.418 and 1.258 to 1.442 respectively. In details, both speed up enhanced and speed up overall performances can be realized as shown in Figure 7. From the figure, it can be seen that both of the speed up measurements decrease against larger map size. This trend could occur because of the hardware used reaches its limitation of processing. The hardware used is containing up to 5376 concurrent kernel instances. In details, the processing of 80x80 and 100x100 map sizes are comprised of 6400 neurons and 10000 neurons which contains more than capacity of the hardware used.

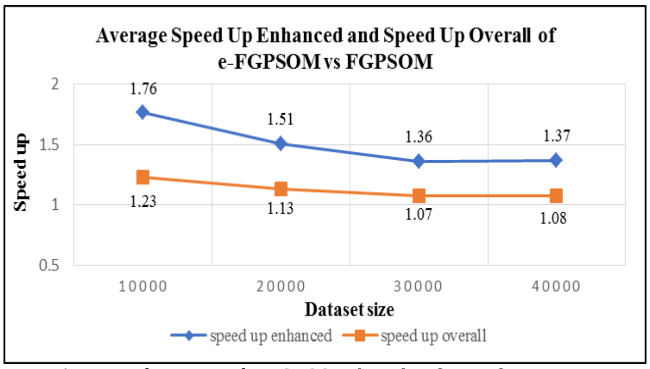

Fig. 7: Performance of e-FGPSOM based on larger dataset sizes

According to these results, this study agreed with [29] who have reported that the speed up ratios became lower when the parallel processing capacity of GPU had reached to its limit. It can be assumed that if the hardware processing capacity sufficient to cater the map size, the speed up enhanced can be maintain over average 1.5 or even more. Moreover, greater utilization of GPU cores by using the proposed works has increased the processing speed of parallel SOM.

5. Conclusion

This paper has presented parallel SOM using HUMA platform. The proposed solution is adapted with multiple stimuli approach. As a result, the number of kernels in GPU increased to five kernels rather than three kernels. This improvement is to enhance the utilization cores in the GPU. The overall results demonstrate that the proposed work is able to enhance parallel SOM in terms of efficiency and scalability performance. The advantages of the proposed work compared to the standard version of parallel SOM are:
i) More efficient due to combination of batch and online SOM training that executed on HUMA platform.
ii) More scalable in terms of cores utilization in GPU due to its imposition with multiple stimuli approach.
In conclusion, the proposed work is able to resolve the problems of communication latency and underutilized cores in GPU. Accordingly, the proposed work is able to offer a greater solution for small to medium sized of data analysis software. Although the proposed work has imposed with two modifications which are implementing multiple stimuli and adapting into GPU computing with unified memory model, the proposed work still far to achieve high score of speed up overall measurement. There is a limitation of the proposed work where the synchronization points at find BMU step could burden the processing of the find BMU kernel. Therefore, to solve the current limitation, identification of the final BMU values should be done at kernel processing. In the future, we would also implement other features of OpenCL 2.0 such as dynamic parallelism and work-group function in order to improve the limitation of the proposed work.

Acknowledgement

This study was funded under Academic Staff Bumiputera Training Scheme (SLAB) and FRGS grant with grant number FRGS/1/2015/ICT02/UITM/02/6 by Ministry of Education Malaysia. The authors also would like to thank the Universiti Teknologi MARA for supporting this study.

