GPUMLib: A New Library to Combine Machine Learning Algorithms with Graphics Processing Units

Noel Lopes[†*], Bernardete Ribeiro[†] and Ricardo Quintas[†]
[†] Dep of Informatics Engineering, Center for Informatics and Systems of University of Coimbra (CISUC), Portugal
[*]UDI/IPG - Research Unit, Polytechnic Institute of Guarda, Portugal
noel@ipg.pt, bribeiro@dei.uc.pt, rquintas@student.dei.uc.pt

***Abstract*—The Graphics Processing Unit (GPU) is a highly parallel, many-core device with enormous computational power, especially well-suited to address Machine Learning (ML) problems that can be expressed as data-parallel computations. As problems become increasingly demanding, parallel implementations of ML algorithms become critical for developing hybrid intelligent real-world applications. The relative low cost of GPUs combined with the unprecedent computational power they offer, make them particularly well-positioned to automatically analyze and capture relevant information from large amounts of data. In this paper, we propose the creation of an open source GPU Machine Learning Library (GPUMLib) that aims to provide the building blocks for the scientific community to develop GPU ML algorithms. Experimental results on benchmark datasets demonstrate that the GPUMLib components already implemented achieve significant savings over the counterpart CPU implementations.**

***Keywords*-GPU computing; machine learning algorithms**

I. Introduction

The phenomenal growth of the Internet combined with the emergence of a multitude of devices capable of gathering, storing and sharing information anytime, anywhere, resulted in an abundant wealth of data available to researchers. This brings the need for intelligent systems, and subsequently the issues posed by more challenging and demanding machine learning (ML) algorithms, often computationally expensive

The lack of openly available implementations is a serious obstacle to algorithm replication and application to new tasks and therefore poses a barrier to the progress of the ML field. Sonnenburg et al. argue that these problems could be significantly amended by incentivizing researchers to publish software under an open source model [1]. This model has many advantages that ultimately lead to: better reproducibility of experimental results and fair comparison of algorithms; quicker detection of errors; quicker adoption of algorithms; faster adoption of ML methods in other disciplines and in industry [1]. Sonnenburg et al. also support that software (and data) should be distributed under a suitable open source license along with scientific papers. They point out this is common practice in biomedical research, where protocols and biological samples are frequently made publicly available. Although nowadays there are plentiful excellent toolkits which provide support for developing ML software in several environments (e.g. Python, R, Lua, Matlab) [2], they fail to meet the expectations in terms of computational performance, in particular when dealing with many of today's real-world problems. The time required by ML algorithms, such as the well-known back-propagation can make their use impracticable. To cope with this issue it is necessary to shift towards the use of high-performance ML libraries. Graphics Processing Units (GPUs) represent a feasible solution to address above issues in this direction. GPUs are designed for high-performance rendering where repeated operations are common, therefore are much more effective in utilizing parallelism and pipelining than CPUs [3]. It is not uncommon to reach speedups of over 50× and there are cases where the computations can be reduced from weeks to hours [4]. Therefore, GPUs provide an attractive alternative to use dedicated hardware [5] by enabling high performance implementations of ML algorithms [6] in particular in cost expensive datasets. Moreover, GPUs are widely available and relatively inexpensive [5], [6]. However only recently General-Purpose computing on GPU (GPGPU) has become the scientific computing platform of choice, mainly due to the debut of NVIDIA CUDA (Compute Unified Device Architecture) platform [7], which allows programmers to use the industry-standard C language together with extensions to target a general purpose, massively parallel processor (GPU). Owens et al. provided a very exhaustive survey on GPGPU, identifying many of the algorithms, techniques and applications implemented on graphics hardware [8].

