International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 3 (2011) pp. 355-362

GPUMLib: An Efficient Open-Source GPU Machine Learning Library

Noel Lopes[1] **and Bernardete Ribeiro**[2]

[1]UDI, Polytechnic Institute of Guarda, Portugal
CISUC, University of Coimbra, Portugal
noel@ipg.pt

[2]Department of Informatics Engineering
CISUC, University of Coimbra, Portugal
bribeiro@dei.uc.pt

Abstract: **Graphics Processing Units (GPUs) placed at our disposal an unprecedented computational-power, largely surpassing the performance of cutting-edge CPUs (Central Processing Units). The high-parallelism inherent to the GPU makes this device especially well-suited to address Machine Learning (ML) problems with prohibitively computational intensive tasks. Nevertheless, few ML algorithms have been implemented on the GPU and most are not openly shared, posing difficulties for researchers and engineers aiming to develop GPU-based systems. To mitigate this problem, we propose the creation of an open source GPU Machine Learning Library (GPUMLib) that aims to provide the building blocks for the development of efficient GPU ML software. Experimental results on benchmark datasets show that the algorithms already implemented yield significant time savings over the CPU counterparts.**
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 on-growing collection of data being available at our disposal. However, availability does not necessarily imply usefulness and humans facing the innumerable requests, imposed by modern life, need help from intelligent systems to cope and take advantage of the high-volume of data generated and accumulated by our society. In particular, Machine Learning (ML) systems that can extract relevant and useful information from today's large repositories of data assume a major importance.

Although, at present there are plentiful excellent toolkits which provide support for developing ML software in several environments (e.g. Python, R, Lua, Matlab) [1], those fail to meet the expectations of researchers and engineers, in terms of computational performance, when dealing with many of today's real-world problems.

Typically, ML algorithms are computationally expensive and its complexity is often directly related with the amount of data to be processed. Rationally, as the volume of data associated with ML problems increases, the trend is to have more challenging and computationally demanding problems that can become intractable for traditional CPU (Central Processing Unit) architectures. Therefore, the pressure to shift development towards parallel architectures with high-throughput has been accentuated. In this context, the Graphics Processing Unit (GPU) represents a compelling solution to address the increasing needs of computational performance, in particular in the ML field.

In the last eight years the performance and capabilities of the GPUs have been significantly augmented and todays GPUs, included in mainstream computing systems, are powerful, highly parallel and programmable devices that can be used for general-purpose computing applications [2]. Since GPUs are designed for high-performance rendering where repeated operations are common, they are much more effective in utilizing parallelism and pipelining than CPUs [3]. Hence, they can provide remarkable performance gains for computationally-intensive applications involving data-parallelizable tasks. It is not uncommon, for GPU implementations to achieve significant time reductions as compared with CPU counterparts (e.g. weeks of processing on the CPU may be transformed into hours on the GPU [4]). Such characteristics did not go unnoticed, spawning interest in the scientific community who successfully mapped a broad range of computationally demanding problems to the GPU. As a result, the GPU represents a credible alternative to traditional microprocessors in the high-performance computer systems of the future [2].

To successfully take advantage of the GPU, applications and algorithms should present a high-degree of parallelism, large computational requirements and be related with data throughput rather than with the latency of individual operations [2]. Since most ML algorithms and techniques fall under these guidelines, GPUs provide an attractive alternative to the use of dedicated hardware [5] by enabling high-performance implementations of ML algorithms [6]. Furthermore, the GPU peak performance is growing at a much faster pace than the CPU performance [7] and since GPUs are

used in the large gaming industry, they are mass produced and regularly replaced by new generation with increasing computational power and additional levels of programmability. Consequently, unlike many earlier throughput-oriented architectures, they are widely available and relatively inexpensive [5, 6, 8].

Overtime a large body of powerful learning algorithms, suitable for a wide range applications, has been developed in the field of machine learning. Unfortunately, the true potential of these methods is not fully capitalized, since existing implementations are not openly shared, resulting in software with low usability and weak interoperability [9]. Moreover, 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 [9]. This model presents many advantages that ultimately lead to: better reproducibility of experimental results and fair comparison of algorithms; quicker detection of errors; quicker adoption of algorithms; innovative applications and easier combination of advances by allowing researchers and practitioners to build on top of existing resources (rather than re-implementing them); faster adoption of ML methods in other disciplines and in industry [9]. Recognizing the importance of publishing ML software under the open source model Sonnenburg et al. even propose a method for formal publication of ML software, similar to what the ACM Transactions on Mathematical Software provide for Numerical Analysis. They also argue that supporting software and data should be distributed under a suitable open source license along with scientific papers, pointing out that this is common practice in some biomedical research, where protocols and biological samples are frequently made publicly available [9].

