Training Acceleration of Multi-Layer Perceptron using Multicore CPU and GPU under MATLAB Environment

Shefa A. Dawwd / Assist. Prof.
shefa.dawwd.2014@ieee.org

Noor M. AL Layla
aneng.noor@gmail.com

Computer Engineering Department
College of Engineering, University of Mosul, Mosul-Iraq

Abstract

Training of Artificial Neural Networks (ANNs) for large data sets is a time consuming mission. In this paper, accelerating the training of artificial neural network is achieved by a parallel training using either Multicore Central Processing Unit (CPU) or General Purpose Graphics Processing Unit (GPGPU). The training is implemented using five datasets with diverse amounts of patterns and with different neural network parameters in Multilayer Perceptron (MLP). The results show a significant increase in computation speed, which is increased nearly linear with the number of cores in multicore processor for problems with medium and large training datasets. Also, a considerable speed up is achieved when the GPU is used to train the MLP with the large training datasets. While a single core processor is a better choice when the data set size is small. The optimal number of cores or the type of the parallel platform should be employed according to the load of computation.

Key Words: Parallel Training, Parallel programming, Neural Network, Multicores processor, Graphics Processing Unit

تسريع تدريب البيرسبترون متعدد الطبقات باستخدام وحدة المعالجة المركزية متعددة الاقطاب و وحدة المعالجة الرسومية في بيئة ماتلاب

د.شفاء عبد الرحمن داؤود

نورموفق الليلة

قسم هندسة الحاسبات / كلية الهندسة / جامعة الموصل

الخلاصة

يستغرق تدريب الشبكات العصبية الاصطناعية زمن كبير خاصة في حالة كون أنماط وبيانات الشبكة كبيرا، في هذا البحث تم تنفيذ تسريع تدريب الشبكات العصبية الاصطناعية عن طريق تنفيذ التدريب المتوازي للشبكة العصبية الاصطناعية باستخدام اما وحدة المعالجة المركزية المتعددة النوى او وحدة المعالجة الرسومية للأغراض العامة، نفذ التدريب باستخدام خمسة بيانات مختلفة الأحجام والأعداد على شبكات عصبية اصطناعية مختلفة العناصر، أظهرت النتائج زيادة في سرعة تدريب الشبكات العصبية الاصطناعية وكانت سرعة التدريب للشبكة تزداد بشكل خطي تقريبا مع زيادة عدد النوى لوحدة المعالجة المركزية للشبكات ذات البيانات المتوسطة والكبيرة او عند استخدام وحدة المعالجة الرسومية للشبكات العصبية ذات الأنماط والبيانات الكبيرة جدا. بينما يفضل استخدام وحدة المعالجة المركزية وحيدة النواة مع بيانات تدريب قليلة. عدد النوى الامثل او نوع منصة التوازي يجب ان توظف اعتمادا على كم الحسابات.

Received: 23 – 4 - 2014

Accepted: 5 – 5 - 2015

.

1. Introduction

Neural networks are efficient to solve problems where mathematical modeling of the problem is difficult. They are used to exceeded problems including feature extraction, noise reduction, classifications and image matching. Typically large data is required to train the neural networks. As the size of the neural network increases the time required to train increases exponentially. Many Attempts are made to reduce the training time of the neural network by reselecting initial values[1][2]. Training a neural network for large and complicated problems normally leads to use very large amounts of training data with hundred thousand or even millions of patterns. Such training can take weeks and even months to get the desired precision. In the same way, finding an optimal neural network configuration requires a certain amount of cross-validation testing, which can be also very time consuming. Therefore there is a need to speed up the training process of neural networks, especially for large training datasets. Recently multicore and multithreaded CPUs with shared memory are a cost effective way of obtaining significant increases in CPU performance. An exponential growth in performance is expected in the near future from more hardware threads and cores per CPU [3].In the other way there are some attempts to speeding the training implemented by Graphical Processing Unit (GPU) [4]. Researches focused their attention recently on parallelizing a variety of computational intelligence algorithms [5-8] using the new processors. For neural networks two basic methods of parallelization can be defined: parallelizing the neural network structure and parallelizing the training process [9]. The first method exploits the parallel nature of neural networks, and assigns to each processing node (neuron) a separate thread [5]. All neurons in one layer are processed simultaneously and synchronized before propagating into the next layer. The second method is to assign a part of the training dataset to each thread and train them simultaneously. This method is covered in [6], where a three-layer perceptron neural network is parallelized and tested using two and eight threads. The same technique is implemented for dual-core processors in [7]. MATLAB is used in technical computing, it is preferred to develop an algorithm in MATLAB first. Then the code can be converted into C Language for real life demands. In this paper acceleration of training the neural network made for Multi-Layer Perceptron using Backpropagation algorithm and the sigmoid nonlinear activation function implemented on Core i7 processors and The GeForce 610M contain 48 cores under MATLAB environment are achieved.

