Neural PCA and Maximum Likelihood Hebbian Learning on the GPU

Pavel Krömer[1,2], Emilio Corchado[2,3], Václav Snášel[1,2], Jan Platoš[1,2], and Laura García-Hernández[4]

[1] Department of Computer Science, VŠB-Technical University of Ostrava, 17.listopadu 15/2172, 708 33 Ostrava-Poruba, Czech Republic
[2] IT4Innovations, 17.listopadu 15/2172, 708 33 Ostrava-Poruba, Czech Republic
{pavel.kromer,vaclav.snasel,jan.platos}@vsb.cz
[3] Departamento de Informática y Automática, Universidad de Salamanca, Spain
escorchado@usal.es
[4] Area of Project Engineering, University of Cordoba, Spain
ir1gahel@uco.es

Abstract. This study introduces a novel fine-grained parallel implementation of a neural principal component analysis (neural PCA) variant and the maximum Likelihood Hebbian Learning (MLHL) network designed for modern many-core graphics processing units (GPUs). The parallel implementation as well as the computational experiments conducted in order to evaluate the speedup achieved by the GPU are presented and discussed. The evaluation was done on a well-known artificial data set, the 2D bars data set.

Keywords: neural PCA, Maximum Likelihood Hebbian Learning, Exploratory Projection Pursuit, GPU, CUDA, performance.

1 Introduction

Modern many-core GPUs have been successfully used to accelerate a variety of meta-heuristics and bio-inspired algorithms [6,12,13] including different types of artificial neural networks [1,10,11,14,15,17,18,20,22,24]. To fully utilize the parallel hardware, the algorithms have to be carefully adapted to data-parallel architecture of the GPUs [21].

Artificial neural networks (ANNs) performing PCA and MLHL are known to be useful for the analysis of high dimensional data [5,25]. Their main aim is to identify interesting projections of high dimensional data to lower dimensional subspaces that reveal hidden structure of the data sets. Due to the relative simplicity of their operations and generally real-valued data structures, such a networks are suitable for a parallel implementation on multi-core systems and on the GPUs that reach peak performance of hundreds and thousands giga FLOPS (floating-point operations per second) at low costs.

This study presents a design and evaluation of a novel fine-grained data-parallel implementation of an ANN for PCA and MLHL for the nVidia Compute Unified Device Architecture (CUDA) platform.

1.1 Neural PCA and MLHL

PCA is a standard statistical technique for compressing data; it can be shown to give the best linear compression of the data in terms of least mean square error. There are several ANNs which have been shown to perform PCA e.g. [9,19].

The Negative Feedback Network [9] for the PCA is defined as follows. Consider an N-dimensional input vector x and an M-dimensional output vector y with W_{ij} being the weight linking input j to output i and let η be the learning rate. The initial situation is that there is no activation at all in the network. The input data is feedforward via the weights from the input neurons (the x-values) to the output neurons (the y-values) where a linear summation is performed to get the output neuron activation value. It can be expressed as:

$$y_i = \sum_{j=1}^{N} W_{ij} x_j, \quad \forall i \tag{1}$$

The activation is fed back through the same weights and subtracted from the inputs (where the inhibition takes place):

$$e_j = x_j - \sum_{i=1}^{M} W_{ij} y_i, \quad \forall j \tag{2}$$

After that, simple Hebbian learning is performed between input and outputs:

$$\Delta W_{ij} = \eta e_j y_i \tag{3}$$

The effect of the negative feedback is the network learning stability. This network is capable of finding the principal components of the input data [9] in a manner that is equivalent to Oja's Subspace algorithm [19], and so the weights will not find the current Principal Components but a basis of the Subspace spanned by these components.

Maximum Likelihood Hebbian Learning [2,3,4,8] is based on the previous PCA-type rule and can be described as a family of learning rules based on the following equations: a feedforward step (1) followed by a feedback step (2) and then a weight change, which is as follows:

$$\Delta W_{ij} = \eta y_i \text{sign}(e_j) |e_j|^{p-1} \tag{4}$$

Maximum Likelihood Hebbian Learning (MLHL) [2,3,4,8] has been linked to the standard statistical method of Exploratory Projection Pursuit (EPP)[4,7].