The remainder of this paper is structured as follows. In the next section we summarize the open source and the proprietary GPU parallel implementations of ML algorithms and analyze the need for an open source GPU ML library. Section III details our proposal for a GPU ML library (GPUMLib). Section IV reports the results obtained with the algorithms implemented by GPUMLib on well-known benchmarks. Finally in Section V we draw the conclusions and future work.

II. A Review of GPU Parallel Implementations of Machine Learning Algorithms

Over the past few years, the GPU has evolved from a special-purpose processor for rendering graphics into a highly parallel programmable device that plays an increasing role in scientific computing applications [2]. The benefits of using GPUs for general purpose programming have been recognized for quite some time. Using GPUs for scientific computing allowed a wide range of challenging problems to be solved, providing the mechanisms for researchers to study larger datasets. However, only recently, General-Purpose computing on GPU (GPGPU) has become the scientific computing platform of choice, mainly due to the introduction of NVIDIA CUDA (Compute Unified Device Architecture) platform [10], 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 [11].

We conducted an in-depth analysis of several papers dealing with GPU ML implementations. To illustrate the overwhelming throughput of current research, we represent in Figure 1 the chronology of ML software GPU implementations, based on the data scrutiny from several papers [3, 5–7, 12–40]. 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 the GPU implementations included in GPUMLib we were able to find only four more open source GPU implementations of ML algorithms. This is an obstacle to the progress of the ML field, as it forces researchers and engineers, facing problems where the computational requirements are prohibitive, to build from scratch GPU ML algorithms that were not yet released under open source. Moreover, being an excellent ML researcher does not necessary imply being an excellent programmer [9]. Additionally, the GPU programming model is significantly different from the traditional models [2, 8] and to fully take advantage of this architecture one must first become versed on the specifics of this new programming paradigm. Thus, many researchers may not have the skills or the time required to implement GPU ML algorithms from scratch. To alleviate this problem we propose the creation of a GPU ML library (GPUMLib) that we hope will concentrate the works of ML researchers and practitioners [41]. Such a library would reduce the effort spent by researchers in the ML field while implementing new algorithms on the GPU. Also, it would contribute to the development of innovative applications in the area.

III. GPU Machine Learning Library Proposal

We aim at building 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, CUDA has steadily become the scientific computing platform of choice [10]. Therefore, choosing CUDA will allow us to attract other researchers and developers into helping on this effort.

A. A Brief Overview of CUDA

The CUDA architecture exposes the GPU as a massive-parallel device that operates as a co-processor to the host (CPU). A CUDA capable device (GPU) is composed of an array of Streaming Multiprocessors (SMs), which in turn contain a set of processing units (observe Figure 2). On newer devices (Fermi architecture), such as the GTX 480, each SM contains 32 scalar processor cores, while on older devices, such as the GTX 280, each SM possesses 8 cores.

To take advantage of this architecture, programs must break down data-parallel workload tasks into independent processing blocks. A processing block consists of multiple execution threads, up to a maximum of 512 (1024 in the case of