Most of the nowadays laptops or desktop PCs are provided with the above mentioned parallel platforms. Our goal is to investigate the acceleration of training using such platforms under the most commonly popular parallel programming paradigm (MATLAB) used in technical computing. If the performance of network is adequate, then there is no need to transfer the code to other parallel programming language which may be complicated to be achieved.

2.Multi-Layer Perceptron (MLP)

MLP model has an input layer, a hidden layer and an output layer. This structure of the network allows many calculations to be performed at the same time to decrease the processing time [10]. The hidden layer is connected with others by interconnected weights. A general structure of an MLP neural network which consists of one input layer, one single hidden layer and one output layer, illustrated in Figure (1).

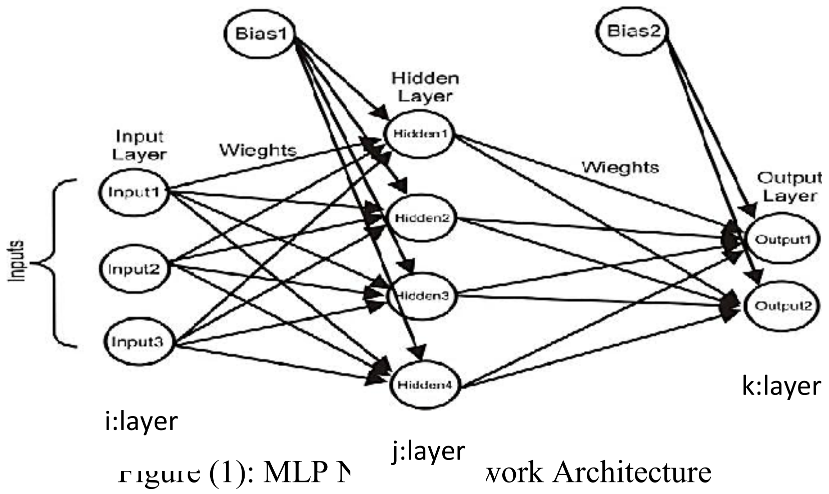


Figure (1): MLP N work Architecture

3. Back Propagation (BP) Learning Algorithm

This algorithm can be used with any multi-layer neural network with any activation function that can be driven to get outputs of their neurons through repetitive process, The central idea of this algorithm is that the errors of the hidden layer unit are determined by backpropagating the errors of the units of the output layer. Therefore this method is known as backpropagation learning rule. Thus backpropagation can be considered as learning rule algorithm to multiple layer networks and nonlinear differentiable transfer functions[12]. Each iteration of this algorithm consists of several steps. The steps of back propagation algorithm can be reviewed in [13].

Typically many epochs are required for training a backpropagation neural network. The original algorithm updates the weights after each training pattern is presented. A common variation is a batch updating, in which weight updated are accumulated over an entire epoch before being applied.

The choice of initial weights will affect whether the network reaches a global (or only a local) minimum of the error and, if so, how quickly it converges. The update of the weight between two layers depends on both the derivative of the first layer activation function and the activation of the second layer respectively. For this reason, it is important to avoid choices of initial weights that would make either activations or derivatives of activations are zero. The values for the initial weights must not be too large, or the initial input signals to each hidden or output unit will be likely to fall in the region where the derivative of the sigmoid function has a very small value. On the other hand, if the initial weights are too small, the network input to a hidden or output unit will be close to zero, which causes an extremely slow learning. Initialization of the weights and biases must be set to random values between -0.5 and 0.5 or between -1 and 1 or some other suitable interval[10].

The basic procedure for the training of the neural network can be mentioned as follow:

1- Apply the data vector to the network and calculate the corresponding outputs.
2- Compare the actual outputs with the target outputs and determine the error.
3- Determine in which direction to change the weights in order to reduce the error.
4- Apply updated weights.
5- Forward and backward computation will be repeated with each inputs sets until the error for all inputs in training set are reduced to an acceptable value[14].

4. Sequential Training of Artificial Neural Networks

Typically, the ANN takes many epochs before it has learned the problem to within an acceptable error. Sometimes learning is not successful due to a particular random initialization of the connection weights. In these cases learning is stopped after some max number of epochs, and new random weights are computed for another try. There are a variety of factors that work together to cause very long training times for backpropagation. The primary cause is due to the large number of connection weights, which can be hundreds or thousands. Backpropagation performs a gradient descent on the error surface generated by evaluating the mean squared error of the neural network on the training set. Gradient descent can be fooled by local minima of the error surface and can take a very large number of epochs to converge to an acceptable minimum. It is not unusual for more difficult problems to take hundreds of thousands or even millions of epochs. Additionally, training can take a long time due to a large training data size. Also, just processing a single training pattern can take a long time on a complicated or large ANNs[15].

5. Parallel Training of Artificial Neural Networks

There are two techniques to parallelize the training of artificial neural networks depending on the size of the network and the available memory.

