Journal of Engineering and Applied Sciences 13 (2): 314-320, 2018
ISSN: 1816-949X

Parallel Self-Organizing Map using MapReduce in GPUs Environment

Faeez Abd Rashid, Noor Elaiza Abd Khalid, Muhammad Firdaus Mustapha and Mazani Manaf
Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Shah Alam,
40450 Selangor, Malaysia

Abstract: One of the drawbacks of MapReduce characteristic is overlap communication. It causes implementation inefficiency in the GPUs environment. However, this can be overcome using incremental reduction method. This method will enhance the communication process on GPUs environment as an alternative to execution using CPU. This enhancement is based on Python with support of CUDA technologies which can execute this whole process in GPUs environment. In order to achieve the good performance, this study is proposing to design the MapReduce with incremental reduction and then to construct it and finally to test the enhancement method to the self-organizing map with handwriting dataset.

Key words: Self-organizing map, enhanced mapreduce, incremental reduction, graphical processing units, enhancement, method

INTRODUCTION

Large data relates to volume, variety and velocity that impose pressure in managing data by clustering and extracting value towards predictive analytics and decision-making (Chen and Zhang, 2014). Clustering is an important unsupervised learning problem that deals with organizing objects into groups whose members have similar patterns/characteristics which come with high computational costs (Feizi-Derakhshi and Zafarani, 2012).

Kohonen's Self-Organizing Map (SOM) is an unsupervised clustering algorithm that maps multidimensional data onto geometric relationships topology-preserving map which constitute good clustering but high in computational complexity (Kohonen, 2013).

This algorithm differs from other artificial neural networks in terms competitive learning as compared to back propagation with descending gradient which is based on learning through correction of errors. Furthermore, it preserves the input space topological properties by employing a neighborhood function. It has been dubbed as one of the successful clustering algorithm. Conversely, its high computational complexity makes processing time consuming (Kohonen, 2013).

As the computational complexity increases, the need for higher computing power is in big demand. Even with the invention higher powered Central Processing Unit (CPU) such as Duo-core and the quad core could not satisfy the need for better and faster processing hardware. The success of gaming using the GPU has diverted computer scientist from the super computer which are large and not portable to focus on the ability of the GPU as their faster and more efficient processing power. The advent of GPU has managed to increase the processing power in small computer such as personal computers and laptops. Present day applications needs to process large amount of data in a timely manner. Recent research in the programmability of GPUs has allowed graphics hardware to be used as general purpose platform for computationally challenging tasks (Wittek and Daranyi, 2013). GPU has provided a good platform in terms of hardware. Graphical Processing Unit (GPU) computing has been proved to optimize for large data of graphics applications and high throughput of floating-point operations (Perelygin *et al*., 2014). However, designing a good algorithm that was previously implemented sequentially is not a trivial task. This involves modification of the algorithm that can utilize the GPU capability. The modification includes designing a framework that can utilize this hardware.

NVIDIA created CUDA to increase the speed an algorithm on parallel platform. It paves the way of harvesting the capability of the Graphics Processing Unit (GPU) to significantly improve computing performance. MapReduce is one of big data technology that emerges to overcome high computational challenges in an organization posed by the massive creation of data in various machines and media (Wittek and Daranyi, 2013). It is a well know paradigm to distribute load in data-intensive tasks (Wittek and Daranyi, 2013). Many

Corresponding Author: Faeez Abd Rashid, Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Shah Alam, 40450 Selangor, Malaysia

researchers have migrated to MapReduce framework to provide a way to utilize the GPU. Many researchers have deliberated on applying GPUs in problem domains involving visualizations and scientific applications. GPU plays the important domain to process high computational complexity algorithms in MapReduce framework (Elteir *et al*., 2010). MapReduce frameworks are usually implemented for processes workloads that are incremental and repetitive in nature. Nonetheless, this framework research in overlapping communication which create extra overhead making inefficient during GPU tasks distribution (Elteir *et al*., 2010).

Since, the learning nature of Kohonen network algorithms consists of repeating steps that have very little changes in terms of input data. Since, incremental MapReduce frameworks are usually implemented for processes research loads that are incremental in nature, hypothetically it should be able to improve speed of the SOM algorithm.

