Enhancing Parallel Self-organizing Map on Heterogeneous System Architecture

Muhammad Firdaus Mustapha(✉), Noor Elaiza Abd Khalid(✉), Azlan Ismail, and Mazani Manaf

Universiti Teknologi MARA, 40450 Shah Alam, Selangor, Malaysia
firdausl9@gmail.com, azlanismail08@gmail.com, {elaiza,mazani}@tmsk.uitm.edu.my

Abstract. Self-organizing Map (SOM) is a very popular algorithm that has been used as clustering algorithm and data exploration. SOM consists of complex calculations where the calculation of complexity depending on the circumstances. Many researchers have managed to improve online SOM processing speed using discrete Graphic Processing Units (GPU). In spite of excellent performance using GPU, there is a situation that causes computer hardware underutilized when executing online SOM variant on GPU architecture. In details, the situation occurs when number of cores is larger than the number of neurons on map. Moreover, the complexities of SOM steps also increase the usage of high memory capacity which leads to high rate memory transfer. Recently, Heterogeneous System Architecture (HSA), that integrated Central Processing Unit (CPU) and GPU together on a single chip are rapidly attractive the design paradigm for recent platform because of their remarkable parallel processing abilities. Therefore, the main goal of this study is to reduce computation time of SOM training through adapting HSA platform and combining two SOM training processes. This study attempts to enhance the processing of SOM algorithm using multiple stimuli approach. The data used in this study are benchmark datasets from UCI Machine Learning repository. As a result, the enhanced parallel SOM algorithm that executed on HSA platform is able to score a promising speed up for different parameter size compared to standard parallel SOM on HSA platform.

Keywords: Parallel self-organizing map · GPU computing
Heterogeneous system architecture

1 Introduction

Self-organizing Map (SOM) is an unsupervised neural network that has been used as data analysis method. It is being widely used and applied to solve clustering and data exploration problems in various domain areas [1]. There were many researches have been found in the literature that used SOM to solve clustering problem [2, 3]. Despite its excellent performance, there are problems related to slow processing when visualizing large map size [4]. This imposed heavy workload on the processor especially

A. Mohamed et al. (Eds.): SCDS 2017, CCIS 788, pp. 162–174, 2017.
https://doi.org/10.1007/978-981-10-7242-0_14

when dealing with winner-search and updating weightage of neurons on the map [1]. On the other hand, the datasets dimension also have high influence in SOM processing [5].

This situation attracts much interest among researchers to improve SOM processing by parallelizing the algorithm. Among the common ways to parallelize SOM are network or map partitioning [6, 7] and data or example partitioning [8, 9]. However, there also efforts to parallelize SOM algorithm through combining both network and data partitioning [10, 11] with the interest to gain advantages of both parallelism. In the meantime, most of research works on improving SOM are aimed to achieve efficiency and scalability in their proposed works. In details, proposed parallel SOM that efficient should be faster in term of processing than the previous version [12, 13]. Meanwhile, some research works are attempting to increase the utilization of processing elements in executing the SOM algorithm [7, 14]. Furthermore, several research works pursue to lower the power consumption [15], and solve computer cluster problems [16, 17].

On the other hand, two computer architectures mostly used by researchers in improving SOM algorithm are; Single Instruction Stream Multiple Data Stream (SIMD) and Multiple Instruction Streams Multiple Data Streams (MIMD). Some of the SIMD architectures used Graphic Processing Unit (GPU) computing [18, 19], Field Programmable Gate Array (FPGA) [6, 20], and specialized hardware architecture for Artificial Neural Network (ANN) [21, 22]. Meanwhile, MIMD architectures are employed by researchers to parallelize SOM consists of different types of computer clusters. Among several computer architectures, GPU computing offers an efficient solution at lower cost compared to others.

GPU or widely known as General Purpose Graphic Processing unit (GPGPU) is a many core processor consisting hundreds or even thousands of compute cores [23]. It has been used to speed up applications of scientific computing and simulations. GPU computing has been proven to have high throughput in processing large data floating point operations in graphic applications [24]. Since the introduction of GPU programming frameworks such as of Compute Unified Device Architecture (CUDA) in 2007 and Open Computing Language (OpenCL) in 2009 [24], the GPUs have become popular in designing parallel algorithm in the quest for higher speed. In the meantime, the evolution of hardware technology, has made it possible to design high performance scientific computing software [25]. Essentially, GPU is an accelerator to Central Processing Unit (CPU) that has been mainly used for graphic purposes before applying to process scientific data. Combination of CPU and GPU that work closely together creates a paradigm known as Heterogeneous Computing (HC) [26].