5.1 Network Parallel Training (NPT)

Network Parallel Training, is a technique used to parallelize neural network training, In this method the neurons of the ANN are divided across machines in the cluster, so that each machine have a portion of the neural network. Each training pattern is processed by the cluster machines in parallel. To process a single training pattern, communication is required between any cluster nodes containing neurons that are connected by an edge (see Figure 2). Network Parallel Training attacks the training time problem by improving the time to process a single training pattern. It has the potential to work well when implemented on special-purpose parallel hardware, however, itis much less likely to work well on a cluster of workstations. In order to benefit from being split up across multiple machines, the neural network must be large enough to prevent the cost of communication between neurons on different cluster nodes from overwhelming the potential speedup from parallelizing the computation. The network latency and bandwidth on most cluster systems will significantly limit the degree of parallelism possible for training even in large sized ANNs. When the original ANN is of small or medium size, there is even less benefit from an NPT approach as there will be little local computation between communication points [15].

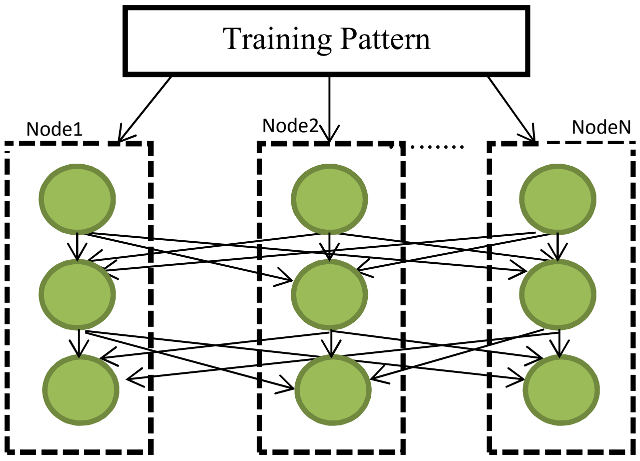


Figure (2): Neurons are divided into multiple nodes NPT.

5.2 Pattern Parallel Training (PPT)

Pattern Parallel Training is a technique used for parallelizing the training of artificial neural networks that is designed to work on cluster computers. The patch training is suitable to be used with this technique. In PPT the full Neural Network is duplicated at each node (see Figure 3). Each node then trains its local copy of the network on a subset of the training data which is selected randomly. When the local computation is complete, nodes broadcast their final weight updates to other nodes. In this system, an epoch consists of the local computation of the weight updates on a subset of the patterns. At the end of each epoch, every node applies the weight updates from the other nodes to its neural network and determines if the training is complete or if another training epoch is needed based on the error condition. The speed-up in training is achieved by reducing the time of performing a single epoch; in parallel each node evaluates just a subset of the full training data each epoch. By communicating only at the end of each epoch, the communication costs of Pattern Parallel Training are reduced[15].

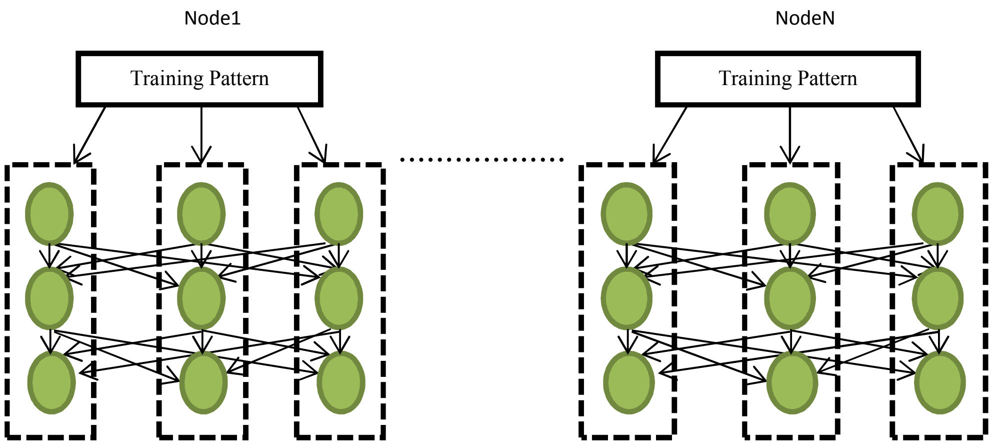


Figure (3): Neural Network is duplicated in PPT.

6 . Implementation Platforms

The CPU, Multicore CPU, and GPU are chosen in this paper as platforms for implementing the sequential and parallel training of MLP using BP algorithm. In what follows, a brief introduction of each platform is overstated.

6 .1 Central Processing Unit

The CPU architecture has only one processing unit in the chip (As seen in figure 4), for performing the arithmetic or logical operations. At any particular time, only one operation can be performed [16].

6.2 Multi-Core Processor

A multi-core processor is a system that comprises of two or more independent cores (or CPUs). The cores are generally integrated onto one integrated circuit die (known as a chip multiprocessor), or they are integrated onto multiple dies on a single chip package, (See figure 5) [17].

