2009 First International Conference on Computational Intelligence, Communication Systems and Networks

Reducing Feed-Forward Neural Network Processing Time Utilizing Matrix Multiplication Algorithms on Heterogeneous Distributed Systems

Ali Kattan, Rosni Abdullah, Rosalina Abdul Salam
School of Computer Science, Universiti Sains Malaysia
1800 USM, Penang, Malaysia
admin@alikattan.com, rosni@cs.usm.my, rosalina@cs.usm.my

Abstract

This paper presents a work in progress that aims to reduce the overall training and processing time of feed-forward multi-layer neural networks. If the network is large processing is expensive in terms of both; time and space. In this paper, we suggest a cost-effective and presumably a faster processing technique by utilizing a heterogeneous distributed system composed of a set of commodity computers connected by a local area network. Neural network computations can be viewed as a set of matrix multiplication processes. These can be adapted to utilize the existing matrix multiplication algorithms tailored for such systems. With Java technology as an implementation means, we discuss the different factors that should be considered in order to achieve this goal highlighting some issues that might affect such a proposed implementation.

1. Introduction

Artificial Neural Networks (ANN) implementation techniques are usually based on different factors including cost, hardware availability, accuracy, processing speed and the intended overall performance. There is always a trade-off between the accuracy of the implementation and the reliability of its performance [1]. In order to facilitate the process of rapidly modifying the network's parameters and learning methods, it is always preferable to have soft network architecture in the early stages of application development even if the target implementation is special hardware. Software simulation on conventional computers is usually used as a comparison performance measure against other ANN implementation approaches [2].

The inherent parallelism of ANNs could lead to huge amount of computations if these are carried out sequentially where such computations involve many floating point operations. Processing time is directly proportional to the size of a given ANN and the nature of conducted operations. For real-time processing tasks, the hardware implementation may often be the only viable solution [3]. Still, the choice of off-board or on-board training process affects the size of hardware used and level of parallelism exploited by the hardware [4]. Off-board training is conducted externally using software before committing to silicon.

The intended ANN parallelization can be achieved either by network-partitioning, pattern-partitioning, or combination of these two [5]. In the first technique, nodes and weights of the ANN are partitioned among different processors. In the second technique, pattern-partitioning, the training set is distributed over the processors while keeping a complete copy of the whole network in each processor node [6]. Parallel implementations on homogeneous PC clusters are common utilizing C/C++ [7], [8].

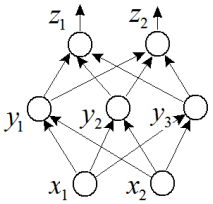
The remainder of this paper is organized as follows: Section 2 discusses ANN computations and matrix multiplication. Section 3 discusses matrix multiplication on heterogeneous processing nodes. Section 4 discusses the proposed ANN implementation on heterogeneous systems. Section 5 includes some of the early simulation results. And finally the conclusions are in section 6.

2. ANN Computations and Matrix Multiplication

Both ANN feed forward phase and the backpropagation phase calculations can be expressed in terms of matrix operations where matrix multiplication is very frequent [9]. Matrix multiplication is one of the computationally expensive operations since it occurs for each neuron processing. Many matrix multiplication parallel processing

978-0-7695-3743-6/09 $25.00 © 2009 IEEE
DOI 10.1109/CICSYN.2009.67

techniques have been already introduced. Such techniques can be categorized as either homogeneous or heterogeneous based the nature of its underling processing nodes. All these techniques aim to render the multiplication of dense matrices possible by taking care of memory considerations and to trying to reduce the conventional matrix multiplication $O(N^3)$ operations [7], [10], [11].



$$(1)\quad \begin{bmatrix} \bar{y}_1 \\ \bar{y}_2 \\ \bar{y}_3 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & b_1 \\ w_{21} & w_{22} & b_2 \\ w_{31} & w_{32} & b_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ -1 \end{bmatrix} \qquad (2)\quad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} f(\bar{y}_1) \\ f(\bar{y}_2) \\ f(\bar{y}_3) \end{bmatrix}$$

$$(3)\quad \begin{bmatrix} \bar{z}_1 \\ \bar{z}_2 \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} & v_{13} & a_1 \\ v_{21} & v_{22} & v_{23} & a_2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ -1 \end{bmatrix} \qquad (4)\quad \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} f(\bar{z}_1) \\ f(\bar{z}_2) \end{bmatrix}$$

Figure 1. Feed-forward ANN with forward-pass matrix calculations.