On top of that, many researchers have attempted to capitalize HC to execute SOM algorithm in parallel manner. However Hasan et al. [5] found that when parallelizing SOM with larger map size and high attribute dimension, it will significantly slow down the processing even with both CPU and GPU. Many researchers agreed that executing SOM on GPU shows significantly increase processing speed for large data compared to executing on CPU only [11, 20, 27]. Moreover, due to the restrictions imposed by past GPU architectures, most of these frameworks treated the GPU as an accelerator which can only work under close control of the CPU. Consequently, the communication protocol between CPU and GPU is a source of high latency which causes bottleneck. This drawback is enhanced when using distributed memory of HC where the memory management has to be manually configured by the programmer to manage data movement between CPU and GPU which includes data transfer between host code and device code [28].

A more recent technology hardware design of heterogeneous systems is a single integrated unify CPUs and GPUs circuit chip which is known as Heterogeneous System Architecture (HSA) [28]. This technology provides a unified programming platform which eliminates programmability barrier and reduces CPU and GPU communication latency. The introduction of OpenCL 2.0 that supports HSA specification in July 2013 is able to improve communication by allowing the GPU to manage their own resources as well as access some of the CPU's resources. It also introduces Shared Virtual Memory (SVM) which allows the host and the device to share a common virtual address range [29]. This reduces overhead by eliminating deep copies during host-to-device and device-to-host data transfers. Deep copies involve complete duplicating objects in the memory thus reduce redundancies [28].

In view of the above discussion, this study proposed an enhanced parallel SOM which executes on HSA compliant processor to solve clustering problem. The data used are benchmark datasets that were acquired from UCI Machine Learning Repository [30]. The performance of the algorithm is evaluated in terms of efficiency, scalability and accuracy. Efficiency and scalability are based on processing time while accuracy is based on number of class generated.

This paper is organized as follows. Section 2 explains about the proposed work on parallel SOM using HSA. Section 3 provides explanation on experimental setup while Sect. 4 discusses the experimental result and discussion. Lastly, Sect. 5 provides conclusion and future research directions.

2 Proposed Work

The proposed work consists of enhanced parallel SOM architecture and enhanced parallel SOM algorithm which will be explained in Sects. 2.1 and 2.2 respectively.

2.1 Enhanced Parallel SOM Architecture

Previous studies that highlighted parallel SOM have been successfully executed on GPU. Almost all the researchers in the literature apply parallelism at calculate distance and find Best Matching Unit (BMU) steps. There are many of them apply parallelism at update weight step. Consequent of that, this study proposed to parallelize these three steps into the new enhanced parallel SOM architecture. Meanwhile, heterogeneous system compromises a promising solution for reducing latency in communication between CPU and GPU. In order to gain these advantages, the proposed architecture is utilizing OpenCL 2.0 platform which specifically SVM feature. The implementation of this work is based on fined-grained SVM buffers. The fined-grained SVM buffers are synchronized during the implementation of SVM buffer which could reduce communication latency between CPU and GPU. The design of the proposed architecture is extended from the previous work [31] where it is introduced with two parallel kernels for distance calculation and find BMU as depicted in Fig. 1. The idea of parallel