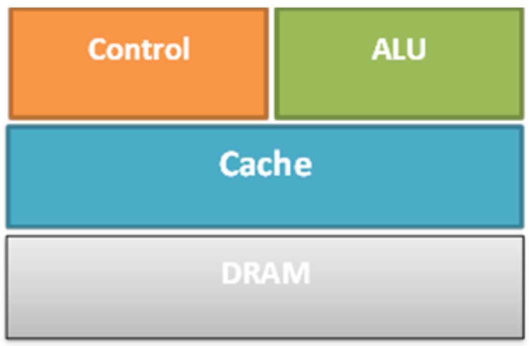


Figure (4) CPU hardware architecture

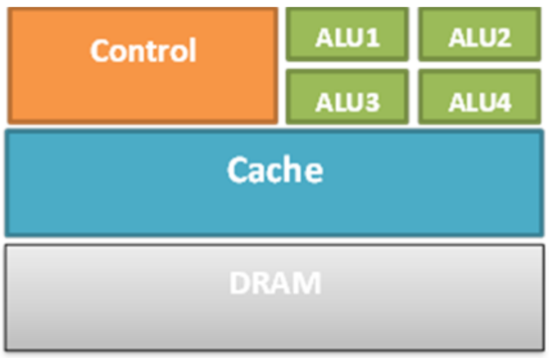


Figure (5) Multicore hardware architecture

6.3 Graphic Processing Unit

GPU is viewed as a computing device operating as a coprocessor to the main CPU (host) processor constructed in a superscalar fashion. A GPU is implemented as an aggregation of multiple so-called multiprocessors, which consists of a number of Single Instruction Multiple Data (SIMD) ALUs integrated as network on a chip as shown in figure (6). A single ALU is called processor. According to the SIMD concept, every processor within a multiprocessor must execute the same instruction at the same time, only the data may vary [18].

Cache	🟠
Control	🟢
ALU	🔵

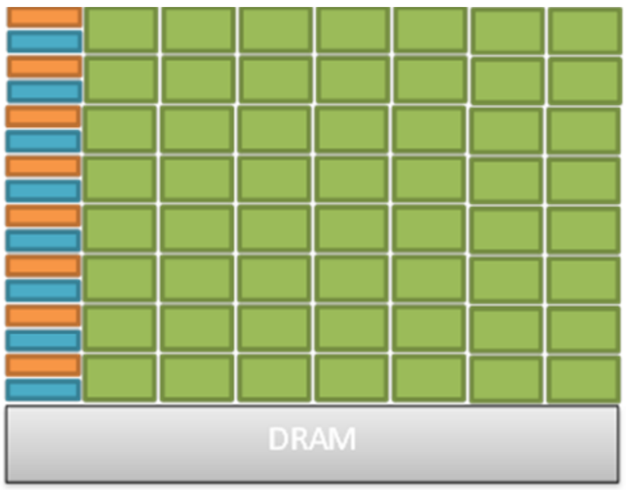


Figure (6) GPU hardware architecture

7. Benchmark, Algorithm, Software and Hardware Models

Using MLP as an encoder is taken as a benchmark to test the proposed training techniques. Different sizes of encoders are used. Then different sizes of networks are used. The input and target for each network is the input pattern and its code respectively. For example, if 8-bits encoder is to be implemented using MLP (Figure 7), then eight neurons in the input layer and the same should be used as a target. The MLP is trained for 2^8 different inputs and 2^8targets (encoded outputs). If the size of the encoder (word length) is enlarged (for example 16-bit encoder has 2^{16}=65,536 words), then the number of training patterns and the training time overhead are increased dramatically. Also, the size of MLP should be enlarged enough to accommodate this change.