2 GPU Computing

Modern graphics hardware has gained an important role in the area of parallel computing. The data-parallel architecture of the GPUs is suitable for vector and matrix algebra operations and it is nowadays widely used for scientific computing. The GPUs and general purpose GPU (GPGPU) programming have established a new platform for neural computation. The usage of the GPUs to accelerate neural information processing and artificial neural networks pre-dates the inception of general purpose GPU APIs [1,11,17,18]. At that time, the data structures were mapped directly to native GPU concepts such as textures and the operations were implemented using vertex and pixel shaders of the GPUs. Often, the ANNs were implemented using graphic oriented shader programs, OpenGL functions, or DirectX functions to accelerate ANN operations. For example, a 20 times accelerated feedforward network on the GPU was presented by Oh and Jung in [18]. Martínez-Zarzuela et al. [17] proposed a 33 times faster GPU-based fuzzy ART network. Ho et al. [11] developed a simulator of cellular neural network on the GPU that was 8 to 17 times faster than a corresponding CPU version, and Brandstetter and Alessandro [1] designed a 3 to 72-fold faster radial basis function network powered by the GPUs.

The GPGPU APIs have simplified the development of neural algorithms and ANNs for the graphics hardware significantly [10,16] and a variety of neurocomputing algorithms were ported to the GPUs [10,14,15,16,18,20,22,24]. The CUDA platform was used to achieve 46 to 63 times faster learning of a feedforward ANN by the backpropagation algorithm by Sierra-Canto et al. [24] while Lopes and Ribeiro [14] reported a 10 to 40 faster implementation of the multiple backpropagation training of feedforward and multiple feedforward ANNs.

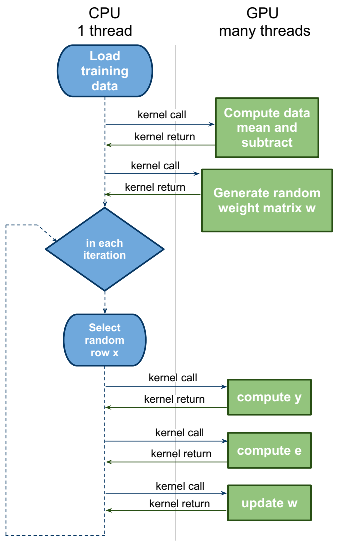
Ghuzva et al. [10] presented a coarse-grained implementation of the multilayer perceptron (MLP) on the CUDA platform that operated a set of MLPs in parallel 50 times faster than a sequential CPU-based implementation. The training of a feedforward neural network by genetic algorithms was implemented on CUDA by Patulea et al. [20] and it was 10 times faster than a sequential version of the same algorithm. An application of a GPU-powered ANN for speech recognition is due to Scanzio et al. [22]. The GPU technology accelerated the ANN approximately 6 times. Martínez-Zarzuela et al. [15] used the GPU to speedup a neural texture classification process and achieved 16 to 26 times better performance than on the CPU. In [16], the authors implemented a fuzzy ART network on the CUDA platform and achieved a 57-fold peak speedup.

An example of the use of GPUs for unsupervised neural networks is due to Shitara et al. [23]. Three different graphic cards were used to benchmark the performance of the algorithm and it was shown that the GPUs can improve the performance of the SOM up to 150 times for certain hardware configurations.

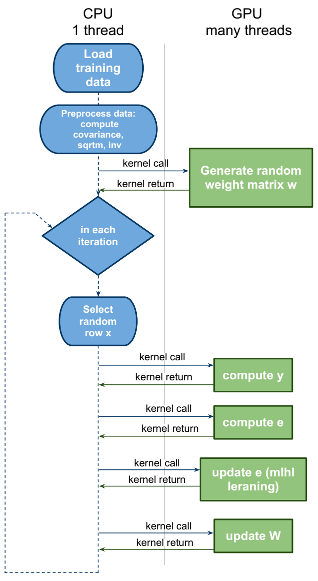
In this research, the CUDA platform is used to accelerate the training of the Negative Feedback Network and also for MLHL.

3 A Version of Neural PCA and MLHL on CUDA

According to the authors' knowledge, there is no prior research on the acceleration of the training phase of ANNs for PCA and MLHL by the GPUs. However, Oh and Jung [18] have combined the PCA and an ANN with CUDA accelerated feedforward pass for a system for view-point tolerant human pose recognition.



(a) Neural PCA Flowchart



(b) MLHL Flowchart

Fig. 1. Neural PCA and MLHL on CUDA

The CUDA implementation of the Negative Feedback Network and MLHL is outlined in fig. 1a and fig. 1b respectively. The GPU was used to accelerate the iterative phase of the algorithm (i.e. (1) - (4)). The implementation used the *cublas* library, a set of custom kernels that implemented operations not available in *cublas* such as the *sign* function, and auxiliary kernels for common operations such as generation of batches of random numbers. All operations of the iterative phase of network training were implemented on CUDA to minimize memory transfers between the host and the device and maximize the performance of the implementation.