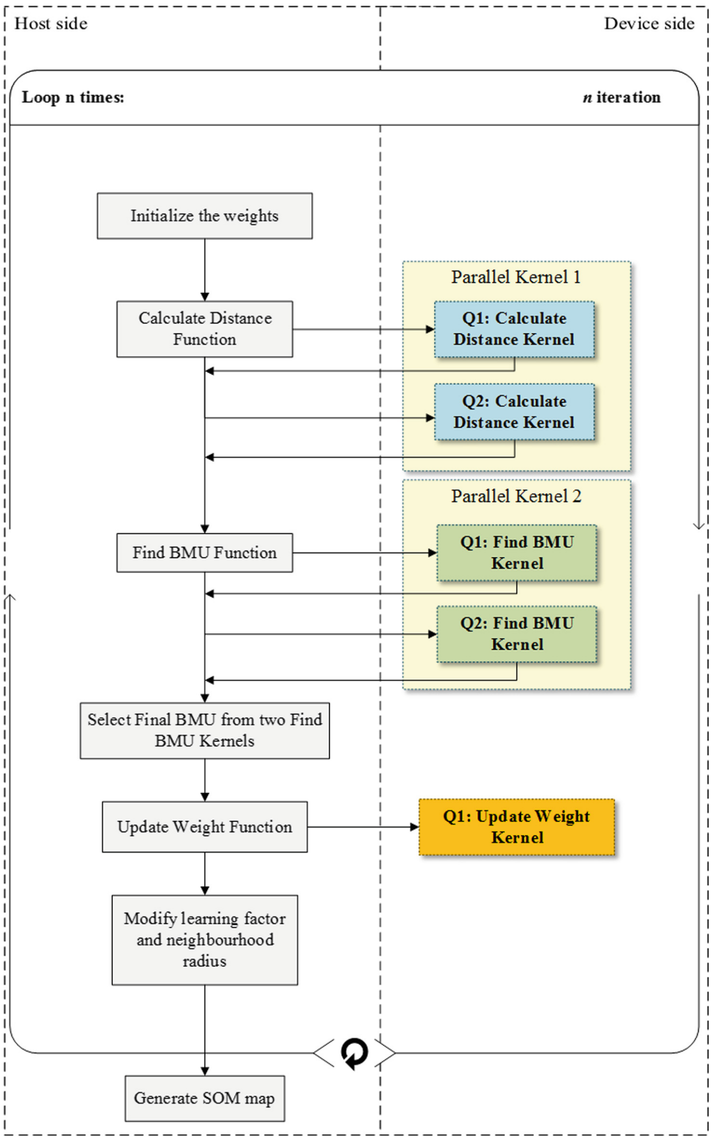

Fig. 1. Enhanced parallel SOM architecture

kernels is based on the analogy of biological neural networks in which neurons respond to the stimuli in parallel [32]. However this work is differ from [32] in term of SOM training types and hardware architecture. The main reason of duplicating the kernels is to increase utilization of work units in GPU.

The design of parallel kernels is realized by implementing multiple stimuli into the architecture. The main reason of parallelized the kernels are to increase utilization of work units in GPU. This work is supported by OpenCL where OpenCL allows a programmer to create more than one queue for execution. The queue will process based on out-of-order execution [29]. In out-of-order execution mode there is no guarantee that the enqueued commands will finish in the order they were queued. The first parallel kernel is Parallel Kernel 1 that consists of Calc_Dist_Kernel_1 and Calc_Dist_Kernel_2. Meanwhile, Parallel Kernel 2 consists of Find_BMU_Kernel_1 and Find_BMU_Kernel_2. The execution of kernels in each parallel kernel might overlap each other. This situation will create a chance to increase the utilization of work units in GPU side.

This study attempts to implement batch learning process through duo stimuli which leads to reduce training cycle to half. This solution that combines both online training and batch training will gain the benefits of the two types of training. The enhanced parallel SOM architecture is predicted to reduce the computation time due to the reduction of the training cycle to half and maintain the final results as online training. The enhanced parallel SOM architecture is essentially implement batch learning processing for executing two calculate distance kernels and two find BMU kernels. For instance, the execution of Calc_Dist_Kernel_1 is considered as executing one task and the execution of Calc_Dist_Kernel_2 also considered as another task. The rest of the algorithm of the proposed solution is similar to online training SOM algorithm.

2.2 Enhanced Parallel SOM Algorithm

This section will describe the parallel algorithm that applies the enhanced parallel SOM architecture. The enhanced parallel SOM algorithm includes additional three steps compared to the original SOM algorithm. For better understanding to the reader, the enhanced parallel SOM algorithm in this study is labeled with e-FGPSOM. Figure 2 illustrates pseudocode of e-FGPSOM.