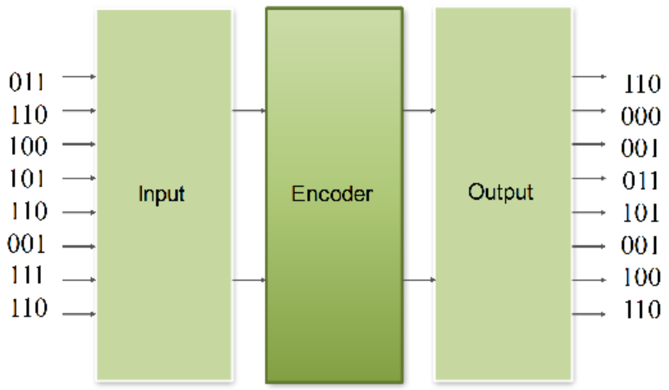
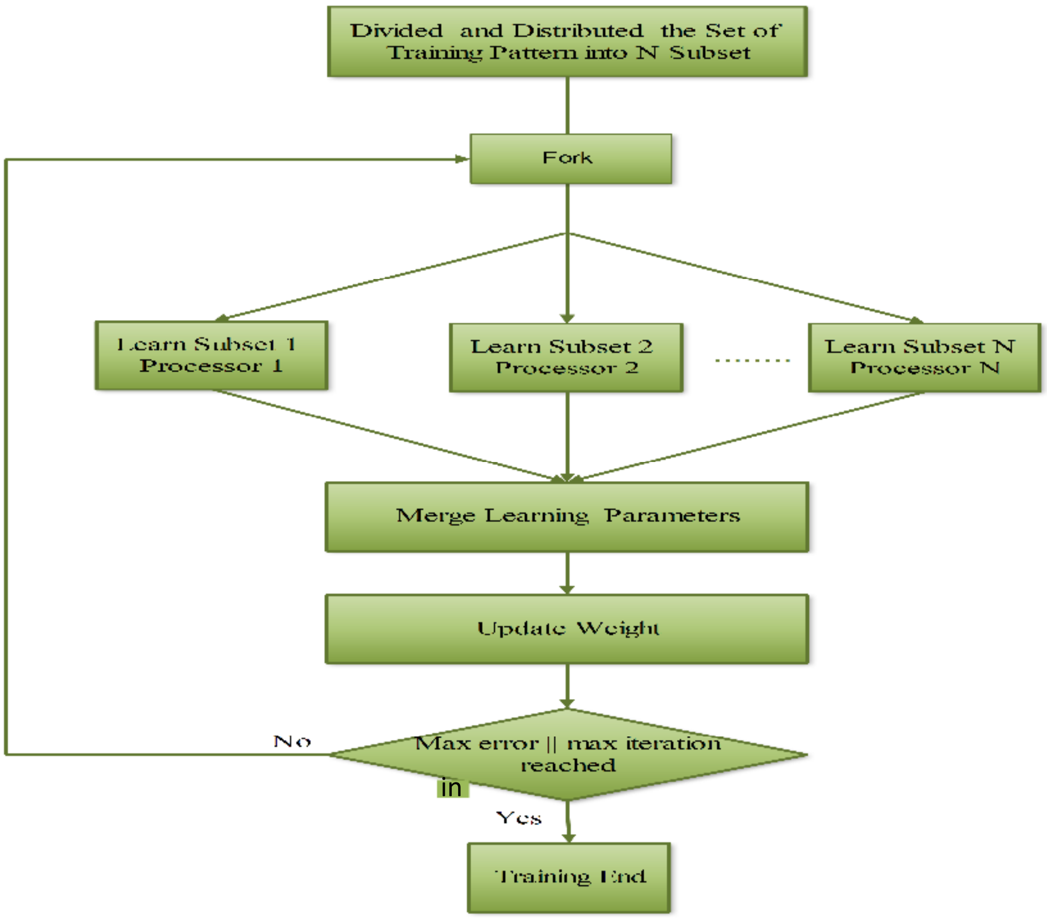


Figure (7) An 8-bit encoder

To cope with this challenge, pattern parallel training algorithm is implemented to train a large word length neural network based encoder (Figure 8). The PPT technique is tested for parallel training of ANN on the five different sizes of training data set (see Table (1)). .


Divided and Distributed the Set of Training Pattern into N Subset
Fork
Learn Subset 1 Processor 1
Learn Subset 2 Processor 2
....
Learn Subset N Processor N
Merge Learning Parameters
Update Weight
No
Max error || max iteration reached
in
Yes
Training End
Figure (8) Parallel Training Algorithm

The computational load is proportional with both the encoder word length and the network size which in turn is determined according to the encoder size. Thus, the computational load is divided into three main categories: small, medium and large.

The PPT is implemented using a Laptop of an Intel(R) Core(TM) i7-2670QM CPU@ 2.20 GHz , 8GB RAM, and the GeForce 610M GPU which contains 48 cores with 1 GB of RAM. The later was installed on a Laptop of an Intel (R) Core(TM) i5-3210M 4 CPUs operating at 2.5 GHz, 6GB RAM. The MATLAB environment software is used. Some special instructions and functions are used to transfer the data and computations from the host processor to the accelerator (GPU) and gather them back to the host such as **gpuArray, nndata2gpu** and **gpu2nndata.** Also, when a multicore processor is used, **matlabpool open** enables the parallel language features within the Matlab language to use parallel instructions such as **parfor** or **spmd** and **matlabpool close** return Matlab to its traditional sequential mode

Teble (1): Different training data sizes for encoders of vairiant word lengths.

