ISSN: 0975-833X

Available online at http://www.journalcra.com

International Journal of Current Research
Vol. 9, Issue, 06, pp.51818-51822, June, 2017

INTERNATIONAL JOURNAL OF CURRENT RESEARCH

RESEARCH ARTICLE

PARALLELIZATION OF BACKPROPAGATION ALGORITHM AND BENCHMARKING

***,[1]Ishan Borker, [2]Ruchika, [1]Vinoth Kumar, R. and [2]Aditya Kumar Sinha**

[1]Department of Computer Science and Engineering, Veltech Dr. RR and Dr. SR University, Chennai, Tamil Nadu, India
[2]Centre for Development in Advanced Computing (C-DAC), Pune, India

ARTICLE INFO

Article History:
Received 20th March, 2017
Received in revised form
18th April, 2017
Accepted 10th May, 2017
Published online 20th June, 2017

Key words:

Artificial Neural Networks (ANN), Backpropagation algorithm (BPA), Open MP, Fork-join.

ABSTRACT

Back propagation Algorithm is a technique to train Artificial Neural Networks to calculate the gradient of the error function with respect to all the weights. This gradient is then used to update the weights to reduce the error function (https://mattmazur.com/2015/03/17/a-step-by-step back propagation-example/). Back propagation Algorithm is a supervised learning approach in neural networks. Open MP is a model used for parallel programming to improve efficiency and time. It is used to produce more efficient neural networks. This technique executes the algorithm in parallel. This paper summarizes the basic Back propagation Algorithm and measures the time of execution of the serial code. It is then compared with the time of execution of parallel code. Also various methods like code profiling, code optimization are being used. Also, the need to perform benchmarking is required to check the relative performance of the program on different system architectures.

Citation: Ishan Borker, Ruchika, Vinoth Kumar, R. and Aditya Kumar Sinha, 2017. "Parallelization of Backpropagation Algorithm and Benchmarking", *International Journal of Current Research*, 9, (06), 51818-51822.

INTRODUCTION

Machine learning is the domain in computer science that is used to design complex algorithms and models. It gives the chance to the computers to learn without programming explicitly. Applications for machine learning include bioinformatics, classifying DNA sequences, computer vision, game playing, internet fraud detection, marketing, economics, natural language processing, robotlocomotion, speech and handwriting recognition. Artificial Neural Network is the type of algorithm in machine learning. It is a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs (Khatri *et al*., 2015). ANN is composed of multiple nodes. The neurons are connected by links and they interact with each other. The result is then passed to the other neurons. The output at each node is called its activation or node value. Each link is associated with weight (https://www.tutorialspoint.com/artificial_ intelligence/artificial_intelligence_neural_networks.htm). Back propagation is a method used with gradient descent. It calculates the gradient of a loss function with respect to all the weights in the network.

****Corresponding author: Ishan Borker,***
Department of Computer Science and Engineering, Veltech Dr. RR and Dr. SR University, Chennai, Tamil Nadu India.

The gradient is fed to the optimization method which updates the weights to minimize the loss function. Back propagation requires the desired output for each input value in order to calculate the loss function gradient (https://mattmazur.com/ 2015/03/17/a-step-by-step backpropagation-example/). Open MP is a technique used to write amulti-threaded application. It provides the platform independent set of compiler pragmas, directives, function calls and environment variables to use the parallelism (Yadav *et al*., 2010). Master thread spawns the number of threads as required (Schuessler and Loyola, 2011). We use Open MP for improving the back propagation algorithm to achieve better efficiency with an available number of threads. Also, the speedup of the program increases with the increase in the number of threads. Parallelism can be done in BP by node parallelism. Here weight updating takes place by evaluating each node in a single layer in parallel (Pethick *et al*., 2003).

MATERIALS AND METHODS

Figure 3 represents the proposed architecture of the paper. It consists of 5 sections:

1. Input the out brain click data set: It is the input dataset required for execution of BPA.

2. Serial computation of BP algorithm: Here program code in serial manner is being generated and put into execution.

3. Output in the form of updating the weights: It gives the desired result by updating the weights in the neural network.

4. Various other techniques like Profiling of the code using flat profile and call graph etc, Optimization of the code from o2 to o4 levels and Parallelization of the code using Open MP is being carried out.