3.1 Experiments and Results

To evaluate the performance of the Negative Feedback Network and MLHL on CUDA, the fine-grained parallel implementations were compared to sequential single-threaded CPU implementations of the same algorithms. Both networks were implemented from scratch in C/C++ and CUDA-C and their execution times for the same data set were compared. The experiments were performed on a server with 2 dual core AMD Opteron processors at 2.6GHz and an nVidia Tesla C2050 device with 448 cores at 1.15GHz. The server was running Linux operating system and CUDA SDK 4.0 was used.

To obtain a randomized high-dimensional data set with clear internal structure and simple interpretation, two variants of the 2D bars data set were generated. The first one contained 10000 records with 256 attributes and the second

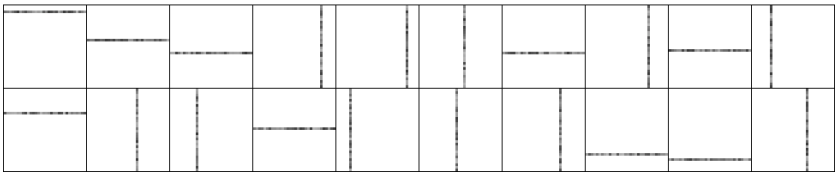


Fig. 2. First 20 records of the 1024-dimensional data set as 32×32 images

one contained 10000 records with 1024 attributes. Each record in the data set can be seen as an $n \times n$ image with a single vertical or horizontal bar painted by different shades of gray (represented by real values between 0.7 and 1). The visualisation of the first 20 records of the 1024-dimensional data set is shown in fig. 2. It can be expected, that in such a data set, the pictures with the bar in the same position ought to form (at least one) cluster, i.e. there might be at least $n + n$ clusters. The randomized data sets used in this study contained 15 and 31 unique bar positions respectively.

The data sets were processed by both, the Negative Feedback Network and MLHL on CPU and GPU with the following parameters: 100000 iterations, learning rate 0.00001 and the MLHL parameter $p = 2.2$.

In the experiment, the dimension of the target subspace m was set to the powers of 2 from the interval $[2, DIM]$ (where DIM was the full dimension of the data set) and the execution time of network training was measured. The results are visualized in fig. 3. It clearly illustrates how the execution time grows with the dimension of the target subspace m and with the number of attributes. These two parameters define the complexity of the vector-matrix operations. As expected, the CPU is faster for small m ($m < 32$ for 256-dimensional data and $m < 16$ for 1024-dimensional data) for the Negative Feedback Network. The MLHL on the GPU was faster than the CPU-based implementation of the same algorithm even for small values of m. The speedup obtained by the parallel implementation for the 256-dimensional data set ranged from 1.4 for $m = 32$ to 5.5 for $m = 256$ for the Negative Feedback Network and from 1.5 to 6.1 for the MLHL. The performance increase was more significant for the 1024-dimensional data set. The improvement in the training time of the Negative Feedback Network on the GPU ranged from 1.36 times faster training for $m = 16$ to 47.95 faster training for $m = 512$. The processing of the 1024-dimensional data set by the MLHL on the GPU was between 2.18 to 47.81 times faster than on the CPU.

The performance results of both algorithms for the 1024-dimensional data set on different hardware are visualized in fig. 3. It displays the dependency of the execution time (y-axis, note the log scale) on the dimension of the target subspace m and illustrates how the GPU versions of the algorithms outperform the CPU versions by an order of magnitude for larger m.

The visual results of the projection of the 1024-dimensional data set to the 2-dimensional subspace are for both methods shown in fig. 4. Figure 4a shows the results of the projection by the neural PCA and fig. 4b shows the structure of the same data processed by the MLHL. Points representing images that had a bar in the same position were drawn in the same color. We can clearly see that both