In depth of the proposed works, the algorithm begins with initializing SOM parameters such as learning factor and weights at the host side. The input data is retrieved and stored into an array. The training process begins with selecting duo stimuli randomly. Each input will be assigned to a command queue. In order to realize duo stimuli method, e-FGPSOM requires two sets of parameter of two calculate distance kernels and two update weight kernels due to the algorithm employs separate command queue for each kernel execution. Each kernel requires a map array. Both map arrays should contain similar values of neurons' weights before the kernels processing is started. Right after the map has initialized using SOM_map_array_1, the values of SOM_map_array_1 will be copied into SOM_map_array_2. The SOM map array is very important because it will be employed within all kernel processing. At the step 2, the algorithm obtains duo stimuli from the dataset randomly and then at the step 3, the host broadcast the two inputs to device side, specifically the host assigns to two calculate distance kernels; Calc_Dist_Kernel_1 and Calc_Dist_Kernel_2, and two find BMU kernels; Find_BMU_Kernel_1 and Find_BMU_Kernel_2. Once all information has been broadcasted, the kernels are ready for the execution especially for calculate distance kernel at step 4. Each kernel at the GPU side is invoked by function respectively. The functions also provide setting, initializing parameters, and call the kernels. For example, the calculate distance function is used to call Calculate Distance kernel and it is done the same way with the other two kernels.

```
<BEGIN>
1.Randomly initialize the neuron′ weights $W\left[w_{j1}, w_{j2}, \ldots w_{jm}\right]$
2.The HOST side ramdomly selects duo stimuli, $x_i$ and $x_{i+1}$
from $X\left[x_1, x_2, \ldots x_m\right]$ according to sequence.
3.The HOST side broadcasts the two input at once
$X\left[(x_i, x_{i+1}), \ldots, (x_{m-1}, x_m)\right]$ to DEVICE side.
4.Execute calculate distance kernels, Calc_Dist_Kernel_1
and Calc_Dist_Kernel_2 based on out-of-order execution
mode.
5.Execute find BMU kernels, Find_BMU_Kernel_1 and
Find_BMU_Kernel_2 based on out-of-order execution mode.
6.The HOST side select the winner of the winners from
both find BMU kernels.
7.The HOST side braoadcasts the final BMU to DEVICE
side.
8.Execute update weights kernel.
9.Repeat step 2 to 8 until certain termination
condition.
10.Generate the map using visualization algorithm.
<END>
```

Fig. 2. Enhanced parallel SOM algorithm

Essentially, the original flow of SOM algorithm is maintained unless with the additional two parallel stimuli execution that has applied for execution of two calculate distance kernels and two find BMU kernels in step 4 and step 5 respectively. The executions of two calculate distance kernels and two find BMU kernels are based on out-of-order execution mode. There is no guarantee that the enqueued commands will finish execution in the order because the execution of kernel is based on when the clEnqueueNDRangeKernel calls are made within a command-queue.

Calculate Distance Kernel. The calculate distance kernel is used to calculate the distance between neurons and current input vector. The amount of work units is employed to parallelize the calculation distance step is mapped by amount of work-items on GPU where the amount of work-items is equal to the number of neurons in the SOM map. Specifically, each work-item of the kernel is responsible for finding the distance between a single neuron and the current input vector. This study applies Manhattan distance calculation.

Find BMU Kernel. The Find BMU kernel applies two stages reduction method. The kernel utilizes work items the same amount of neurons on SOM map. The first stage of reduction method is to find the minimum distance for each local work group. The values of minimum distances of each work group will be stored into local array. The second stage is to find the minimum distance for each Compute Unit (CU). The minimum values of each CU then stored into global array and the host will determine the winning neurons. After the execution of both Find BMU 1 and Find BMU 2, at the step 6, the winner among the winners or final BMU from the both kernels will be selected with the minimum value at the host side. With the selected of the final BMU means the input vector which has the final BMU will use to update the map meanwhile the loser input will be eliminated. The final BMU will be broadcasted to device side for execution of third kernel, the update weight kernel.

Update Weight Kernel. This kernel is the third kernel in the proposed architecture that updates the weight of neurons based on learning rate and neighborhood function. The learning rate defines how much a neuron's vector is altered through an update with referring to how far the distance of the neuron from the BMU on the map. The BMU and its close neighbors will be altered the most, while the neurons on the outer edges of the neighborhood are changed the least. The enhanced version includes array copying process at host side right after the update weight kernel completed the execution. The array copying process is to copy the updated map array into another map array which is not selected as final BMU. The array copying codes are simply assign one array to another array with the same size through looping. Immediately after executing the three kernels, the learning factor and neighborhood radius are updated with the new values. All of the steps included in the loop block will repeat until certain number iterations or epochs before the SOM map is generated.

3 Experimental Setup

Two experiments have been conducted in this study which is result evaluation and result validation. These experiments are performed with the interest to evaluate the proposed work. Initially, this study employs four benchmark datasets from UCI Machine Learning Repository [30] for results validations. The details of the datasets are shown in Table 1. These datasets have been selected because information on the number of classes is known in advance and ease for the validation process. The information of number of classes is as provided by Fränti [33]. These datasets are processed by e-FGPSOM where each experiment is performed to validate each dataset.