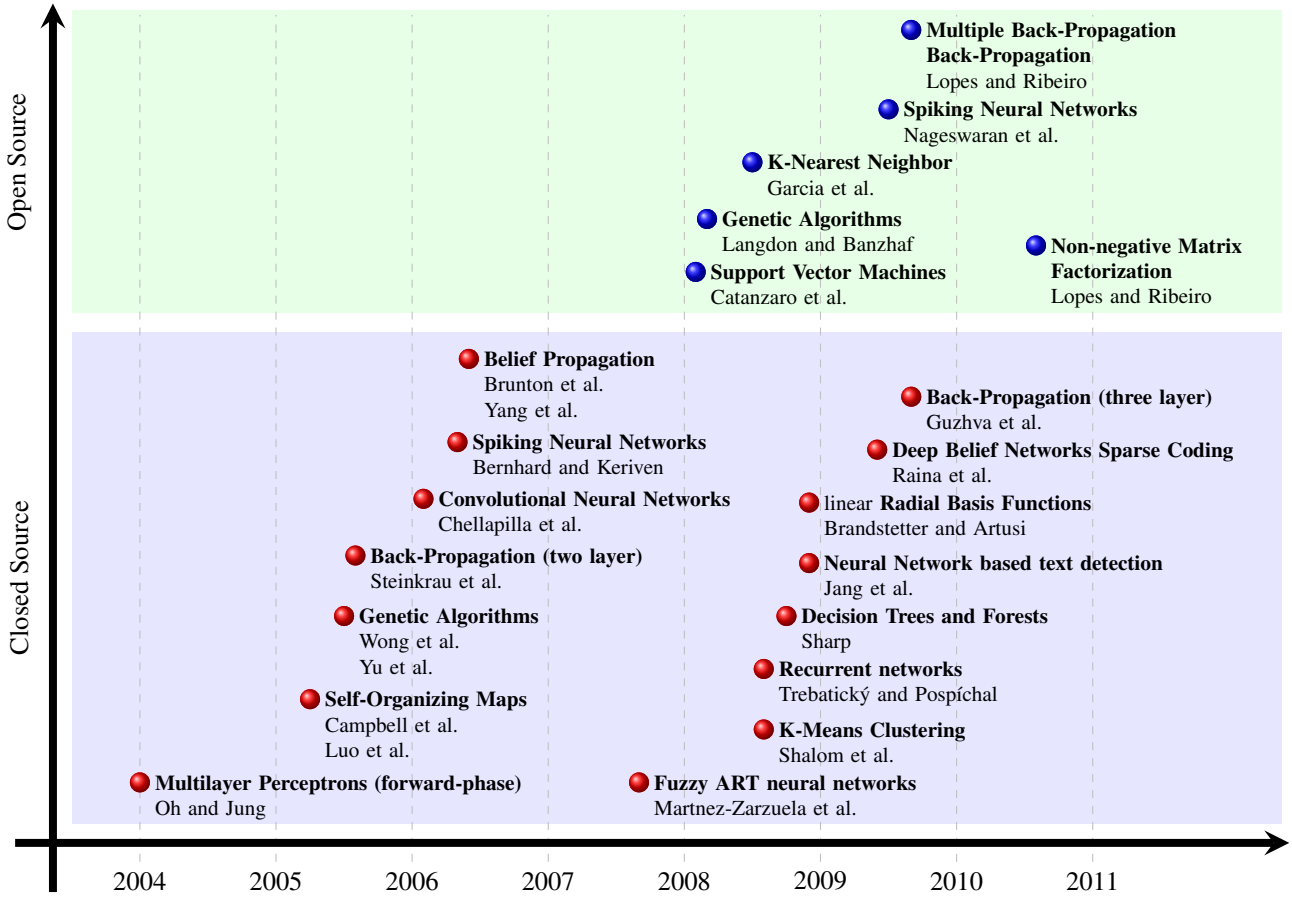

Figure. 1: Chronology of ML software GPU implementations.

the Fermi architecture). The set of thread blocks forms a grid. Figure 3 depicts this organization.

The number of blocks processed simultaneously depends on the number of SMs of the device and on the amount of resources (e.g. shared memory, registers) required by each block. Thus, the order in which the blocks are executed is arbitrary and blocks should not be depend on each other. However, within a block, threads can cooperate among themselves by sharing data (using the on-chip shared memory which is much faster than the global memory) and synchronizing their execution. This is possible, because each block runs entirety on a single SM.

When a block finishes its execution, new blocks are assigned to the vacated SMs. In this manner, the grid running can fully benefit of the GPU high-core count, scaling well on different devices (with a different number of SMs). The SM employs a new architecture called SIMT (single-instruction, multiple-thread). The multiprocessor SIMT unit creates, manages, schedules, and executes groups of 32 parallel threads called warps. Unlike the threads in the CPU, GPU threads are lightweight and to hide the global memory (off-chip) latencies a large number of threads (thousands or even millions, depending on the problem) is necessary [42]. Thus, a much finner granularity must be considered when defining the work to be carried out by each thread.

To specify the work of each thread, programmers must define special C/C++ functions, named kernels. Kernels, define the sequence of tasks to be carried out in each thread mapped over a domain (the set of threads to be invoked) [43]. To identify the actual thread location in the domain and allow each thread to work on separate parts of a dataset, kernels can access a set of intrinsic thread-identification variables (e.g. *threadIdx*, *blockIdx*, *blockDim* and *gridDim*). Figure 4 shows one of the kernels used in the GPUMLib implementation of the Non-Negative Matrix Factorization algorithm.