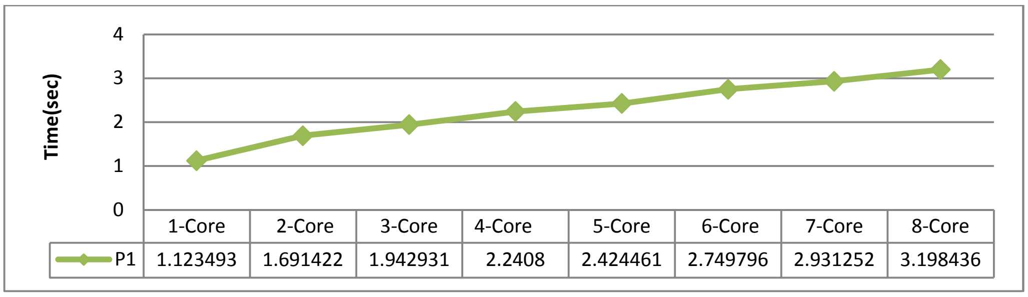
type	**word length (# bits)**	**# Training set patterns**	**Computational Load**
P1	8	256	Small
P2	12	4096	Medium
P3	16	65536	Medium
P4	28	65536	Large
P5	32	65536	Large

8. Experimental Results

As shown in Figure (9), when a multicore processor is used for small samples of training data set, the time required to train the network is even increased when the number of employed cores are increased(see a and b in Figure(9)). The time required to initialize cores and the communications among cores dominants over the benefits acquired from the parallelization when dealing with small training data set. The best time is obtained when using single core processor for P1 data set and two core processor for P2 data set. The communication overhead seems to be vanished if larger samples of training data set are required to train the network. That is because the computation operations become larger than the communication and initialization operations (see c, d and e in Figure(9)).
The speed up was calculated :

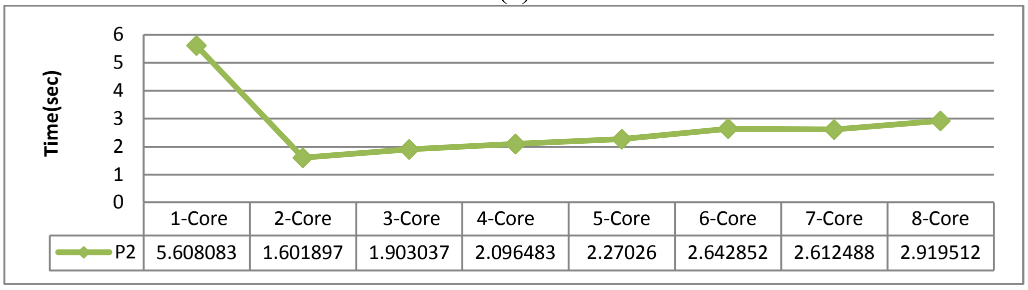
$$\text{Speedup} = \frac{\text{Sequential time}}{\text{Parallel time}} \qquad \dots\dots\dots (1)$$

As much as 4x speed up is achieved over a single core processor, when 8-cores processor is used (Table 2).



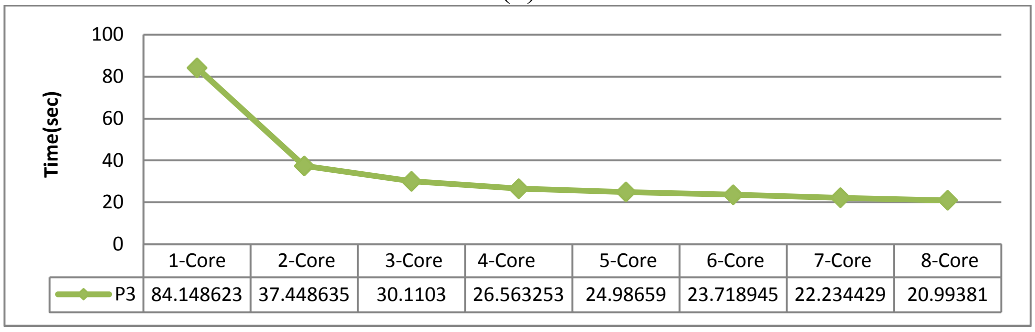
	1-Core	2-Core	3-Core	4-Core	5-Core	6-Core	7-Core	8-Core
P1	1.123493	1.691422	1.942931	2.2408	2.424461	2.749796	2.931252	3.198436

(a)



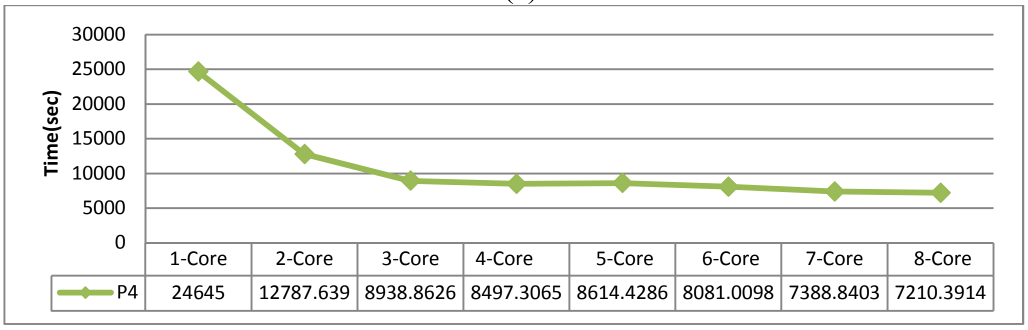
	1-Core	2-Core	3-Core	4-Core	5-Core	6-Core	7-Core	8-Core
P2	5.608083	1.601897	1.903037	2.096483	2.27026	2.642852	2.612488	2.919512