Parallel and distributed matrix multiplication approach depends on partitioning the resulting matrix C of C=AB into N sub-matrices. These partitions are referred to as blocks. Then each group of sub-matrix blocks are to be computed on one node. For homogenous environment such blocks are equal in size, while for a heterogeneous environment size is proportional to the computing power of the target node.

3. Matrix Multiplication on Heterogeneous Processing Nodes

A heterogeneous system is a collection of machines of varying architectures, and operating environments connected via network with different speed links [12]. Besides lower cost, the availability of such systems is high when compared to homogeneous multicomputer systems. However, load-balancing is main concern when considering such systems since each processing node should receive an amount of work in accordance to its computing power.

Optimal matrix partitioning problem on p heterogeneous processors is typically reduced to the geometrical problem of partitioning a unit square into rectangles [11]. This problem has proved to be NP-complete for the most general case [13]. However, many heuristics and schemes have been introduced to offer a solution that is more suitable in terms of the constraints of the underlying heterogeneous system [14]. A more convenient system in terms of implementation uses master-worker architecture [10]. The master is responsible for distributing the tasks to workers as well as receiving back the results. Matrix partitioning of the latter is depicted in Figure 2 where the atomic elements to be manipulated are not matrix coefficients but instead square blocks of size qxq.

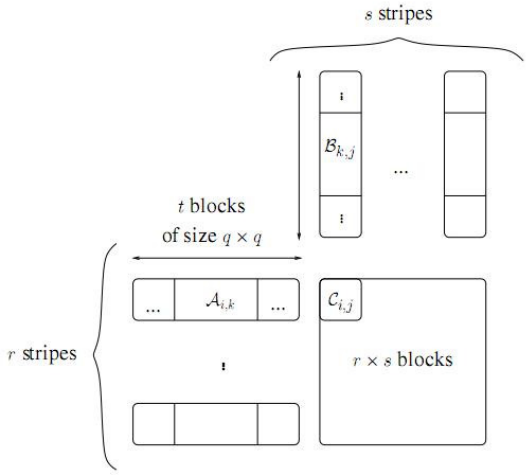


Figure 2. Partitioning of the three matrices A, B, and C to target a star network S = {P0, P1, P2, ... , Pp} of processors [10].

4. ANN Proposed Implementation on Heterogeneous Systems

The basic idea behind this work is to look at the ANN computation as a set of matrix multiplication processes in order to adapt the aforementioned parallel and distributed matrix multiplication techniques. Java technology would render such implementation possible. Due to its rich and powerful set of features, Java is considered a suitable high performance development language for developing distributed applications on heterogeneous systems [15]. The most important feature of Java is its platform-independent paradigm making it possible to distribute code with a consistent public API, while keeping the implementation details private [12], [16].

We intend to use the network-partitioning technique described earlier, and a master-slave application hierarchy similar to that described in [10]. The ANN forward phase computations consist mainly of the multiplication of the input vector by the weights matrix. We can then group a number of input vectors to

create an input matrix representing a set of inputs or a batch-input. This process renders as a matrix-matrix multiplication operation. Figure 3 shows part (1) of Figure 1 modified for an example of batch input of three.

$$\begin{bmatrix} \bar{y}_{11} & \bar{y}_{12} & \bar{y}_{13} \\ \bar{y}_{21} & \bar{y}_{22} & \bar{y}_{23} \\ \bar{y}_{31} & \bar{y}_{32} & \bar{y}_{33} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & b_1 \\ w_{21} & w_{22} & b_2 \\ w_{31} & w_{32} & b_3 \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ -1 & -1 & -1 \end{bmatrix}$$

Figure 3. Feed-forward ANN batch input of three patterns.

The same is true for the back-propagation training phase if we are to consider batch-training instead of online training. The process also renders as matrix-matrix multiplication operation. Batch-training was proved to be inefficient for a single computer implementation [17]. However it was proved that the training completion time for a cluster implementation is nonlinear when compared to a single computer implementation [7]. In addition, speed-up has been obtained with multiplicative batch-update step [18].