Table 1. Benchmark datasets for result validations

Benchmark dataset	Number of training data	Number of attributes of pattern	Number of classes
Iris	150	4	3
Glass	214	9	6
Wine	178	13	3
Yeast	1484	8	10

On the other hand, for evaluations purpose, this study employs Bank Marketing benchmark dataset from UCI Machine Learning Repository [34]. Initially, data pre-processing takes place before the experiment is conducted. Data pre-processing is one of the important steps in data mining to obtain final datasets that can be considered as correct and useful for further data mining algorithm [35]. There are four methods in data pre-processing which are data cleaning, data integration, data transformation and data reduction [36]. Thus, this study uses method of data reduction by using discretization technique to convert values of the selected attributes [37]. This study converts all the values that appear in categorical into numeric values for easy processing in SOM.

Firstly, the experiment starts with result validation. The objective of result validation is to make sure the proposed work is capable to generate correct results by comparing the results from e-FGPSOM with results from other researchers. Four dimension sizes have been used: 4, 9, 13 and 8 parameters according to specific benchmark datasets as shown in Table 2. The algorithm is tested on the same map size, 40 × 40 by using 250 iterations.

Table 2. Experimental design for enhanced parallel SOM evaluation

Experiment series	Datasets parameter		SOM parameter		Performance measurement
	No. of samples	No. of parameter	No. of iteration	Map size	
Result validation	Iris, Glass, Wine, Yeast	4, 9, 13, 8	250	40 × 40	Number of classes
Result evaluation	10000, 15000	3, 5, 8	30	50 × 50	Time, s and speed up

On the other note, result evaluation is concerning on evaluation of the proposed work on different dimension size or number of parameter of dataset. Three dimension sizes have been used: 3, 5 and 8 parameters. The algorithm is tested on the same map size, 50 × 50 by using 30 iterations. Result evaluation applies time comparison and speed up [38] for performance measurements.

Moreover, for analysis purpose, the result of this study is compared to the result of our previous work [31] referred as FGPSOM. FGPSOM is based on standard parallel SOM on HSA platform. Both FGPSOM and e-FGPSOM implemented on OpenCL 2.0. These experiments are conducted on a laptop that equipped with Intel Skylake i7-6700HQ processor and built in Intel® HD Graphics 530.

4 Experimental Result and Discussion

4.1 Result Validations

Firstly, the experimental result starts with result validation. As mentioned in Sect. 3, result validation for e-FGPSOM is based on four benchmark datasets which are Iris, Glass, Wine, and Yeast dataset. Figure 3 illustrates the generated results in map visualization by e-FGPSOM. This figure shows that Iris dataset and wine dataset clearly produce three classes. This results can be validated with Fränti [33] that the proposed work is capable to generate the similar results. Meanwhile, the results of glass

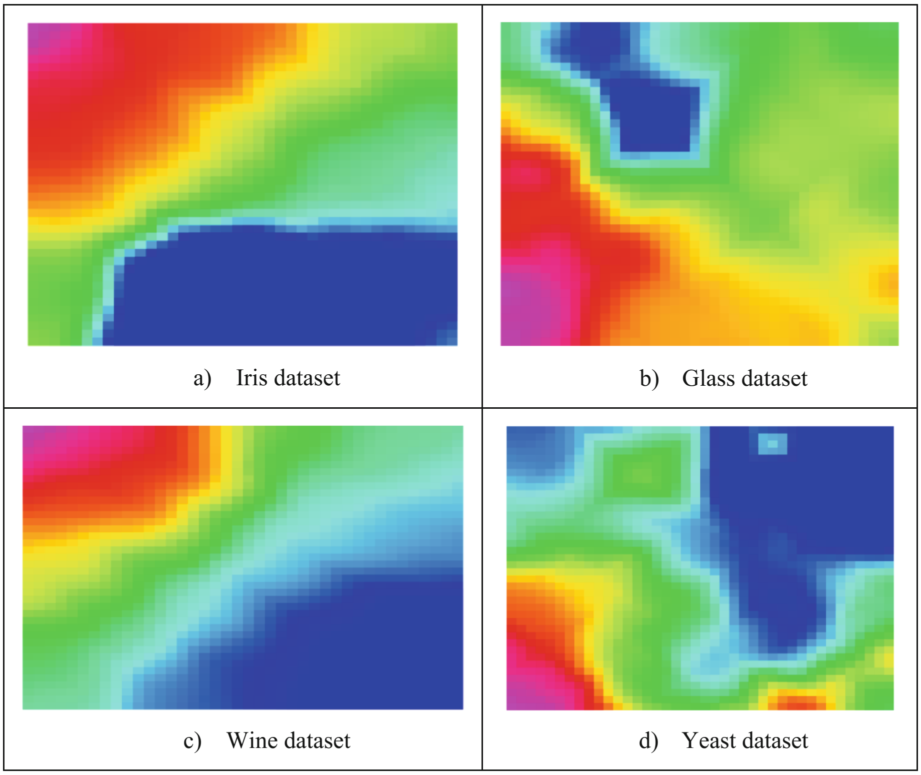

Fig. 3. Result produced by using algorithm of the enhanced parallel SOM architecture

and yeast datasets are quite subjective to be decided. However, one could be identified the number of classes via observation that the proposed work also capable to generate equivalent results [33].