This study proposes to enhance Kohonen's Self-Organizing Map (SOM) clustering techniques by incorporating incremental reduction MapReduce framework to overcome overlapping communication in MapReduce. The number of tasks being reduced is declared at the start of the job. This is very similar to the original MapReduce framework. Reduction begins within a reduce task when sufficient number of mapped outputs are received. The results from this process are locally stored. These processes are iterated and will terminate when all the mapped output has been retrieved (Elteir *et al*., 2010).

Literature review: Many clustering based algorithm such as Kohonen network (SOM) needs a lot of repetitive calculation that makes it slow to produce results. GPU on the other hand helps to speed up processing. Many attempts has been made to speed up algorithm such as Kohonen network (Gajdos and Platos, 2013; Moraes *et al*., 2012). Previous studies shows interconnection between GPUs and MapReduce are challenged by the overlapping communication with computation (Wittek and Daranyi, 2013). As pointed out in the previous section that the issue of overlapping communication in MapReduce can be overcome by incremental reduction MapReduce method (Elteir *et al*., 2010). This process of MapReduce in GPUs environment is chosen as the framework or based of operation of self-organizing map.

Self-organizing map: Self-organizing map is an unsupervised competitive learning neural network characterized by clustering and visualization properties. Constructed as a learning algorithm for numeric (vector) data, it has been applied in wide range of applications in big data analysis including various fields such as medical, businesses, bioinformatics and satellite (Gogoglou *et al*., 2016).

Kohonen (2013) introduced an efficient Self-Organizing Map (SOM) that displays hidden data topological structures in one or two dimensional space. In spite of its numerous advantages there still exist some demerits quality associated with Kohonen neural network that remains unexplored. Being an unsupervised neural network, the algorithm inevitably dependents on massive iterations that classify this algorithm as data intensive computing complexity.

Graphical processing units: Graphical Processing Units (GPUs) begin its life as hardware to improve the performance of graphics utilized in games. In time the GPU become a focal point of computer scientist to improve the performance of high computational complexity of their algorithms. This in turns forces the GPU industry to migrate the GPU design to cater for a more general-purpose computation and called it the General Purpose Graphic Processing Unit (GPGPU) (Kirk and Hwu, 2013). In the initial phase the GPGPU were applied on programming a complex calculator that involves a set of fixed operations to produce a more desirable results.

Due to the original idea of creating the GPU, initially the GPU utilization is mostly linked to graphics related processes such as 2/3-dimensional images. The process involves graphical primitives which are high in complexity making it difficult in terms of design. However, as year goes the development of a more robust high level third party language has manage to remove the graphics component. Further along, the invention of non-graphics linked languages has made it possible to apply the GPGPU on as general purpose applications. This opens the horizon for many researchers to harvest the ability of GPGPU in speeding up high computational complexity algorithms. Several researchers have successful speedup their algorithms (Wittek and Daranyi, 201; Lachmair *et al*., 2013).

GPUs executable environment: Programming GPGPU is not a simple task using low level programming language. Recently, many very high level language has been introduced to simplify the programming task. Python is one of the new high level languages that have gain popularity due to its easy and manageable interpreted language characteristics. However, it has always been seen as a laid back language for high-performance computing (Nvidia, 2017) which significantly changed when continuum analytics came up with NumbaPro

Python compiler. CUDA Python also run on NumbaPro Python compiler as part of the Anaconda accelerate which allows programmer to reap the benefits of rapid iterative development and speed by utilizing both CPUs and Nvidia GPUs processing units. Thus, adapting regular Python applications using this available tool on critical data intensive iterative functions on GPUs will accelerate the application (Nvidia, 2017). One of the success story example is the MandelBrot calculation that was accelerated with CUDA Python nineteen times speed-up over the CPU-only accelerated version using GPGPUs and a 2000 times speed-up over pure interpreted Python code. Whereas an experiment on Monte Carlo Option Pricer with CUDA Python manage achieved thirty times speed-up over interpreted Python code (Nvidia, 2017).