Proper heterogeneous node benchmarking is essential for an effective load-balancing otherwise network-traffic could yield a low computation-to-communication ratio [7]. Our initial tests showed that traditional, Java-ported, benchmarking techniques [19] and suite-based techniques [20], are unsuitable for this kind of distributed implementation. Running the same benchmark several times on the same system gave quite different results. We believe that such benchmarks are not suitable for our intended computation [21]. In addition, many challenges exist in benchmarking Java software [22] that are overlooked in these "ported" versions.

5. Early Simulation Results

A simulation prototype has been developed in Java to run on a single computer. This prototype models the ANN representation using matrix objects. These objects facilitate data distribution, processing and manipulation in blocks.

The 60,000 handwritten digit of the MNIST database [23] were used to train the ANN but with bipolar normalized values instead of gray scale levels. The ANN is composed of 784, 30 and 10 units representing input, hidden, and output units respectively. This sums to around 24,000 weight connections. Values of the training parameters, like the learning rate and the momentum, were fixed during training for the sake of comparison.

Using the same training set file, the ANN was first trained using the standard online and then using the conventional batch method with fixed batch step size. Batch proved to be inefficient regardless of the batch step size used and as reported earlier.

Batch training was used again but with pre-arranged training set file. The training set was arranged as a sequence of random chunks each having the same digit such that chunk size coincides with the intended batch step size. For instance if the batch step size is 300 then chunks of 300 similar digits are arranged randomly in the training set file (e.g. 300 digits of 7's followed by 300 digits of 0's and so on). The conventional matrix multiplication method was used in processing. Figure 4 shows the results of the online, conventional batch, and four different training cycles using batch with a pre-arranged/chunked training set of 75, 150, 300 and 675.

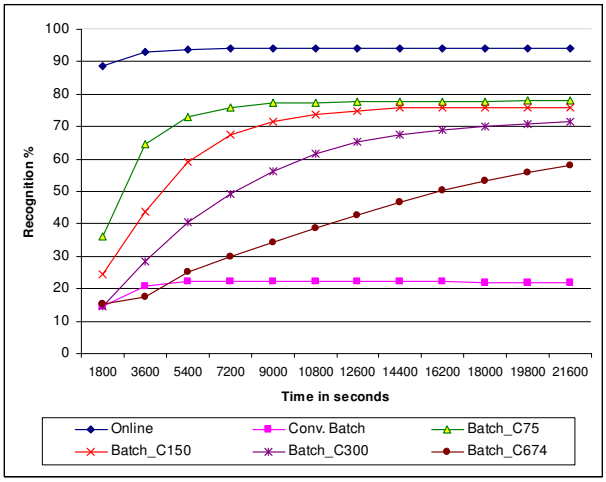


Figure 4. Recognition percents for different training methods of a 784-30-10 ANN.

Tuning some of the training parameters showed better recognition results for our pre-arranged/chunked batch training method.

The training cycles were halted manually after a certain period of time since reaching a higher recognition percent would take a considerable amount of time using this single computer simulation.

6. Conclusions

Though online training on a single computer is still more efficient than batch training methods, our early simulation results showed a significant improvement using batch method with pre-arranged training set. However, it is clear that the larger the batch step size used in this method the slower is the convergence.

If the suggested faster matrix multiplication method is used utilizing heterogeneous distributed and parallel processing environment then we anticipate that convergence time will be much faster.

The training parameters and the batch weight-update method need to be investigated further to achieve better results.

Aiming for the intended heterogeneous distributed and parallel implementation we intend to study the relation between different ANN factors like size and nature of the dataset used, the effect of matrix block-size to number of heterogeneous processing nodes, and the heuristic used for load-balancing between different processors. We infer that difference between the highest and the lowest benchmarked nodes would have an effect on the overall performance due to load-balancing. These issues are currently being investigated and a prototype heterogeneous distributed and parallel implementation model is under development.

7. References

[1] P. Moerland, E. Fiesler, and R. Beale, "Neural Network Adaptations To Hardware Implementations," in *Handbook of Neural Computation*: Oxford University Press, 1997, pp. E1.2:1-13.

[2] R. W. Duren, R. J. Marks II, and P. D. Reynolds, "Real-Time Neural Network Inversion on the SRC-6e Reconfigurable Computer," *IEEE TRANSACTIONS ON NEURAL NETWORKS,* vol. 18, pp. 889-901, 2007.

[3] V. Bochev, "Distributed Arithmetic Implementation of Artificial Neural Networks," *IEEE Transactions on Signal Processing,* vol. 41, pp. 2010-2013, 1993.