5. Benchmarking: Here the testing of the parallel, as well as serial code, is done on different system architectures like Intel 64-bit i5 processor and Intel Xeon Phi processor with an increase in the number of threads. It is done to evaluate the performance with respect to CPU time, memory, I/O communication.

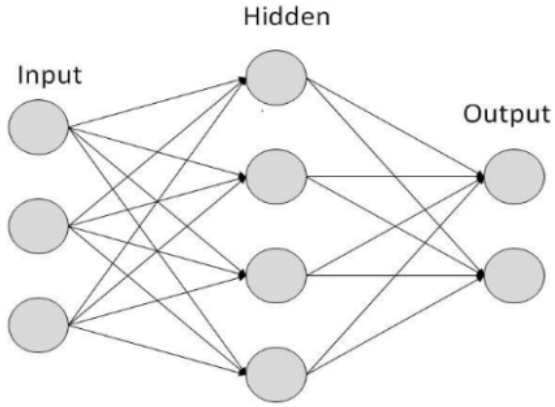

Figure 1. Simple ANN (athttps://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm)

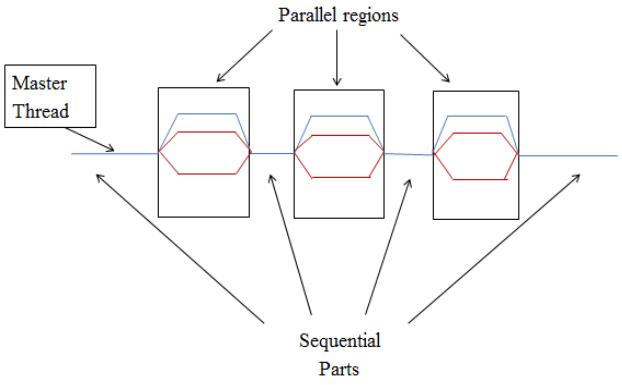

Figure 2. Fork-join model of Open MP

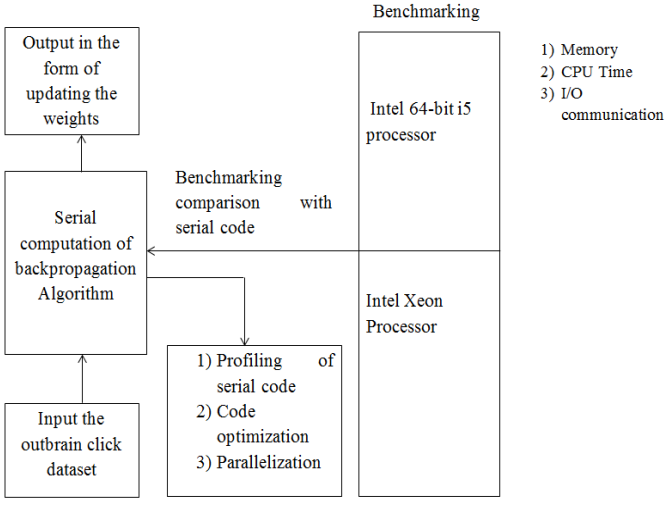

Figure 3. Architectural Diagram

Figure 4 represents the workflow of the paper. The stages performed are following: Input dataset of different sizes. Implement the Serial code of BP algorithm. Evaluate the code with respect to CPU time, memory usage, and I/O communication. Then perform code profiling using techniques like flat profile, call graph. Also, perform code optimization from o2 to o4 levels. Finally perform the benchmarking on the64-bit i5 processor, Intel Xeon Phi processor.