4.2 Results Evaluation

For result evaluation purpose, the proposed algorithm tested on the same dataset size and map size. However, this study is focusing on experimenting the proposed algorithm on different dimension size of dataset; 3, 5, and 8 parameters. Figure 4 demonstrates the comparison result between FGPSOM and e-FGPSOM. From this figure, the results of e-FGPSOM perform better than results of FGPSOM for both datasets. The result of e-FGPSOM shows the improvement compared to FGPSOM where all the results of e-FGPSOM capable to reduce the processing time. The experiment exposes that e-FGPSOM successfully improved the processing speed compared to FGPSOM when experimenting on larger dimension size of dataset.

For the details, according to 10000 dataset, e-FGPSOM achieves 1.24x, 1.13x, and 1.10x of speed up for 3, 5, and 8 parameters respectively. Meanwhile, for 15000 dataset, e-FGPSOM scores 1.16x, 1.11x, and 1.10x of speed up for 3, 5, and 8 parameters

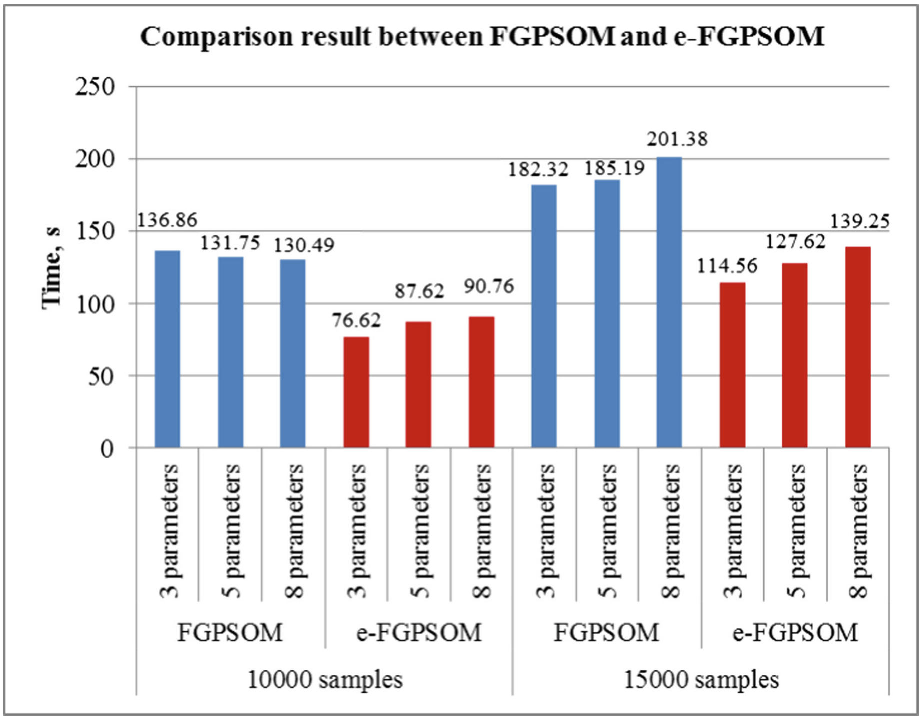

Fig. 4. Result produced by using algorithm of the enhanced parallel SOM architecture

respectively. The proposed algorithm is capable to score speed up over FGPSOM due to it applies the proposed architecture which consists of parallel kernels. The execution of parallel kernels has triggered multiple stimuli processing. The processing of multiple stimuli capable to increase the utilization of GPU cores compared to FGPSOM. However, due to complexity of processing larger dataset and larger parameters size, the speeds up scores are reducing over the increasing the dataset size and parameter size.

5 Conclusion

In this study, we proposed an enhanced parallel SOM that based on heterogeneous system architecture. The proposed architecture is extended from parallel SOM researches that consist of three kernels: calculate distance kernel, find BMU kernel, and update weight kernel. The proposed architecture is included with two calculate distance kernels and two find BMU kernels. The proposed architecture is designed with the aim to increase the utilization of processing element on GPU.