[4] K. Mathia and J. Clark, "On neural network hardware and programming paradigms," in *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02)*. vol. 3 Hawaii, USA, 2002, pp. 2692-2697.

[5] A. T. Chronopoulos and J. Sarangapani, "A distributed discrete-time neural network architecture for pattern allocation and control," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)*, Florida, USA, 2002, pp. 204-211.

[6] G. Dahl, A. McAvinney, and T. Newhall, "Parallelizing Neural Network Training for Cluster Systems," in *The IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN 2008)*, Innsbruck, Austria, 2008.

[7] C.-C. Jeng and I.-C. Yang, "Practical Implementation of Back-Propagation Networks in a Low-Cost PC Cluster," *Neural Information Processing,* vol. 4, pp. 33-37, 2004.

[8] A. Novokhodko and S. Valentine, "A Parallel Implementation of the Batch Backpropagation Training of Neural Networks," in *International Joint Conference on Neural Networks (IJCNN'01)*, Washington, DC, 2001, pp. 1783 - 1786.

[9] M. I. Soliman and S. A. Mohamed, "A highly efficient implementation of a backpropagation learning algorithm using matrix ISA," *Journal of Parallel and Distributed Computing,* vol. 68, pp. 949–961, 2008.

[10] J. Dongarra, J.-F. Pineau, Y. Robert, and F. Vivien, "Matrix product on heterogeneous master-worker platforms," in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Salt Lake City, UT, USA, 2008, pp. 53-62.

[11] A. Lastovetsky, "On Grid-based Matrix Partitioning for Heterogeneous Processors," in *Sixth International Symposium on Parallel and Distributed Computing (ISPDC'07)* Hagenberg, Austria, 2007, p. 51.

[12] J. Al-Jaroodi, N. Mohamed, H. Jiang, and D. Swanson, "Modeling parallel applications performance on heterogeneous systems," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, Nice, France, 2003.

[13] O. Beaumont and V. Boudet, "Matrix Multiplication on Heterogeneous Platforms," *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS,* vol. 12, pp. 1033-1051, 2001.

[14] S. Ohshima, K. Kise, T. Katagiri, and T. Yuba, "Parallel Processing of Matrix Multiplication in a CPU and GPU Heterogeneous Environment," in *7th International Meeting on High Performance Computing for Computational Science (VECPAR'06)* Reio de Janeiro, Brazil, 2006, pp. 305-318.

[15] J. M. Pérez, L. M. Sanchez, F. García, A. Calderón, and J. Carretero, "High performance Java input/output for heterogeneous distributed computing," in *Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC 2005)*, Cartagena, Spain, 2005, pp. 969-974.

[16] S. Matsuoka and S. Itou, "Towards Performance Evaluation of High-Performance Computing on Multiple Java Platforms," *Future Generation Computer Systems,* vol. 18, pp. 281-291, October 2001.

[17] D. Randall Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural Networks,* vol. 16, pp. 1429-1451, 2003.

[18] P. Cruz, "Speeding Up Backpropagation with Multiplicative Batch Update Step," in *Adaptive and Natural Computing Algorithms*. vol. Part I Vienna: Springer 2005, pp. 22-24.

[19] J. J. Dongarra, R. Wade, and P. McMahan, "Linpack Benchmark -- Java Version. Retrieved from: http://netlib.org/benchmark/linpackjava/

[20] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, B. Moss, A. Phansalkar, D. Stefanovic, T. VanDrunen, D. v. Dincklage, and B. Wiedermann, "The DaCapo benchmarks: java benchmarking development and analysis," in *Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (OOPSLA'06)*, Portland, Oregon, USA, 2006, pp. 169-190.

[21] R. M. Yoo, H.-H. S. Lee, H. Lee, and K. Chow, "Hierarchical Means: Single Number Benchmarking with Workload Cluster Analysis," in *IEEE 10th International Symposium on Workload Characterization (IISWC 2007)*, Boston, MA, USA, 2007, pp. 204 - 213.

[22] B. Boyer, "Robust Java benchmarking," IBM, developerWorks, 2008. Retrieved from: http://www.ibm.com/developerworks/java/library/j-benchmark1.html

[23] Y. LeCun and C. Cortes, "The MNIST Database of Handwritten Digits.", 1998. Retrieved from: http://yann.lecun.com/exdb/mnist/