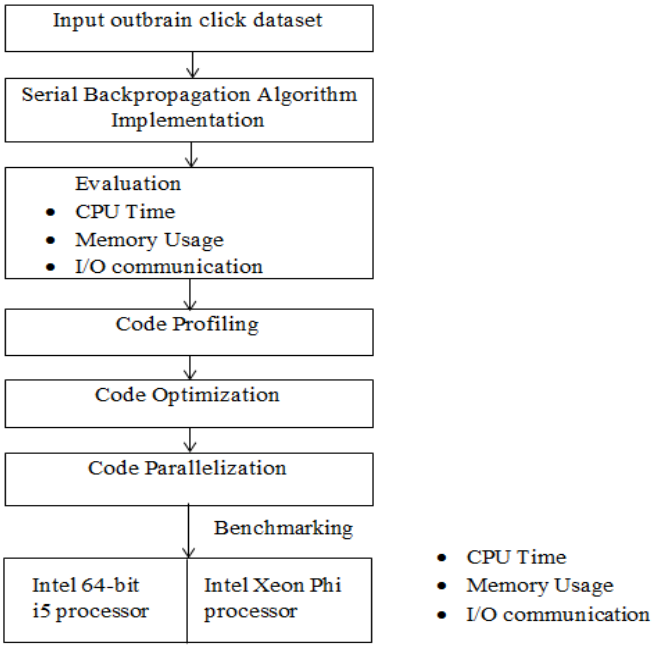

Figure 4. Workflow

Implementation

The out brain click dataset has been used as input for the BP algorithm. The dataset is taken for different sizes (24MB, 96MB, 250MB, 500MB, 750MB and 1GB).The attributes of the dataset are display_id, clicked, input, input weight, and output weight. Apply BP algorithm on this dataset. The result can be calculated based on updating the weights. Then we calculate the time of execution of serial code, memory usage and I/O communication. After this task, we perform profiling and optimization of the code. After this we use Open MP for parallelizing the code and then calculate the time of execution of serial code, memory usage and I/O communication and compare it with that to the serial code. Finally benchmarking of both the serial and parallel code is done on Intel 64-bit processor and Intel Xeon processor.

RESULTS ANALYSIS

Table I. Comparison of serial and parallel code with respect to CPU Time (in seconds) (Intel 64-bit processor)

Size of Dataset	Serial code	Parallel code	
		4 threads	8 threads
24MB	20.07	19.37	19.36
96MB	81.54	77.73	76.83
250MB	219.50	212.09	210.44
500MB	442.81	404.22	400.3
750MB	642.76	572.63	567.70
1GB	887.84	801.43	801.66

Table II. Comparison of serial and parallel code with respect to Memory usage (in bytes) (Intel 64-bit processor)

Size of Dataset	Serial code	Parallel code	
		4 threads	8 threads
24MB	12191	12122	12232
96MB	12324	12384	12430
250MB	7045	6146	6187
500MB	6985	6224	6279
750MB	9495	9521	9565
1GB	9678	9716	9757

III. Serial Code

Table III and IV. Comparison of serial and parallel code with respect to I/O communication (Intel 64-bit processor)

Size of Dataset	Average CPU (%)				
	% user	% nice	% system	% iowait	% Idle
24MB	19.59	0.05	11.91	1.23	67.22
96MB	19.38	0.05	11.75	1.20	67.62
250MB	19.45	0.08	11.03	1.89	67.55
500MB	18.32	0.09	10.72	2.06	68.81
750MB	21.2	0.06	12.09	1.56	65.09
1GB	19.68	0.05	11.96	1.23	67.08

Size of dataset	Average CPU (%)									
	% user		% nice		% system		% Iowait		% Idle	
	4 threads	8 threads	4 threads	8 threads	4 threads	8 threads	4 threads	8 threads	4 threads	8 threads
24MB	19.52	19.47	0.05	0.05	11.87	11.86	1.2	1.2	67.34	67.4
96MB	19.36	19.33	0.05	0.05	11.74	11.72	1.2	1.2	67.66	67.7
250MB	17.41	17.31	0.11	0.11	10.03	9.96	2.45	2.43	70	70.19
500MB	18.36	18.44	0.09	0.09	10.77	10.84	2.07	2.08	68.71	68.55
750MB	17.61	17.8	0.04	0.04	10.74	10.94	1.12	1.09	70.48	70.12
1GB	19.86	19.92	0.05	0.05	12.08	12.11	1.24	1.24	66.77	66.68

IV. Parallel Code

Table V. Comparison of serial and parallel code with respect to CPU Time (in seconds) (Intel Xeon processor)

Size of Dataset	Serial code	Parallel code	
		4 threads	8 threads
24MB	21.93	19.12	18.92
96MB	86.54	77.28	78.44
250MB	223.57	215.09	213.37
500MB	442.4	424.42	428.28
750MB	633.77	605.47	610.95
1GB	914.83	851.4	854.12