MapReduce framework: MapReduce framework contains mapping and reducing functions. Map and reduce were previously common in functional programming. However, in MapReduce framework the application of mapping and reducing functions are used for a different purpose (Dean and Ghemawat, 2008). This framework was originally design for processing I/O-bound and data intensive. The main architecture the hardware is similar to an organization consisting of entities where a leader is appointed to lead several subordinates as workers. These entities are refered as master node and worker nodes in the computer harware which forms a cluster. MapReduce framework takes advantage of this architecture. The mapping step involves the master node to subdivide the input data or processes into smaller subproblems. Each subproblems are mapped to the workers node. The mapping step may be iterated by taking the worker node as master node which creates a multilevel tree structure. Finally, the root node of the tree will be processed and the result will be pass to the node higher up the tree. This process is called the reducing step. This mapping and reducing steps are portray in Fig. 1.

Figure 1 shows the run-time system concurrently split tasks into instances in the map tasks phase with the sub input data that are processed at the root of the tree to produce intermediate results. These intermediate results are in turn copied to the reduce tasks phase. This process are iterated until the final output are produced. All the input data are stored as key/value pairs for efficient data indexing and partitioning. The original MapReduce design enforced are bounded with the rules where the reduce phase only begins when the map tasks phase is completed.

Incremental reduction: Incremental Reduction (IR) is a framework that defines the number of task reduction at the start of the process which is similar to the original

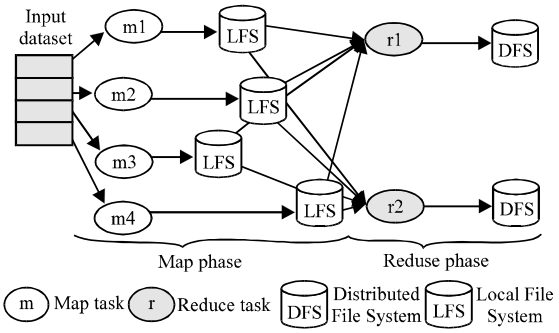

Fig. 1: MapReduce framework based on Perelygin *et al.* (2014)

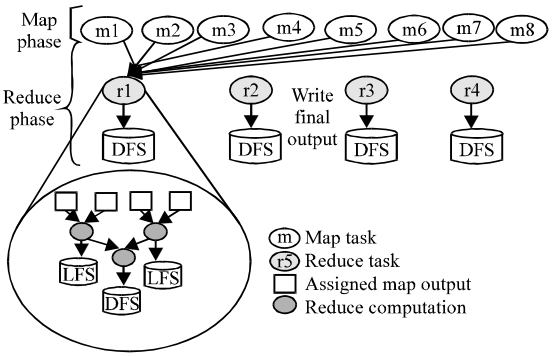

Fig. 2: Incremental reduction in MapReduce based on Elteir *et al.* (2010)

MapReduce framework. However, the reduction phase begins early in the reduce task (Elteir *et al*., 2010). The reduction phase starts immediately when enough number of map outputs which are stored locally are received within a reduction task. These process is repeated until the map outputs are completely generated.

IR framework consists of three phases as shown in Fig. 2. The first phase is the shuffling phase. In this phase, the intermediate results of upper level mapped tasks are copied from the output of all its sub map tasks. The second phase is sorting where the retrieved intermediate results are sorted and merged according to their keys. The final phase is the application of reduce function on the values associated to each key. The shuffling phases are usually run concurrent with the sorting phase to enhance performance (Elteir *et al*., 2010).

MATERIALS AND METHODS

This research proposes to compare the performance of Kohonen network constructed in the original MapReduce and an improvised incremental reduction MapReduce framework.