We conducted an in-depth analysis of several papers dealing with GPU ML implementations. To illustrate the overwhelming throughput of current research, we conceived Figure 1, containing the chronology of ML software GPU implementations, based on the data scrutiny from several papers [3], [5], [6], [9]–[28]. The number of GPU implementations of ML algorithms has increased substantially over the last few years. However, only a few of those were released under open source. Aside from our own implementation of the Multiple Back-Propagation software (available at http://dit.ipg.pt/MBP) [26] we were able to find only four more open source GPU implementations of ML algorithms. This is a major obstacle to the progress of the ML field, as it forces researchers, facing problems with high-computational requirements, to build from scratch GPU ML algorithms. Moreover being an excellent ML researcher does not necessary imply being an excellent programmer [1]. Thus many researchers may not have the skills or the time required to implement GPU ML algorithms. To alleviate this problem we propose the creation of a GPU ML

978-1-4244-7365-6/10/$26.00 ©2010 IEEE

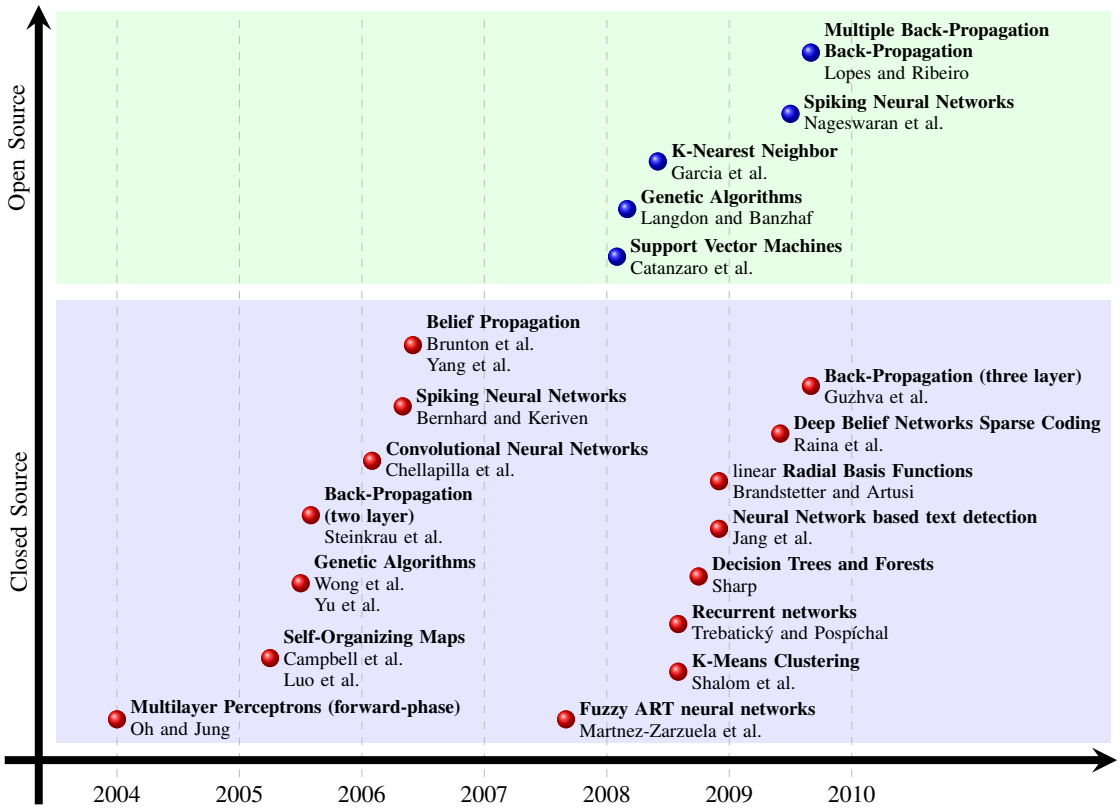

Fig. 1. Chronology of ML software GPU implementations.

library (GPUMLib) that we hope will concentrate the works of ML researchers and practitioners. Such a library would reduce the effort spent by ML people when implementing new algorithms on the GPU and contribute to the creation of innovative applications in the ML field.

The remainder of this paper is structured as follows. The next section details the proposal for a GPU ML library (GPUMLib). Section III reports the results obtained with the algorithms implemented on well-known benchmarks. Finally in Section IV we draw the conclusions and future work.