Table VI. Comparison of serial and parallel code with respect to Memory usage (in bytes) (Intel Xeon processor)

Size of Dataset	Serial code	Parallel code	
		4 threads	8 threads
24MB	9329	14986	15073
96MB	9409	6419	6834
250MB	9463	7348	7695
500MB	9519	8170	8751
750MB	9567	8850	8917
1GB	9630	8979	9049

VII. Serial Code

Table VII and VIII. Comparison of serial and parallel code with respect to I/O communication (Intel Xeon processor)

Size of dataset	Average CPU (%)				
	% user	% nice	% system	% iowait	% idle
24MB	0.91	0	0.76	0.17	98.15
96MB	0.92	0	0.77	0.17	98.14
250MB	5.52	0.03	6.5	4.55	83.40
500MB	5.71	0.01	7.21	1.84	85.22
750MB	6.09	0.01	7.95	1.08	84.88
1GB	6.67	0.01	8.2	0.58	84.55

VIII. Parallel Code

Size of dataset	Average CPU (%)									
	% user		% nice		% system		% iowait		% Idle	
	4 threads	8 threads	4 threads	8 threads	4 threads	8 threads	4 threads	8 threads	4 threads	8 threads
24MB	3.2	3.18	0.04	0.04	1.53	1.65	3.69	3.27	91.54	91.85
96MB	3.54	3.65	0.03	0.03	2.66	2.97	2.39	2.09	91.39	91.26
250MB	3.92	4.24	0.02	0.02	3.59	4.24	1.61	1.41	90.87	90.09
500MB	4.84	5.29	0.02	0.02	5.28	6.04	1.2	1.11	88.66	87.55
750MB	5.21	5.63	0.01	0.01	5.85	6.57	0.83	0.77	88.10	87.01
1GB	6.38	6.06	0.01	0.01	7.91	7.33	0.65	0.71	85.05	85.89

VIII. Parallel Code

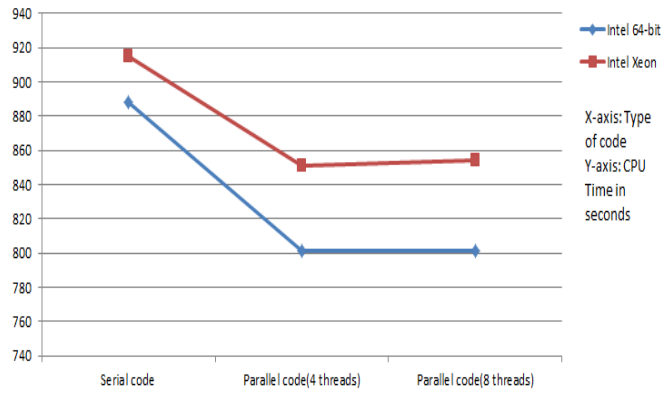

Figure 5. Graph representing the comparison of serial and parallel code with respect to CPU Time (in seconds) on 1GB dataset

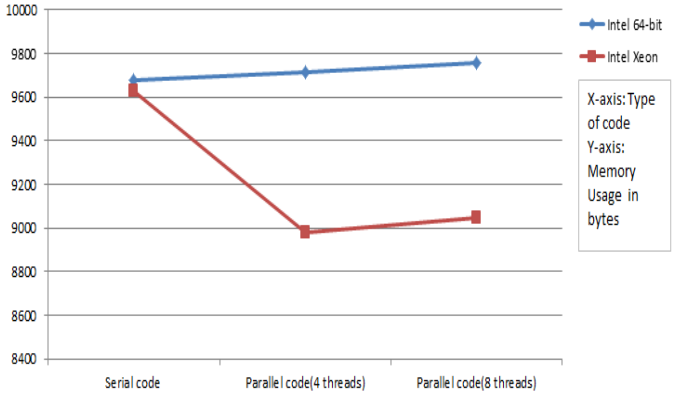

Figure 6. Graph representing the comparison of serial and parallel code with respect to Memory usage (in bytes) on 1GB dataset