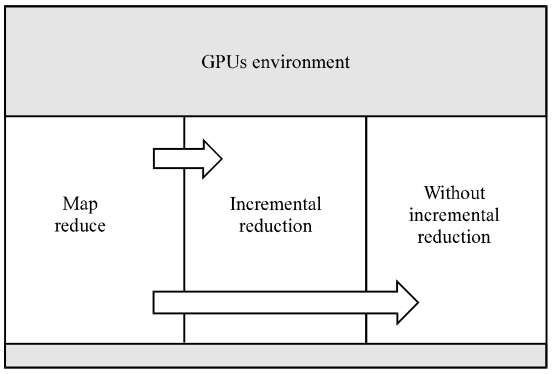

Fig. 3: Process flow of MapReduce intergrate with method on GPUs environment

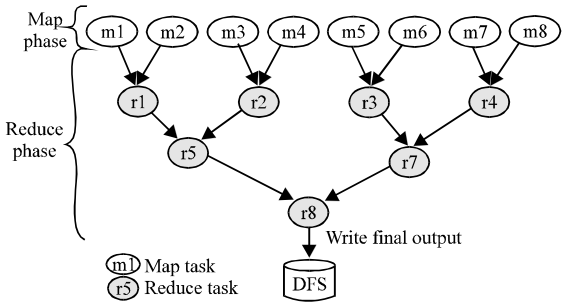

Fig. 4: Hierarchical reduction in MapReduce based on Elteir *et al*. (2010)

This research improvises incremental reduction framework to overcome the issue overlapping communication on MapReduce process (Elteir *et al*., 2010) in the CPU and GPUs environment. Figure 3 shows the research framework with the intention of comparing the performance of utilizing the hierarchical MapReduce (original MapReduce) and with the incremental reduction MapReduce framework in the GPU environment.

The algorithm incorporates into the two MapReduce frameworks with Kohonen network for the purpose of performance measurement. Both frameworks are constructed in the GPU environment.

Hierarchical MapReduce: The original MapReduce framework which is also called the hierarchical reduction framework is also constructed. This framework is depicted in Fig. 4.

Figure 4 shows how the hierarchical reduction purposely overlapped the map and reduces phases. The reduce phase aggregates the partly reduced tasks forming a tree-like hierarchy structure.

Mapreduce with incremental reduction: The incorporation of incremental reduction into MapReduce framework involves rendering the reduction task as early

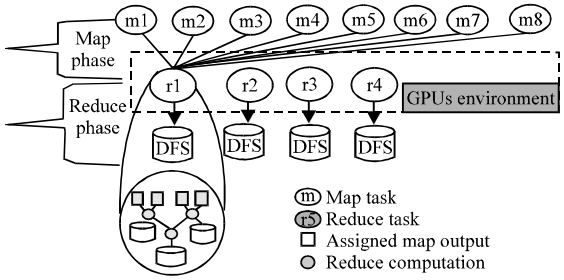

Fig. 5: Incremental reduction in GPUs environment

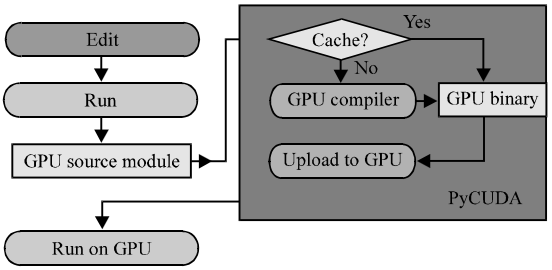

Fig. 6: Workflow of PyCUDA GPU program compilation based on Klockner *et al*. (2012)

as possible in the MapReduce framework. This improvised structure of incremental reduction MapReduce is depicted in Fig. 5. The main aim is to enhance the MapReduce performance so as to overcome the issue of overlap communication. This framework is implemented using CUDA technologies from Nvidia.

Python in GPU environment: As previously discussed, python language are used to develop the MapReduce framework by utilizing CUDA technologies (GPUs) called as PyCUDA executing on GPUs environment (Klockner *et al*., 2012). The overall research flow that utilizes the PyCUDA is shown in Fig. 6.

The gray shaded area is an automated PyCUDA process of accumulation (GPU compile process) and caching. The Kohonen network algorithm structure are implemented in Python programming language which incorporates incremental reduction MapReduce framework of the utilization of available GPUs on a Jupyter notebook.

This research proposes to compare the performance of Kohonen network in hierarchical MapReduce and incremental reduction MapReduce framework simulation.

Hierarchical method: This method just reads the data or files which are then distributed by the mapping task in the map phase. This method then reduces the distribute data by passing the parameter to the assigned reduce tasks.

Incremental reduction method: In this method, begin reading the data or file without the distributing process. This method reduce the data by passing the parameter to the assigned reduce tasks.

Both methods is constructed as structure of MapReduce framework which utilizes the built in python library as array management and parallel processing. The testing phase involves using Kohonen network as testing tool to determine the efficiency of the framework.