II. Proposal for a GPU Machine Learning Library

We aim at the creation of an open source high performance GPU ML library, by using the CUDA architecture. This architecture seems to be the logical choice to build such a library, because ever since its introduction, around three years ago, it has steadily become the scientific computing platform of choice [7]. Therefore, choosing CUDA will allow us to captivate other researchers and developers to help on this effort.

Our goal is not to replace existing ML libraries such as WEKA (http://www.cs.waikato.ac.nz/ml/weka/) and KEEL (http://www.keel.es/) or duplicate in any way the work that is already done, but rather to complement them. In this sense we envision the integration of GPUMLib in other ML libraries and we intend to provide the tools necessary for its integration with other ML software at a latter phase.

Currently the GPUMLib fully implements the Back-Propagation (BP), the Multiple Back-Propagation (MBP) and the Non-Negative Matrix Factorization (NMF) algorithms. Furthermore, an implementation of the Radial Basis Functions (RBF) neural networks was developed [29] (an incremental version is under development) and the implementation of other algorithms is planned. The library (see Figure 2) is released under the GNU General Public License and its source code, documentation and examples can be obtained at http://gpumlib.sourceforge.net/.

At the core of the library there is a set of CUDA kernels (special C functions that can be executed in parallel) which support the execution of ML algorithms on the GPU. For each specific algorithm, several kernels are required. However, some kernels may be shared by different algorithms.

Each ML algorithm will have its own C++ class, that is responsible for: transferring the information (inputs) needed by the algorithm to the device (GPU); calling the algorithm kernels in the proper order; and transferring the algorithm outputs and intermediate values back to the host (CPU). To facilitate (and standardize) the task of moving information from and to the GPU, GPUMLib also provides a memory access framework that accounts for information transfer between the host and device (and vice-versa) in an effortless and seemly manner. Furthermore the DeviceMatrix class also provides an easy way to multiply and obtain the transpose of device matrices. Moreover, matrices can be created using row-

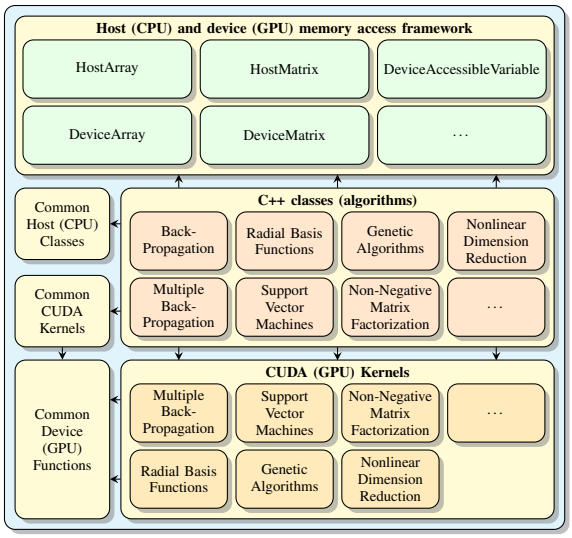

Fig. 2. Main components of the GPUMLib.

major or column-major order which is of major importance due to its impact on the kernels performance.

Finally, a note on the GPUMLib documentation is given, since good documentation plays a major role in the success of any software library. In order to ease its usage and development, GPUMLib provides extensive quality documentation and examples. Moreover the library is practical and easy to use and extend and does not require full understanding of the details inherent to GPU computing.

III. Experimental Setup and Results

To illustrate the GPUMlib, the experimental septup is described along with practical examples with datasets obtained at the UCI machine learning repository [30]. In addition, a more research-oriented problem with images of faces from the MIT Center for Biological and Computer Learning is presented.

Table I presents the speedups ($\times$) obtained by the GPUMLib implementations of the BP and MBP algorithms over the CPU implementations for the poker benchmark, using an 8600 GT device (32 cores) and a GTX 280 device (240 cores). This benchmark consists of classifying a "poker hand" based in the suit and rank of five cards. We divided the suit into four different input variables and the rank into 13 different variables (one for each rank). Thus, the training dataset contained 25010 samples with $5(13+4) = 85$ inputs and ten outputs.