References

[1] W.-P. Tsai, S.-P. Huang, S.-T. Cheng, K.-T. Shao, and F.-J. Chang, “A data-mining framework for exploring the multi-relation between fish species and water quality through self-organizing map,” *Sci. Total Environ.*, vol. 579, pp. 474–483, 2016.
[2] J. Llanos *et al.*, “Load estimation for microgrid planning based on a self-organizing map methodology,” *Appl. Soft Comput.*, vol. 53, pp. 323–335, 2017.
[3] F. Matic *et al.*, “Oscillating Adriatic temperature and salinity regimes mapped using the Self-Organizing Maps method,” *Cont. Shelf Res.*, vol. 132, no. September 2016, pp. 11–18, 2017.
[4] T. Kohonen, “Essentials of the self-organizing map.,” *Neural Netw.*, vol. 37, pp. 52–65, Jan. 2013.
[5] S. McConnell, R. Sturgeon, G. Henry, A. Mayne, and R. Hurley, “Scalability of Self-organizing Maps on a GPU cluster using OpenCL and CUDA,” *J. Phys. Conf. Ser.*, vol. 341, p. 012018, 2012.
[6] S. Hasan, S. M. Shamsuddin, and N. Lopes, “Machine Learning Big Data Framework and Analytics for Big Data Problems,” *Int. J. Adv. Soft Comput. Appl.*, vol. 6, no. 2, pp. 1–17, 2014.
[7] K. Ben Khalifa, B. Girau, F. Alexandre, and M. H. Bedoui, “Parallel FPGA implementation of self-organizing maps,” in *Proceedings. The 16th International Conference on Microelectronics, 2004. ICM 2004.*, 2004, pp. 709–712.
[8] W. Kurdthongmee, “A novel Kohonen SOM-based image compression architecture suitable for moderate density {FPGAs},” *Image Vis. Comput.*, vol. 26, no. 8, pp. 1094–1105, 2008.
[9] W. Kurdthongmee, “A low latency minimum distance searching unit of the {SOM} based hardware quantizer,” *Microprocess. Microsyst.*, vol. 39, no. 2, pp. 135–143, 2015.
[10] T. Abe, Y. Hamano, S. Kanaya, K. Wada, and T. Ikemura, “A Large-Scale Genomics Studies Conducted with Batch-Learning SOM Utilizing High-Performance Supercomputers,” in *Bio-Inspired Systems: Computational and Ambient Intelligence: 10th International Work-Conference on Artificial Neural Networks, IWANN 2009, Salamanca, Spain, June 10-12, 2009. Proceedings, Part I*, J. Cabestany, F. Sandoval, A. Prieto, and J. M. Corchado, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 829–836.
[11] S. J. Sul and A. Tovchigrechko, “Parallelizing BLAST and SOM Algorithms with MapReduce-MPI Library,” in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, 2011, pp. 481–489.
[12] F. C. Moraes, S. C. Botelho, N. D. Filho, and J. F. O. Gaya, “Parallel High Dimensional Self Organizing Maps Using CUDA,” *2012 Brazilian Robot. Symp. Lat. Am. Robot. Symp.*, pp. 302–306, Oct. 2012.
[13] M. Mojarab, H. Memarian, M. Zare, A. Hossein Morshedy, and M. Hossein Pishahang, “Modeling of the seismotectonic provinces of Iran using the self-organizing map algorithm,” *Comput. Geosci.*, vol. 67, pp. 150–162, Jun. 2014.

[14] T. Richardson and E. Winer, "Extending parallelization of the self-organizing map by combining data and network partitioned methods," *Adv. Eng. Softw.*, vol. 88, pp. 1–7, 2015.

[15] T. Kohonen, "The self-organizing map," *Neurocomputing*, vol. 21, no. 1–3, pp. 1–6, 1998.

[16] M. Takatsuka and M. Bui, "Parallel batch training of the self-organizing map using OpenCL," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6444 LNCS, pp. 470–476, 2010.

[17] L. Zhu, W. Guo, and Y. Bai, "Self-Organizing Maps Neural Networks on Parallel Cluster," in *2009 First International Conference on Information Science and Engineering*, 2009, pp. 384–388.

[18] J. Fort, P. Letremy, M. Cottrell, I. Elie, and U. Nancy, "Advantages and drawbacks of the Batch Kohonen algorithm," *Proc Eur. Symp. Artif. Neural Networks ESANN*, pp. 223–230, 2002.

[19] G. Dzemyda and O. Kurasova, "Parallelization of the SOM-Based Integrated Mapping," in *Artificial Intelligence and Soft Computing - ICAISC 2004: 7th International Conference, Zakopane, Poland, June 7-11, 2004. Proceedings*, L. Rutkowski, J. H. Siekmann, R. Tadeusiewicz, and L. A. Zadeh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 178–183.