B. GPUMLib Architecture

GPUMLib does not aim to replace existing ML libraries such as WEKA (`http://www.cs.waikato.ac.nz/ml/weka/`) and KEEL (`http://www.keel.es/`) duplicating the work that has already been 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), Multiple Back-Propagation (MBP), Non-Negative Matrix Factorization (NMF) and Radial Basis Function (RBF) algorithms. Moreover, the implementation of other ML algorithms is already underway. The library is released under the GNU General Public License and its source code, documentation and examples can be obtained at `http://gpumlib.sourceforge.net/`. Figure 5 presents the main components of the library.

At the core of the library there is a set of CUDA kernels which support the execution of ML algorithms on the GPU. Usually, in order to implement a ML algorithm on the GPU,

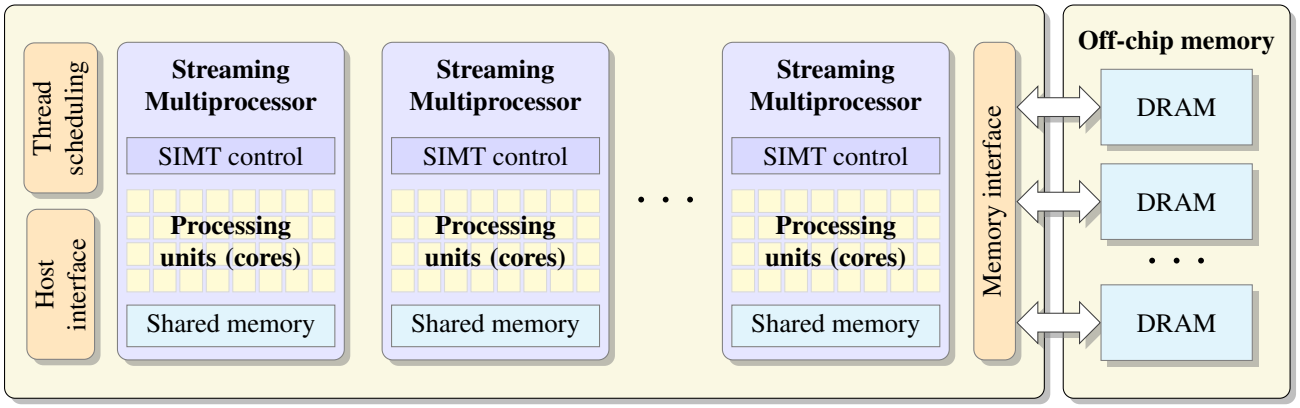

Figure. 2: An NVIDIA GPU (device).

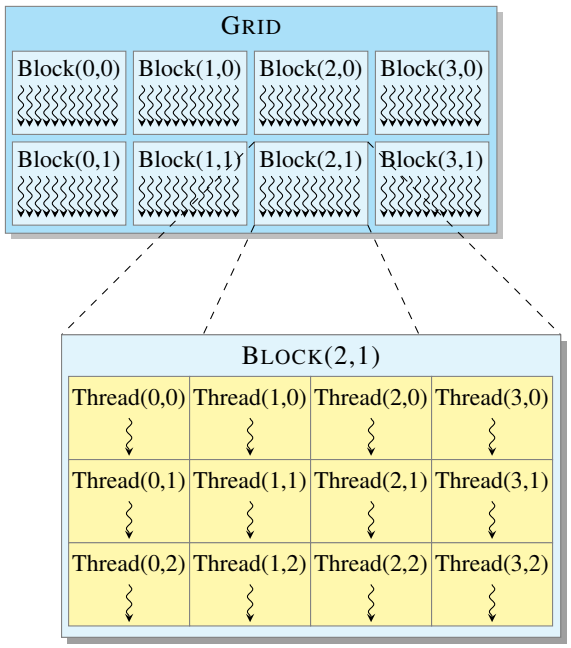

Figure. 3: Organization of threads, for execution on the GPU.

several kernels are required. However, the same kernel might be used to implement different algorithms. For example, the Back-Propagation and the Multiple Back-Propagation algorithms share the same kernels (FireLayer, FireOutputLayer, CalcLocalGradients, CorrectWeights, CalculateRMS and RobustLearning).

Each ML algorithm has 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. This model allows non-GPU developers to take advantage of GPUMLib, without requiring them to understand the specific details of CUDA programming.