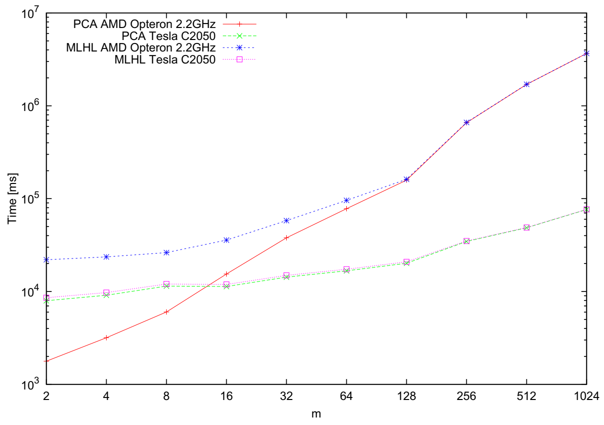


Fig. 3. Neural PCA (Negative Feedback Network) and MLHL execution time for the 1024-dimensional data set

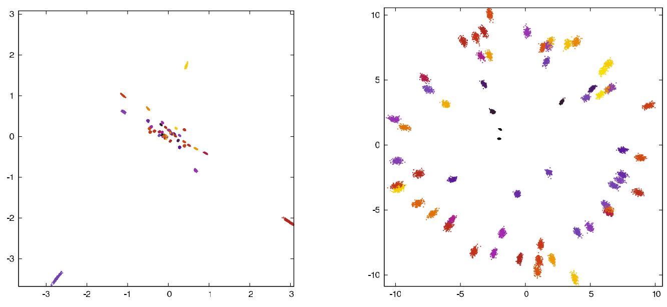


Fig. 4. The results of projection to 2D for the 1024-dimensional data set

projections have emphasized a structure in the data. The neural PCA version clearly separated several clusters from the rest of the data, which populates the center of the graph, while the MLHL lead to a more regular pattern of 2D clusters. This can serve as a visual proof that the CUDA-C implementations of both algorithms provide projections to lower dimensional subspaces with good structure.

4 Conclusions

This research introduced a fine-grained data-parallel implementation of two types of ANNs, the Negative Feedback network for the PCA and the Maximum Likelihood Hebbian Learning network. The GPU versions of the algorithms have achieved for two high-dimensional artificial data sets a significant speedup in training times. When projecting to low dimensional subspaces ($m < 16$), the CPU version of the negative feedback network was faster but when projecting the data to spaces with larger dimension, the GPU was up to 47.99 times faster (for the 1024-dimensional data set and $m = 1024$). The projection through the MLHL network was on the GPU faster for all $m \in [2, DIM]$ ranging from 2.1-fold speedup for $m = 8$ to 47.81 times faster execution time for $m = 1024$.

In the future, other variants of the MLHL will be implemented and the GPU version will be used to process and analyze real world data sets.

Acknowledgements. This research is partially supported through a projects of the Spanish Ministry of Economy and Competitiveness [ref: TIN2010-21272-C02-01] (funded by the European Regional Development Fund). This work was also supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070) and by the Bio-Inspired Methods: research, development and knowledge transfer project, reg. no. CZ.1.07 /2.3.00/20.0073 funded by Operational Programme Education for Competitiveness, co-financed by ESF and state budget of the Czech Republic.

References

1. Brandstetter, A., Artusi, A.: Radial basis function networks gpu-based implementation. IEEE Transactions on Neural Networks 19(12), 2150–2154 (2008)
2. Corchado, E., Fyfe, C.: Orientation selection using maximum likelihood hebbian learning. Int. Journal of Knowledge-Based Intelligent Engineering 2(7) (2003)
3. Corchado, E., Han, Y., Fyfe, C.: Structuring global responses of local filters using lateral connections. J. Exp. Theor. Artif. Intell. 15(4), 473–487 (2003)
4. Corchado, E., MacDonald, D., Fyfe, C.: Maximum and minimum likelihood hebbian learning for exploratory projection pursuit. Data Mining and Knowledge Discovery 8, 203–225 (2004)
5. Corchado, E., Perez, J.C.: A three-step unsupervised neural model for visualizing high complex dimensional spectroscopic data sets. Pattern Anal. Appl. 14(2), 207–218 (2011)
6. De, P., Veronese, L., Krohling, R.A.: Swarm's flight: accelerating the particles using c-cuda. In: Proceedings of the Eleventh conference on Congress on Evolutionary Computation, CEC 2009, pp. 3264–3270. IEEE Press, Piscataway (2009)
7. Friedman, J., Tukey, J.: A projection pursuit algorithm for exploratory data analysis. IEEE Transactions on Computers C- 23(9), 881–890 (1974)
8. Fyfe, C., Corchado, E.: Maximum likelihood Hebbian rules. In: Verleysen, M. (ed.) ESANN 2002, Proceedings of the 10th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 24-26, pp. 143–148 (2002)
9. Fyfe, C.: A neural network for pca and beyond. Neur. Proc. Letters 6, 33–41 (1997)
10. Guzhva, A., Dolenko, S., Persiantsev, I.: Multifold Acceleration of Neural Network Computations Using GPU. In: Alippi, C., Polycarpou, M., Panayiotou, C., Ellinas, G. (eds.) ICANN 2009, Part I. LNCS, vol. 5768, pp. 373–380. Springer, Heidelberg (2009)
11. Ho, T.Y., Lam, P.M., Leung, C.S.: Parallelization of cellular neural networks on gpu. Pattern Recogn. 41(8), 2684–2692 (2008)
12. Krömer, P., Platoš, J., Snášel, V., Abraham, A.: An Implementation of Differential Evolution for Independent Tasks Scheduling on GPU. In: Corchado, E., Kurzyński, M., Woźniak, M. (eds.) HAIS 2011, Part I. LNCS, vol. 6678, pp. 372–379. Springer, Heidelberg (2011)