The overall results indicate that the enhanced parallel SOM or e-FGPSOM is able to improve in terms of efficiency and scalability performance. This is due to e-FGPSOM that optimizes the usage of cores in the GPU. The enhanced parallel SOM or e-FGPSOM also demonstrates some advantages from various perspectives as below:

- The proposed work is capable to generate comparable results (number of classes) compared to other researches by using benchmark datasets. Thus, the proposed work maintains the accuracy of the result.
- The proposed work more scalable in terms of GPU cores utilization due to its imposition with multiple stimuli method. The implementation of multiple stimuli method also improves the proposed work to more efficient.

The proposed work has a limitation where the synchronization point at find BMU step could burden the processing of the find BMU kernel. Based on the limitation, there seems to be an opportunity for improvement. In the future, the consumption time of synchronization point could be reduced through eliminating the access of BMUs values at the host side. The final BMU values should be identified at kernel processing.

Acknowledgement. This study was funded by Ministry of Higher Education (MOHE) of Malaysia, under the FRGS, grant no. FRGS/1/2015/ICT02/UITM/02/6 and Academic Staff Bumiputera Training Scheme (SLAB). The authors also would like to thank the Universiti Teknologi MARA for supporting this study.

References

1. Kohonen, T.: Essentials of the self-organizing map. Neural Netw. **37**, 52–65 (2013)
2. Llanos, J., Morales, R., Núñez, A., Sáez, D., Lacalle, M., Marín, L.G., Hernández, R., Lanas, F.: Load estimation for microgrid planning based on a self-organizing map methodology. Appl. Soft Comput. **53**, 323–335 (2017)
3. Matic, F., Kovac, Z., Vilibic, I., Mihanovic, H., Morovic, M., Grbec, B., Leder, N., Dzoic, T.: Oscillating adriatic temperature and salinity regimes mapped using the self-organizing maps method. Cont. Shelf Res. **132**, 11–18 (2017)
4. McConnell, S., Sturgeon, R., Henry, G., Mayne, A., Hurley, R.: Scalability of self-organizing maps on a GPU cluster using OpenCL and CUDA. J. Phys: Conf. Ser. **341**, 12018 (2012)
5. Hasan, S., Shamsuddin, S.M., Lopes, N.: Machine learning big data framework and analytics for big data problems. Int. J. Adv. Soft Comput. Appl. **6**, 1–17 (2014)
6. Kurdthongmee, W.: A novel Kohonen SOM-based image compression architecture suitable for moderate density {FPGAs}. Image Vis. Comput. **26**, 1094–1105 (2008)
7. Kurdthongmee, W.: A low latency minimum distance searching unit of the SOM based hardware quantizer. Microprocess. Microsyst. **39**, 135–143 (2015)
8. Moraes, F.C., Botelho, S.C., Filho, N.D., Gaya, J.F.O.: Parallel high dimensional self organizing maps using CUDA. In: 2012 Brazilian Robotics Symposium and Latin American Robotics Symposium, pp. 302–306 (2012)
9. Sul, S.J., Tovchigrechko, A.: Parallelizing BLAST and SOM Algorithms with MapReduce-MPI library. In: 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, pp. 481–489 (2011)
10. Mojarab, M., Memarian, H., Zare, M., Hossein Morshedy, A., Hossein Pishahang, M.: Modeling of the seismotectonic provinces of Iran using the self-organizing map algorithm. Comput. Geosci. **67**, 150–162 (2014)
11. Richardson, T., Winer, E.: Extending parallelization of the self-organizing map by combining data and network partitioned methods. Adv. Eng. Softw. **88**, 1–7 (2015)