Testing with Kohonen network: Kohonen network algorithm is selected as the testing algorithm due its unsupervised characteristics of data and iterative intensive properties. This algorithm is used as a case study to compare the performance of both MapReduce frameworks by measuring the speed of processing. Kohonen network are adapted to fit into the hierarchical and incremental reduction framework.

Kohonen network also known as Self-Organizing Map (SOM) is a neural network that visualizes the clustered output. The structure of a SOM consists of an m-dimensional grid (usually bi-dimensional) is made up of a set of neurons, called nodes. Each node is associated to a weight vector which has the same dimension as the input samples used for training the network (Kohonen, 2013). The adjacent nodes are arranged interconnecting in a topological network that creates a topological neighborhood (or map). The most commonly used neighborhood is rectangular or hexagonal. During the training stage, similar weight vectors will migrate and clustered into adjacent nodes (Kohonen, 2013).

The basic self-organizing map training algorithm consists of four steps. In the first step is to initialize the weight of neuron nodes. The second step is to calculate distance between input space and neurons. The metric used to identify the distance is the Euclidean distance. The third step is to find the closest neuron which labeled as Best Matching Unit (BMU). Finally, during adaptation, the weight vectors of the winner node and its neighbors are updated (Kohonen, 2013).

The self-organizing map is constructed by importing the available functions in the Python PyMVPA library collection. These functions are then incorporated into the hierarchical and incremental reduce MapReduce frameworks to be executed on the GPU environment.

This process of MapReduce in self-organizing map is happen when self-organizing map find the matching find the nearest neuron that have their closest weight therefore the MapReduce is doing their job by reducing the task of finding using MapReduce algorithm in finding the nearest neuron. The process is shows as illustrate.

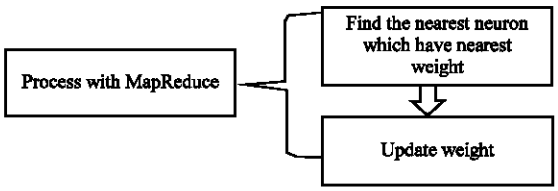

Fig. 7: Process of MapReduce in self-organizing map

Figure 7 shows how SOM are incorporated into the MapReduce framework to be executed on the computers GPU. The SOM algorithm adapted into the MapReduce frameworks are step 3 which is finding the best matching neuron and step 4 which is updating the weight of neuron.

The data collected for this experiment is a multi dimensional data set of handwriting data which consists of records and five attributes. The main focus of the study is to access the performance of the MapReduce framework which is the time taken to execute the whole process. The SOM map are of no importance. This syntax of the process training SOM is "som.train()". The speed are captured using "%time". This will display the time taken to process the SOM in seconds The evaluation are made according to the following steps.

Step 1: Capture the process speed value of both hierarchical and incremental reduction MapReduce framework.

Step 2: Compare the speed of the two framework and calculates the percentage of the speed based on the highest and the lowest speed. The percentage are calculated based on the following Eq. 1:

$$\text{Percentage} = \frac{\left((\text{Highest-Lowers})\times 100\right)}{\text{Highest}} \quad (1)$$

Step 3: Repeat the experiment.

RESULTS AND DICUSSION

In this study, we evaluate the results after repeating the experiment for five times.

Experimental platform: The process of parallelizing the self-organizing map with MapReduce are done on Nvidia Geforce GT 635M 2GB. This GPU can deploy CUDA technology which is important to execute MapReduce framework design on GPU environment. Jupyter notebook are running based on Google Chrome browser which is on GPU environment.

Table 1: MapReduce performance in GPUs environment process self-organizing map algorithm by using handwriting data

Test rounds	Incremental reduction MapReduce (sec)	Normal MapReduce (sec)	Percentage
1	8.99	9.34	3.75
2	27.80	29.60	6.08
3	9.02	10.50	14.10
4	9.08	9.33	2.68
5	9.23	9.33	1.07

Result of MapReduce performance: Table 1 shows the resulting performance of the self-organizing map algorithm constructed in the hierarchical and incremental reduced MapReduce framework. The percentage difference between the MapReduce frameworks are presented in the Table 1.