13. Langdon, W.B., Banzhaf, W.: A SIMD Interpreter for Genetic Programming on GPU Graphics Cards. In: O'Neill, M., Vanneschi, L., Gustafson, S., Esparcia Alcázar, A.I., De Falco, I., Della Cioppa, A., Tarantino, E. (eds.) EuroGP 2008. LNCS, vol. 4971, pp. 73–85. Springer, Heidelberg (2008)
14. Lopes, N., Ribeiro, B.: GPU Implementation of the Multiple Back-Propagation Algorithm. In: Corchado, E., Yin, H. (eds.) IDEAL 2009. LNCS, vol. 5788, pp. 449–456. Springer, Heidelberg (2009)
15. Martínez-Zarzuela, M., Díaz-Pernas, F., Antón-Rodríguez, M., Díez-Higuera, J., González-Ortega, D., Boto-Giralda, D., López-González, F., De La Torre, I.: Multiscale neural texture classification using the gpu as a stream processing engine. Machine Vision and Applications 22, 947–966 (2011)
16. Martínez-Zarzuela, M., Pernas, F., de Pablos, A., Rodríguez, M., Higuera, J., Giralda, D., Ortega, D.: Adaptative Resonance Theory Fuzzy Networks Parallel Computation Using CUDA. In: Cabestany, J., Sandoval, F., Prieto, A., Corchado, J.M. (eds.) IWANN 2009, Part I. LNCS, vol. 5517, pp. 149–156. Springer, Heidelberg (2009)
17. Martínez-Zarzuela, M., Díaz Pernas, F., Díez Higuera, J., Rodríguez, M.: Fuzzy ART Neural Network Parallel Computing on the GPU. In: Sandoval, F., Prieto, A.G., Cabestany, J., Graña, M. (eds.) IWANN 2007. LNCS, vol. 4507, pp. 463–470. Springer, Heidelberg (2007)
18. Oh, K.S., Jung, K.: GPU implementation of neural networks. Pattern Recognition 37(6), 1311–1314 (2004)
19. Oja, E.: Neural networks, principal components, and subspaces. International Journal of Neural Systms 1(1), 61–68 (1989)
20. Patulea, C., Peace, R., Green, J.: Cuda-accelerated genetic feedforward-ann training for data mining. J. of Physics: Conference Series 256(1), 012014 (2010)
21. Sanders, J., Kandrot, E.: CUDA by Example: An Introduction to General-Purpose GPU Programming, 1st edn. Addison-Wesley Professional (July 2010)
22. Scanzio, S., Cumani, S., Gemello, R., Mana, F., Laface, P.: Parallel implementation of artificial neural network training for speech recognition. Pattern Recognition Letters 31(11), 1302–1309 (2010)
23. Shitara, A., Nishikawa, Y., Yoshimi, M., Amano, H.: Implementation and evaluation of self-organizing map algorithm on a graphic processor. In: Parallel and Distributed Computing and Systems 2009 (2009)
24. Sierra-Canto, X., Madera-Ramirez, F., Uc-Cetina, V.: Parallel training of a back-propagation neural network using cuda. In: Proceedings of the 2010 Ninth International Conference on Machine Learning and Applications, ICMLA 2010, pp. 307–312. IEEE Computer Society, Washington, DC (2010)
25. Zhang, K., Li, Y., Scarf, P., Ball, A.: Feature selection for high-dimensional machinery fault diagnosis data using multiple models and radial basis function networks. Neurocomputing 74(17), 2941–2952 (2011)