(b)



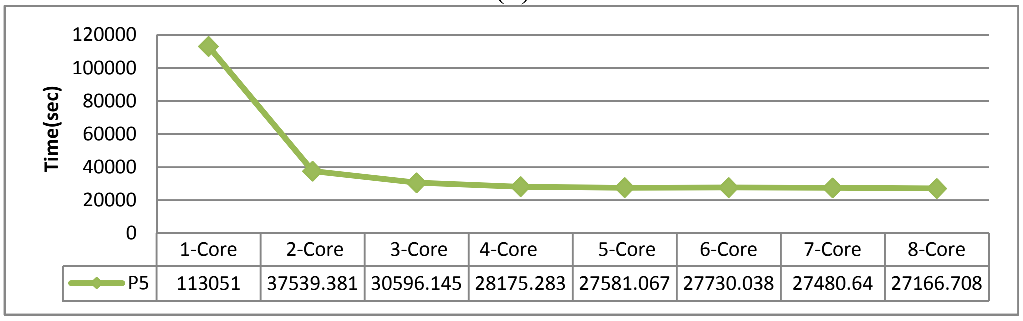
	1-Core	2-Core	3-Core	4-Core	5-Core	6-Core	7-Core	8-Core
P3	84.148623	37.448635	30.1103	26.563253	24.98659	23.718945	22.234429	20.99381

(c)



	1-Core	2-Core	3-Core	4-Core	5-Core	6-Core	7-Core	8-Core
P4	24645	12787.639	8938.8626	8497.3065	8614.4286	8081.0098	7388.8403	7210.3914

(d)



	1-Core	2-Core	3-Core	4-Core	5-Core	6-Core	7-Core	8-Core
P5	113051	37539.381	30596.145	28175.283	27581.067	27730.038	27480.64	27166.708

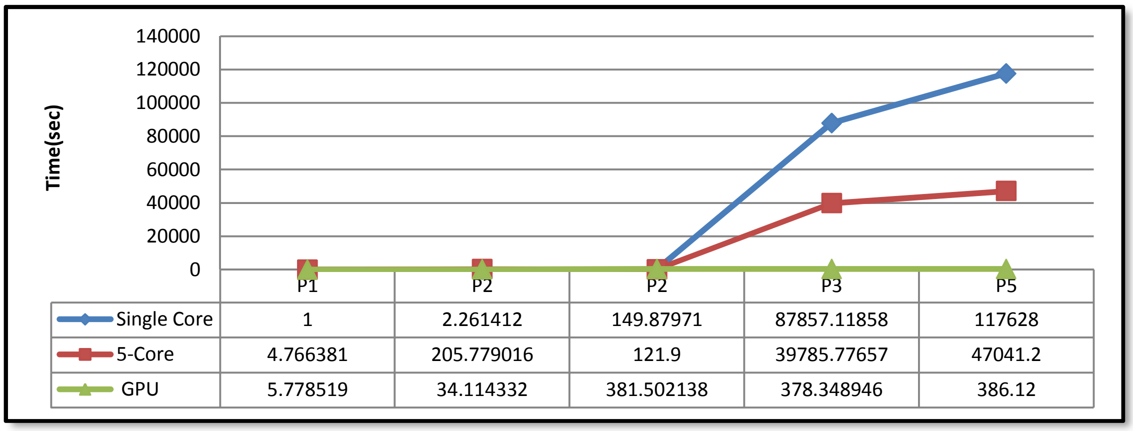
(e)

Figur(9): Time performance using Multicore processor for variant network training set sizes.

Table (2): Best speed up achieved using Multicore CPU(8-Core).

Data	Best Speed up
P1	1.123493
P2	3.500901119
P3	4.008258768
P4	3.417983649
P5	4.16138017

When the GPU is used for implementing the training of the ANN, better results are obtained. Smaller training time for large samples data set is achieved compared with the training time consumed by a single core and a multicore processors (see Figure (10)). As much as 304x speed up is achieved over a single core processor, when GPU is used (Table 3).



	P1	P2	P2	P3	P5
Single Core	1	2.261412	149.87971	87857.11858	117628
5-Core	4.766381	205.779016	121.9	39785.77657	47041.2
GPU	5.778519	34.114332	381.502138	378.348946	386.12

Figure (10): Time performance using Multicore processor and GPU for variant netwok training set size.

Table (3):Best speed up achieved using Multicore CPU and GPU.

Data	Speed Up at Multi core processor(5 Core)	Speed Up at GPU (48 Core)	GPU Utilizations
P1	0.20980	0.17305	28%
P2	0.70949	0.06628	83%
P3	1.22953	0.39286	98%
P4	2.20825	232.211876	98%
P5	2.50053	304.6410449	98%

It is difficult to compare the resulted speedup with previous state of the art published works, since that the parallel platforms are available with diverse types, features and powers. However, one can see that although the Matlab environment is used as a parallel paradigm, a speedup of 304x is achieved in comparison with a speedup of

63x which obtained in [4] where more powerful parallel platform (the NVIDIA card employed is the Tesla C1060 with 240 kernels) is used with CUDA programming codes.