[20] C. Garcia, M. Prieto, and A. Pascual-Montano, "A Speculative Parallel Algorithm for Self-Organizing Maps," *Proc. Parallel Comput. 2005 (ParCo 2005)*, vol. 33, pp. 615–622, 2005.

[21] D. MacLean and I. Valova, "Parallel Growing SOM Monitored by Genetic Algorithm," in *2007 International Joint Conference on Neural Networks*, 2007, pp. 1697–1702.

[22] P. Gajdos and J. Platos, "GPU Based Parallelism for Self-Organizing Map," in *Advances in Intelligent Systems and Computing*, 2013, vol. 179, no. Ihci 2011, pp. 3–12.

[23] U. Seiffert and B. Michaelis, "Multi-Dimensional Self-Organizing Maps on Massively Parallel Hardware," in *Advances in Self-Organising Maps*, London: Springer London, 2001, pp. 160–166.

[24] V. Demian and J. C. Mignot, "Optimization of the self-organizing feature map on parallel computers," in *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, 1993, vol. 1, pp. 483–486 vol.1.

[25] M. Yasunaga, K. Tominaga, and J. H. K. J. H. Kim, "Parallel self-organization map using multiple stimuli," *IJCNN'99. Int. Jt. Conf. Neural Networks. Proc. (Cat. No.99CH36339)*, vol. 2, pp. 1127–1130, 1999.

[26] S. Q. Khan and M. A. Ismail, "Design and implementation of parallel SOM model on GPGPU," *2013 5th Int. Conf. Comput. Sci. Inf. Technol.*, pp. 233–237, Mar. 2013.

[27] A. Faro, D. Giordano, and S. Palazzo, "Integrating unsupervised and supervised clustering methods on a GPU platform for fast image segmentation," *2012 3rd Int. Conf. Image Process. Theory, Tools Appl. IPTA 2012*, pp. 85–90, 2012.

[28] J. Lachmair, E. Merényi, M. Porrmann, and U. Rückert, "A reconfigurable neuroprocessor for self-organizing feature maps," *Neurocomputing*, vol. 112, pp. 189–199, Jul. 2013.

[29] A. De, Y. Zhang, and C. Guo, "A parallel image segmentation method based on SOM and GPU with application to MRI image processing," *Neurocomputing*, vol. 198, pp. 180–189, 2016.

[30] K. Perelygin, S. Lam, and X. Wu, "Graphics Processing Units and Open Computing Language for parallel computing," *Comput. Electr. Eng.*, vol. 40, no. 1, pp. 241–251, Jan. 2014.

[31] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors*. Elsevier, 2013.

[32] S. Mittal and J. S. Vetter, "A Survey of CPU-GPU Heterogeneous Computing Techniques," *ACM Comput. Surv.*, vol. 47, no. 4, p. 69:1--69:35, Jul. 2015.

[33] S. Mukherjee, Y. Sun, P. Blinzer, A. K. Ziabari, and D. Kaeli, "A Comprehensive Performance Analysis of HSA and OpenCL 2.0," *2016 IEEE Int. Symp. Perform. Anal. Syst. Softw.*, no. April, 2016.

[34] G. Kyriazis, "Heterogeneous System Architecture : A Technical Review," pp. 1–18, 2012.

[35] AMD Inc., "Compute Cores," pp. 1–6, 2014.

[36] S. Junkins, "The Compute Architecture of Intel ® Processor Graphics Gen9," 2015.

[37] K. OpenCL, "OpenCL Specification," 2014.

[38] M. Lichman, "UCI Machine Learning Repository," *Irvine, CA: University of California, School of Information and Computer Science*, 2013. [Online]. Available: http://archive.ics.uci.edu/ml.

[39] M. F. Mustapha, N. E. Abd Khalid, A. Ismail, and M. Manaf, "Enhancing Parallel Self-organizing Map on Heterogeneous System Architecture," in *Soft Computing in Data Science*, 2017, pp. 162–174.

[40] S. Moro, P. Cortez, and P. Rita, "A data-driven approach to predict the success of bank telemarketing," *Decis. Support Syst.*, vol. 62, pp. 22–31, 2014.