GPUMLib provides a standard memory access framework to support the tasks of memory allocation and data transfer between the host and device (and vice-versa) in an effortless and seemly manner. Among the classes in this framework, a class is included to represent GPU matrices (DeviceMatrix), providing a straightforward and efficient way of performing

```
__global__ void
UpdateMatrix_ME (float * nm, float * dm, float * m, int elements) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < elements) m[idx] *= nm[idx] / (dm[idx] + SMALL_VALUE);
}
```

Figure. 4: A kernel used to implement the NMF algorithm. CUDA specific keywords appear in bold.

matrix computations (multiplication and transpose). Since the order in which the matrix elements are stored has an undeniable impact on the kernels performance, the DeviceMatrix class supports both row-major and column-major orders, fitting the needs of the users.

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, easy to use and extend and does not require full understanding of the details inherent to GPU computing.

IV. GPUMLib Application Results

To illustrate the expedience of GPUMlib, the experimental setup is described along with practical examples using datasets obtained at the UCI machine learning repository [44]. In addition, a more research-oriented problem with images of faces from the MIT Center for Biological and Computer Learning is presented.

Table 1 presents the speedups ($\times$) obtained by the GPUMLib implementations of the Back-Propagation and Multiple Back-Propagation algorithms over the CPU implementations for the poker benchmark. The tests were conducted using an Intel Core 2 Duo 6600 CPU running at 2.4 GHz and two GPUs: an NVIDIA 8600 GT device with 4 SMs ($4 \times 8 = 32$ cores) and an NVIDIA GTX 280 device with 30 SMs ($30 \times 8 = 240$ cores). The poker benchmark consists of classifying a "poker hand" based on the suit and rank of five cards. In the original dataset there were only two inputs per card (the suit and the rank). However, we divided the suit into four different input variables (one for each suit) and the rank into 13 different variables (one for each rank). The training dataset used in our benchmarks had 25010 samples each containing $5(13 + 4) = 85$ inputs and 10 outputs.

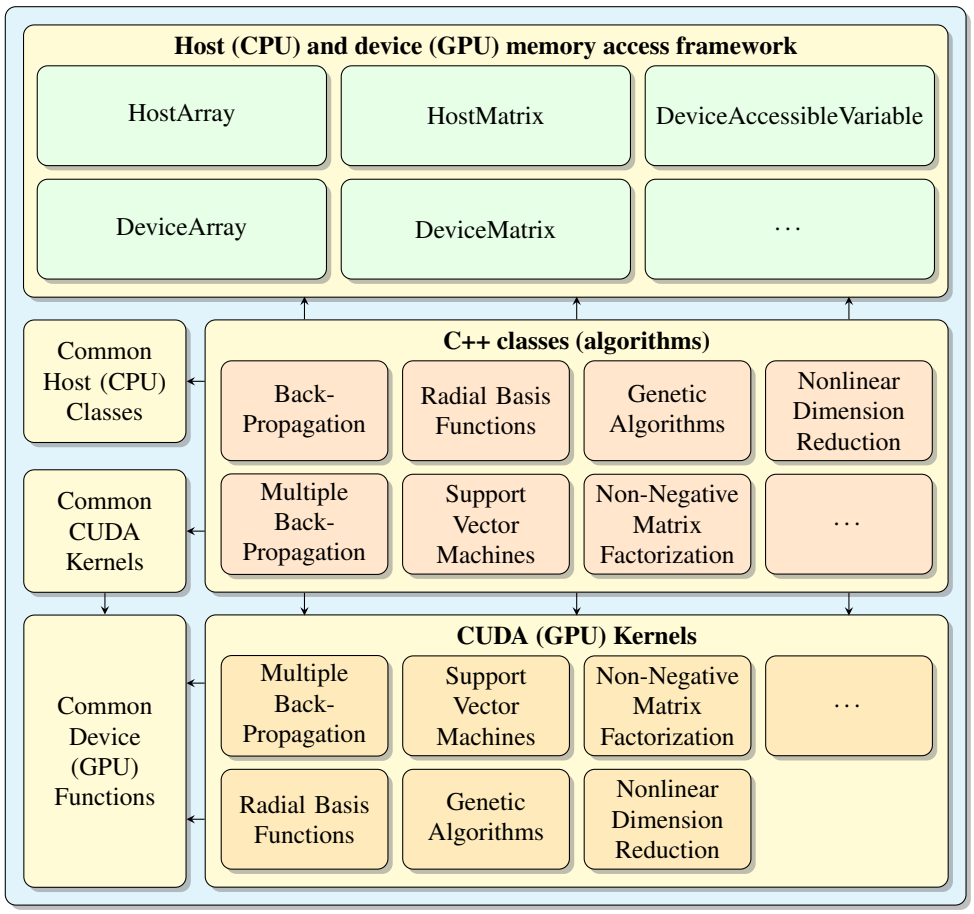

Figure. 5: Main components of the GPUMLib.