Using the 8600 GT device we were able to train networks over $30\times$ faster than on a CPU, whilst using the GTX 280 we achieved speedups of more than $175\times$. Moreover, when using the MBP algorithm, the CPU required more than five minutes for training a network with 300 hidden neurons, during a single epoch. Using the GTX 280 device we could train almost 34 epochs per minute and using the 8600 GT we were able to train almost 11 epochs per minute. Figure 3 shows the number of epochs trained per minute.

To test and validate the implementations of the NMF algorithms, the face database #1 from the MIT Center for

TABLE I
BP and MBP GPU speedups ($\times$) in the Poker problem.

Algorithm	Hidden neurons	8600 GT	GTX 280
BP	12	8.35 ± 0.07	57.49 ± 0.35
	24	8.05 ± 2.15	54.79 ± 0.40
	60	8.73 ± 0.05	59.51 ± 0.44
	120	8.98 ± 0.08	57.55 ± 0.35
	180	17.80 ± 0.20	111.30 ± 0.36
	240	27.41 ± 0.50	159.03 ± 2.71
	300	29.03 ± 1.58	174.91 ± 9.50
MBP	12	8.61 ± 0.07	61.50 ± 0.44
	24	8.13 ± 0.05	58.42 ± 0.31
	60	8.68 ± 0.05	58.61 ± 0.33
	120	21.02 ± 0.16	132.67 ± 0.91
	180	27.85 ± 1.31	171.73 ± 8.44
	240	30.29 ± 0.22	174.45 ± 0.60
	300	30.12 ± 1.15	178.64 ± 6.86

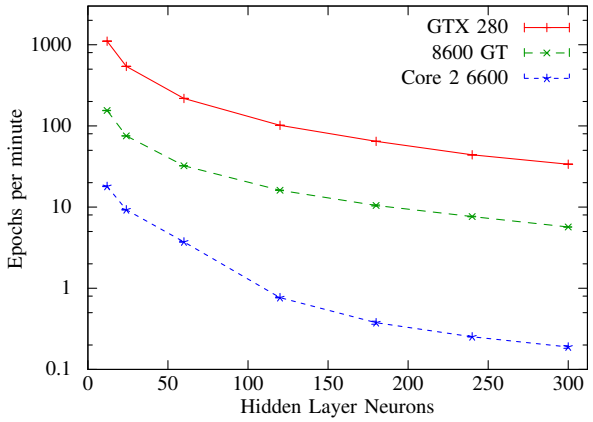

Fig. 3. Number of epochs with MBP for the poker problem.

Biological & Computational Learning (available at http://cbcl.mit.edu/cbcl/software-datasets/FaceData2.html) was used. This database includes a total of 2429 face images of $19 \times 19 = 361$ pixels. Thus, the objective is to decompose matrix $V \in \mathbb{R}_+^{361\times 2429}$ into $W \in \mathbb{R}_+^{361\times r}$ and $H \in \mathbb{R}_+^{r\times 2429}$, such that $V \approx WH$ [31]. Figure 4 presents the speedups provided by a GTX 280 GPU relatively to a Core 2 Quad 2.5 GHz CPU.

The RBF implementation consists of several tasks identified as center selection, parameter selection, and weight calculation. The testing setup for the GPU version consisted of a GeForce 9800 GT (112 cores), while the CPU version was tested on a Intel Core 2 Duo E8400 running at 3.0GHz. Table II presents the results evaluated using 10 fold cross-validation. As expected, the results show higher benefits for larger datasets, as in the case of the Satellite dataset depicting the greatest performance boost.