9.Conclusions

Parallel training implementation of MLP using Multcore CPU and GP-GPU is presented in this work. Parallel programming using MATLAB tools with multiple cores and GPU is investigated over five variant sizes of training data set. The optimal number of cores or the type of the parallel platform should be employed according to the load of computation. Employment of too many cores may degrade the performance. It can be concluded that for both parallel platforms used (Multicore and GPU processors), the efficiency of parallelizing the training process increases when the data set size increases. Using a single core processor is a better choice when the data set size is small. That is because that the time required to initialize cores and to transfer data among the shared memory multiple cores dominants over the benefits acquired from the parallelization. Accelerating the training by using SIMD GPU is very advisable for large training data set over the single and multiple core processors. Parallel computation using GPU on a medium dataset is not worthy. A considerable training speed up is achieved using MATLAB which is a simplest and most commonly popular languages used in technical computing.

References

[1] D. Nguyen, B. Widrow, “Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights” ,Proceedings of the international joint conference on neural networks, Vol. 3, Washington, 1990, pp. 21–26.

[2] Huqqani, Altaf Ahmad, et al. "Multicore and GPU Parallelization of Neural Networks for Face Recognition". Procedia Computer Science 18 (2013): 349-358.

[3]Sodan, A.C., Machina, J., Deshmeh, A., Macnaughton, K., Esbaugh, B., “Parallelism via Multithreaded and Multicore CPUs”. Computer 43(3), 24–32 (2010).

[4] X. Sierra-Canto, F. Madera-Ramirez, and V. Uc-Cetina. "Parallel Training of a Back-Propagation Neural Network Using CUDA". Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on. Dec. 2010, pp. 307 - 312. Doi : 10.1109/ICMLA.2010.52 (cit. on p. 103).

[5] Seiffert, U., “Artificial Neural Networks on Massively Parallel Computer Hardware”. ESANN 2002 Proceedings - European Symposium on Artificial Neural Networks, April24-26, pp. 319–330. Bruges, Belgium (2002) .

[6] Turchenko, V., Grandinetti, L., “Efficiency Analysis of Parallel Batch Pattern NN Training Algorithm on General-Purpose Supercomputer”. IWANN 2009. LNCS, vol. 5518, pp. 223–226. Springer, Heidelberg (2009).

[7] Tsaregorodtsev, V., “Parallel Implementation of back-Propagation Neural Network Softwareon SMP Computers”. Malyshkin, V.E. (ed.) PaCT 2005. LNCS, vol. 3606, pp. 186–192. Springer, Heidelberg (2005).

[8] Lotrič, U., Dobnikar, A. “Parallel Implementations of Recurrent Neural Network Learning”. ICANNGA 2009. LNCS,vol. 5495, pp. 99–108. Springer, Heidelberg (2009).

[9] Schuessler, Olena, and Diego Loyola. "Parallel training of artificial neural networks using multithreaded and multicore CPUs.", Adaptive and Natural Computing Algorithms. Springer Berlin Heidelberg, 2011. 70-79.
[10] Alsmadi, M. k. S., Omar, K. B. and Noah, S. A., "Back Propagation Algorithm: The Best Algorithm Among the Multi-layer Perceptron Algorithm", International Journal of Computer Science and Network Security (IJCSNS), Vol. 9 No.4, pp. 378-383, April, 2009.
[11] Bishop, C. M, "Neural Networks for Pattern Recognition", Clarendon Press, Oxford., (1995).
[12] Krose, B. and Smagt, P. V. D., "An Introduction to Neural Networks ", Eighth edition, University of Amsterdam, (1996),.
[13] Faustt, L., "Fundamental Of Neural Networks Architectures, Algorithms and Applications", Prentice Hall, (1995).
[14] Khamas, B. M., "Neural Networks Based Equlializer Applied to FPGA", M.Sc. thesis, College of electrical and electronics Technology, Baghdad, Iraq, pp 1-102, (2006).
[15] Dahl, George, Alan McAvinney, and Tia Newhall. "Parallelizing neural network training for cluster systems." Proceedings of the IASTED international conference on parallel and distributed computing and networks, Innsbruck, Austria. 2008.
[16] B. Jang,"Evaluation and enhancement of memory efficiency targeting general-purpose computations on scalable data-parallel GPU architectures" ,PhD thesis, Department of Electrical and Computer Engineering, Northeastern University, January 01, 2011.
[17] A. Aljebaly," Multi-Core Computing", Department of Computer Science Western Michigan University.
[18] B. Jang," Evaluation and enhancement of memory efficiency targeting general-purpose computations on scalable data-parallel GPU architectures" ,PhD thesis, Department of Electrical and Computer Engineering, Northeastern University, January 01, 2011.

The work was carried out at the college of Engineering. University of Mosul