12. Garcia, C., Prieto, M., Pascual-Montano, A.: A speculative parallel algorithm for self-organizing maps. In: Proceedings of Parallel Computing 2005 (ParCo 2005), vol. 33, pp. 615–622 (2005)
13. MacLean, D., Valova, I.: Parallel growing SOM monitored by genetic algorithm. In: 2007 International Joint Conference on Neural Networks, pp. 1697–1702 (2007)
14. Dlugosz, R., Kolasa, M., Pedrycz, W., Szulc, M.: Parallel programmable asynchronous neighborhood mechanism for kohonen SOM implemented in CMOS technology. IEEE Trans. Neural Netw. **22**, 2091–2104 (2011)
15. Khalifa, K.B., Girau, B., Alexandre, F., Bedoui, M.H.: Parallel FPGA implementation of self-organizing maps. In: Proceedings of the 16th International Conference on Microelectronics, ICM 2004, pp. 709–712 (2004)
16. Yang, M.-H., Ahuja, N.: A data partition method for parallel self-organizing map. In: International Joint Conference on Neural Networks, IJCNN 1999, vol. 3, pp. 1929–1933 (1999)
17. Schabauer, H., Schikuta, E., Weishäupl, T.: Solving very large traveling salesman problems by SOM parallelization on cluster architectures. In: Proceedings of Parallel and Distributed Computing, Applications and Technologies, PDCAT, pp. 954–958 (2005)
18. Gajdos, P., Platos, J.: GPU based parallelism for self-organizing map. In: Kudělka, M., Pokorný, J., Snášel, V., Abraham, A. (eds.) Intelligent Human Computer Interaction. Advances in Intelligent Systems and Computing, vol. 179, pp. 3–12. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-31603-6_20
19. Nguyen, V.T., Hagenbuchner, M., Tsoi, A.C.: High resolution self-organizing maps. In: Kang, B.H., Bai, Q. (eds.) AI 2016. LNCS, vol. 9992, pp. 441–454. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50127-7_38
20. Lachmair, J., Merényi, E., Porrmann, M., Rückert, U.: A reconfigurable neuroprocessor for self-organizing feature maps. Neurocomputing **112**, 189–199 (2013)
21. Asanović, K.: A fast Kohonen net implementation for spert-II. In: Mira, J., Moreno-Díaz, R., Cabestany, J. (eds.) IWANN 1997. LNCS, vol. 1240, pp. 792–800. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0032538
22. Porrmann, M., Witkowski, U., Ruckert, U.: A massively parallel architecture for self-organizing feature maps. IEEE Trans. Neural Netw. **14**, 1110–1121 (2003)
23. Perelygin, K., Lam, S., Wu, X.: Graphics processing units and open computing language for parallel computing. Comput. Electr. Eng. **40**, 241–251 (2014)
24. Kirk, D.B., Hwu, W.W.: Programming Massively Parallel Processors. Elsevier, Amsterdam (2013)
25. Rauber, T., Rünger, G.: Parallel Programming: For Multicore and Cluster Systems. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-04818-0
26. Mittal, S., Vetter, J.S.: A survey of CPU-GPU heterogeneous computing techniques. ACM Comput. Surv. **47**, 69:1–69:35 (2015)
27. De, A., Zhang, Y., Guo, C.: A parallel image segmentation method based on SOM and GPU with application to MRI image processing. Neurocomputing **198**, 180–189 (2016)
28. Mukherjee, S., Sun, Y., Blinzer, P., Ziabari, A.K., Kaeli, D.: A comprehensive performance analysis of HSA and OpenCL 2.0. In: 2016 IEEE International Symposium on Performance Analysis of Systems and Software (2016)
29. Khronos OpenCL: OpenCL Specification (2014)
30. Lichman, M.: UCI Machine Learning Repository. http://archive.ics.uci.edu/ml
31. Mustapha, M.F., Abd Khalid, N.E., Ismail, A.: Evaluation of parallel self-organizing map using heterogeneous system platform. J. Appl. Sci. **17**, 204–211 (2017)

32. Yasunaga, M., Tominaga, K., Kim, J.H.: Parallel self-organization map using multiple stimuli. In: International Joint Conference on Neural Networks, IJCNN 1999 (Cat. No. 99CH36339), vol. 2, pp. 1127–1130 (1999)
33. Fränti, P., et al.: Clustering datasets. http://cs.uef.fi/sipu/datasets/
34. Moro, S., Cortez, P., Rita, P.: A data-driven approach to predict the success of bank telemarketing. Decis. Support Syst. **62**, 22–31 (2014)
35. Berkhin, P.: A survey of clustering data mining techniques. Group. Multidimens. Data **25**, 71 (2006)
36. Han, J., Kamber, M., Pei, J.: Data preprocessing. In: Data Mining Concept and Techniques, pp. 83–134 (2012)
37. Nawi, N.M., Atomi, W.H., Rehman, M.Z.: The effect of data pre-processing on optimized training of artificial neural networks. Procedia Technol. **11**, 32–39 (2013)
38. Hennessy, J.L., Patterson, D.A.: Computer Architecture, Fourth Edition: A Quantitative Approach. Morgan Kaufmann Publishers Inc., San Francisco (2006)