IV. Conclusion

In this paper we propose the development of a GPU high performance ML library (GPUMLib). One important compo-

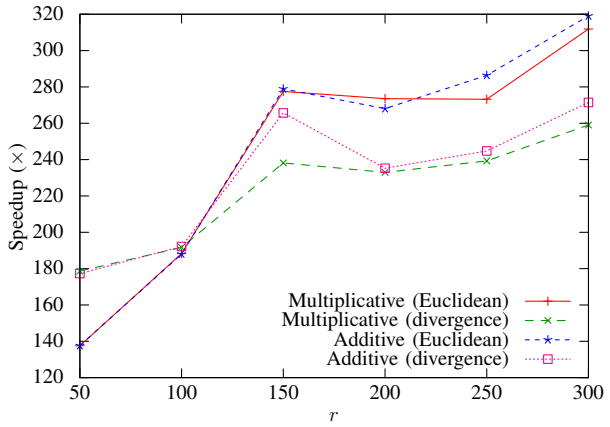

Fig. 4. GPU Speedups for the NMF algorithms.

TABLE II
COMPARISON BETWEEN CPU AND GPU EXECUTION TIMES IN RBF.

Dataset	Samples	Features	CPU(s)	GPU(s)	Speedup
Iris	150	4	4.58	12.36	0.37
Breast	569	31	66.99	28.54	2.35
Vehicle	846	18	452.97	346.55	1.31
Vowel	990	10	994.42	866.70	1.15
CMC	1473	9	638.05	501.78	1.27
Satellite	6458	36	10011.50	2365.66	4.23

nent of the approach relies on the building blocks necessary to create ML software, which can easily be selected by researchers and practitioners in the field minimizing their effort. It was tested with the BP, MBP, NMF and RBF algorithms providing remarkable speedups as compared with the CPU versions of the same algorithms. Finally, we hope to attract the efforts of other researchers on the endeavor of building this library. Future work will extend the GPUMLib to support vector machines, genetic algorithms, among other, opening up the development of hybrid intelligent systems.

ACKNOWLEDGMENTS

FCT (SFRH/BD/62479/2009) is gratefully acknowledged.

REFERENCES

[1] S. Sonnenburg, M. L. Braun, C. S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K.-R. Müller, F. Pereira, C. E. Rasmussen, G. Rätsch, B. Schölkopf, A. Smola, P. Vincent, J. Weston, and R. Williamson, "The need for open source software in machine learning," *Journal of Machine Learning Research*, vol. 8, pp. 2443–2466, 2007.

[2] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[3] H. Jang, A. Park, and K. Jung, "Neural network implementation using CUDA and OpenMP," in *Proc. of the 2008 Digital Image Computing: Techniques and Applications (DICTA '08)*, 2008, pp. 155–161.

[4] N. Lopes and B. Ribeiro, "Fast pattern classification of ventricular arrhythmias using graphics processing units," in *Proc. 14th Iberoamerican Conf on Pattern Recognition*. Springer, 2009, pp. 603–610.

[5] D. Steinkrau, P. Y. Simard, and I. Buck, "Using GPUs for machine learning algorithms," in *Proc. of the 8th Int Conf. on Document Analysis and Recognition*, vol. 2, 2005, pp. 1115–1120.

[6] B. Catanzaro, N. Sundaram, and K. Keutzer, "Fast support vector machine training and classification on graphics processors," in *Proc. of the 25th Int. Conf. on Machine Learning*, vol. 307, 2008, pp. 104–111.

[7] D. Schaa and D. Kaeli, "Exploring the multiple-GPU design space," in *Proc. of the 2009 IEEE Int Symposium on Parallel & Distributed Processing*, 2009, pp. 1–12.

[8] J. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. Lefohn, and T. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[9] K.-S.-S. Oh and K. Jung, "GPU implementation of neural networks," *Pattern Recognition*, vol. 37, no. 6, pp. 1311–1314, 2004.

[10] Z. Luo, H. Liu, and X.Wu, "Artificial neural network computation on graphic process unit," in *Proc. of the 2005 Int. Joint Conf. on Neural Networks*, vol. 1, 2005, pp. 622–626.