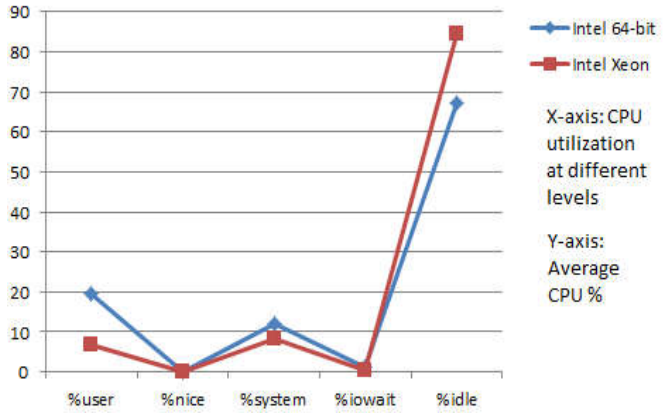

Figure 7. Graph representing the serial code with respect to I/O communication (in average CPU %) on 1GB dataset

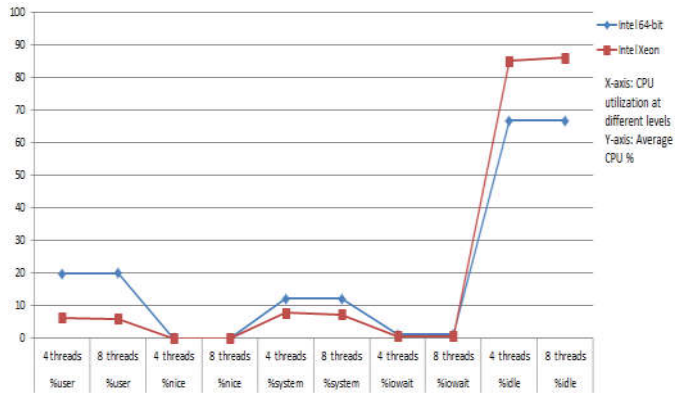

Figure 8. Graph representing the parallel code with respect to I/O communication (in average CPU %) on 1GB dataset

Conclusion

Various algorithms on machine learning were studied. Most efficient was back propagation algorithm. The serial code of this algorithm was implemented on different datasets of different sizes and executed on Intel 64-bit processor. We evaluate the performance with respect to CPU time, memory usage, and I/O communication. After this, code profiling and code optimization were carried out. Then we execute parallel code using OpenMP for all the datasets and evaluate the performance with respect to the above-mentioned parameters. Then we perform benchmarking of both the codes on Intel Xeon processor. It is observed that the time for execution, memory consumed and I/O communication in serial time is more than compared to the parallel code. Also, these factors get reduced on increasing the number of threads. In addition, Intel Xeon allows smooth execution of both the serial as well as parallel code as compared to that on Intel 64-bit processor.

Acknowledgment

I would like to thank Mr. Vinoth Kumar R. of Veltech University Chennai for his valuable suggestions and Mr. Aditya Kumar Sinha and Ms. Ruchika from C-DAC Pune for providing me infrastructure and motivation during the coursework. This work is done as part of M. Tech dissertation work.

REFERENCES

Jiang, P. Chen, C. and Liu, X. 2016. "Time series prediction for evolutions of complex systems: A deep learning approach," 2016 IEEE International Conference on Control and Robotics Engineering (ICCRE), Singapore, pp. 1-6.

Khatri, S. K., Dutta, S. and Johri, P. 2015. "Recognizing images of handwritten digits using learning vector

quantization artificial neural network," 2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)**,** Noida, pp. 1-4.

Shrestha, S., Bochenek, Z. and Smith, C. "Artificial Neural Network (ANN) beyond cots remote sensing packages: Implementation of Extreme Learning Machine (ELM) in MATLAB**,"** 2012 IEEE International Geoscience and Remote Sensing Symposium**,** Munich, 2012, pp. 6301-6304.

Chen, X. and Long, S. 2009. "Adaptive Multi-versioning for OpenMP Parallelization via Machine Learning," 2009 15th International Conference on Parallel and Distributed Systems**,** Shenzhen, pp. 907-912.

Yadav, R., Singh, M. D. and Mahajan, N. 2010. ″Parallelism through dynamic instrumentation at runtime," 2010 Second International Conference on Machine Learning and Computing, Bangalore, pp. 321-325.

****