Table 1 shows the speed of processing SOM on both in frameworks. It is found that the performance of the incremental MapReduce framework supersede the Hierarchical framework in all the five experiments. The low percentage of speed up is small because it was tested on small multidimensional data.

The percentage of improvement is different in every testing round so the result is not consistent in term of processing in different testing round times due to the hidden processing during the test. Nevertheless, the incremental reduction MapReduce does improve processing speed.

CONCLUSION

The hierarchical MapReduce framework issue of overlap communication can be improved with incremental reduction framework executed on the GPU environment. This research has verified that adapting Mapreduce framework with incremental reduction can speed up the parallelization process of SOM or other similar algorithms.

RECOMMENDATIONS

For the future research, this research needs to extend into another test module that employs different techniques with the aim to seek the different in performance, finding the reason of inconsistency of result and in depth analysis. As well, larger dataset should be considered in the experiments.

ACKNOWLEDGEMENTS

This research was funded by Ministry of Higher Education (MOHE) of Malaysia, under the FRGS, grant No. FRGS/81/2015. Researcher also would like to thank the Universiti Teknologi MARA for supporting this study.

REFERENCES

Chen, C.L.P. and C.Y. Zhang, 2014. Data-intensive applications, challenges, techniques and technologies: A survey on big data. Inform. Sci., 275: 314-347.

Dean, J. and S. Ghemawat, 2008. MapReduce: Simplified data processing on large clusters. Commun. ACM, 51: 107-113.

Elteir, M., H. Lin and W.C. Feng, 2010. Enhancing mapreduce via asynchronous data processing. Proceedings of the 2010 IEEE 16th International Conference on Parallel and Distributed Systems (ICPADS), December 8-10, 2010, IEEE, New York, USA., ISBN:978-1-4244-9727-0, pp: 397-405.

Feizi-Derakhshi, M.R. and E. Zafarani, 2012. Review and comparison between clustering algorithms with duplicate entities detection purpose. Intl. J. Comput. Sci. Emerging Technol., 3: 108-114.

Gajdos, P. and J. Platos, 2013. GPU based parallelism for self-organizing map. Proceedings of the 3rd International Conference on Intelligent Human Computer Interaction (IHCI 2011), Volume 179, Prague, Czech Republic, August, 2011, Springer-Verlag, Berlin, Heidelberg, pp: 231-242.

Gogoglou, A., A. Sidiropoulos, D. Katsaros and Y. Manolopoulos, 2016. A scientists impact over time: The predictive power of clustering with peers. Proceedings of the 20th International Symposium on Database Engineering and Applications, July 11-13, 2016, ACM, Montreal, Quebec, Canada, ISBN: 978-1-4503-4118-9, pp: 334-339.

Kirk, D.B. and W.W. Hwu, 2013. Programming Massively Parallel Processors: A Hands-on Approach. 2nd Edn., Elsevier, New York, ISBN: 978-0-12-381472-2, Pages: 514.

Klockner, A., N. Pinto, Y. Lee, B. Catanzaro and P. Ivanov *et al*., 2012. PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation. Parallel Comput., 38: 157-174.

Kohonen, T., 2013. Essentials of the self-organizing map. Neural Netw., 37: 52-65.

Lachmair, J., E. Merenyi, M. Porrmann and U. Ruckert, 2013. A reconfigurable neuroprocessor for self-organizing feature maps. Neurocomputing, 112: 189-199.

Moraes, F.C., S.C. Botelho, N.D. Filho and J.F.O. Gaya, 2012. Parallel high dimensional self organizing maps using CUDA. Proceedings of the 2012 Symposium on Robotics and Latin American Robotics (SBR-LARS), October 16-19, 2012, IEEE, Rio Grande, Brazil, ISBN:978-1-4673-4650-4, pp: 302-306.

Nvidia, 2017. GPU accelerated computing with python. Nvidia, Santa Clara, California. https://developer.nvidia.com/how-to-cuda-python.

Perelygin, K., S. Lam and X. Wu, 2014. Graphics processing units and open computing language for parallel computing. Comput. Electr. Eng., 40: 241-251.

Wittek, P. and S. Daranyi, 2013. Accelerating text mining workloads in a map reduce-based distributed GPU environment. J. Parallel Distrib. Comput., 73: 198-206.