[11] A. Campbell, E. Berglund, and A. Streit, "Graphics hardware implementation of the parameter-less self-organising map," in *Proc. 2005 Intelligent Data Engineering and Automated Learning*. Springer, 2005, pp. 343–350.

[12] Q. Yu, C. Chen, and Z. Pan, "Parallel genetic algorithms on programmable graphics hardware," in *Proc. of the 1st Int. Conf. on Advances in Natural Computation*. Springer, 2005, pp. 1051–1059.

[13] M. Wong, T. Wong, and K. Fok, "Parallel evolutionary algorithms on graphics processing unit," in *Proc. of the 2005 IEEE Congress on Evolutionary Computation*, vol. 3, 2005, pp. 2286–2293.

[14] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," in *Proc. of the 10th Int. Workshop on Frontiers in Handwriting Recognition*, 2006.

[15] F. Bernhard and R. Keriven, "Spiking neurons on GPUs," in *Proc. 2006 Int. Conf. on Computational Science*. Springer, 2006, pp. 236–243.

[16] A. Brunton, C. Shu, and G. Roth, "Belief propagation on the GPU for stereo vision," in *Proc. 3rd Canadian Conf. on Computer and Robot Vision*. IEEE Computer Society, 2006, p. 76.

[17] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nistér, "Real-time global stereo matching using hierarchical belief propagation," in *Proc. of the 2006 British Machine Vision Conf.*, 2006, pp. 989–998.

[18] M. Zarzuela, F. Pernas, J. Higuera, and M. Rodríguez, "Fuzzy ART neural network parallel computing on the GPU," in *Proc. 9th Int. Work-Conf. on Artificial Neural Networks*. Springer, 2007, pp. 463–470.

[19] W. B. Langdon and W. Banzhaf, "A SIMD interpreter for genetic programming on GPU graphics cards," in *Proc. 11th European Conf. on Genetic Programming*. Springer, 2008, pp. 73–85.

[20] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," in *Proc. of the 2008 IEEE Computer Society Conf. on Computer Vision and Pattern Recognition Workshops*, 2008, pp. 1–6.

[21] S. A. Shalom, M. Dash, and M. Tue, "Efficient k-means clustering using accelerated graphics processors," in *Proc. 10th Int. Conf. on Data Warehousing and Knowledge Discovery*. Springer, 2008, pp. 166–175.

[22] P. Trebatický and J. Pospíchal, "Neural network training with extended kalman filter using graphics processing unit," in *Proc. of the 18th Int. Conf. on Artificial Neural Networks*. Springer, 2008, pp. 198–207.

[23] T. Sharp, "Implementing decision trees and forests on a GPU," in *Proc. 10th European Conf. on Computer Vision*. Springer, 2008, pp. 595–608.

[24] A. Brandstetter and A. Artusi, "Radial basis function networks GPU-based implementation," *IEEE Transactions on Neural Networks*, vol. 19, no. 12, pp. 2150–2154, 2008.

[25] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau, and A. V. Veidenbaum, "A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors," *Neural Networks*, vol. 22, no. 5-6, pp. 791–800, 2009.

[26] N. Lopes and B. Ribeiro, "GPU implementation of the multiple back-propagation algorithm," in *Proc. of the 2009 Intelligent Data Engineering and Automated Learning*. Springer, 2009, pp. 449–456.

[27] A. Guzhva, S. Dolenko, and I. Persiantsev, "Multifold acceleration of neural network computations using GPU," in *Proc. of the 19th Int. Conf. on Artificial Neural Networks*. Springer, 2009, pp. 373–380.

[28] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proc. of the 26th Annual Int. Conf. on Machine Learning*, vol. 382. ACM, 2009, pp. 873–880.

[29] R. Quintas, "GPU implementation of RBF networks in audio steganalysis," MSc Thesis, Univ. of Coimbra, July 2010.

[30] A. Asuncion and D. Newman, "UCI machine learning repository," 2007. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

[31] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *NIPS*. MIT Press, 2001, pp. 556–562.