Using the 8600 GT device we were able to train networks over 30× faster than on a CPU, whilst using the GTX 280 we achieved speedups of more than 175×. 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 6 shows the number of epochs trained per minute according to the hardware used, for the poker problem, with the MBP algorithm.

To test and validate the GPUMLib implementations of the NMF algorithms, the face database #1 from the MIT Center for 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, in this case the objective of NMF 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 $V \approx WH$ [45]. Figure 7 presents the speedups provided by an NVIDIA GTX 280 GPU relatively to an Intel Core 2 Quad 2.5 GHz CPU.

The RBF implementation consists of several tasks, such as identified as center selection, parameter selection, and weight calculation. To evaluate its performance an audio steganalysis problem is presented [46]. The goal is to detect hidden (covered) information (steganography) in audio files. A database with 8780 samples, each containing 58 features was

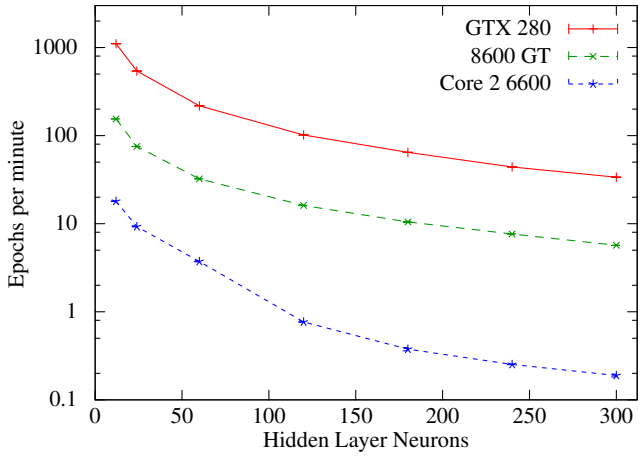

Figure. 6: Number of epochs trained with the MBP for the poker problem.

used. The CPU version was tested on a Intel Core 2 Duo E8400 running at 3.0GHz and the GPU version on two different devices: an NVIDIA GeForce 9800 GT with 14 SMs ($14 \times 8 = 112$ cores) and an NVIDIA GTX 470 with 14 SMs ($14 \times 32 = 448$ cores). For the 9800GT a speedup of 7.49× was obtained while for the GTX470 a speedup of 15.01× was obtained. These results demonstrate the usefullness and scalability of the GPU implementation [47].

Table 1: Speedup (×) provided by the BP and MBP GPU implementations relatively to the CPU, for the Poker problem.

Algorithm	Hidden neurons	GPU 8600 GT	GPU 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

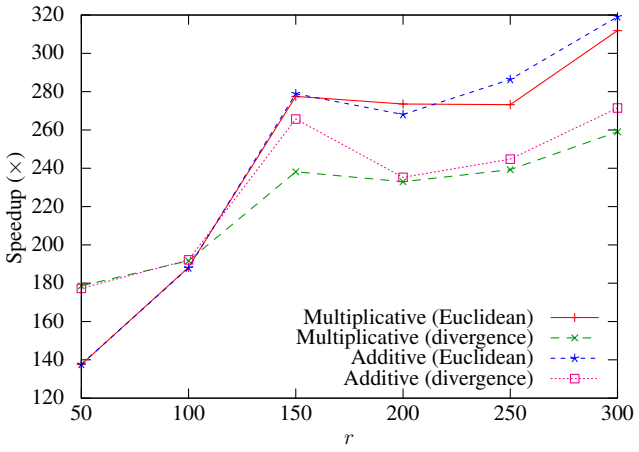

Figure. 7: GPU Speedups for the NMF algorithms.

V. Conclusion

As problems grow increasingly complex and demanding, parallel implementations of ML algorithms become crucial for developing intelligent real-world applications. In this context, the GPU is particularly well positioned to fulfill this need, given its availability, high-performance and relative low-cost. However, based on an in-depth analysis, conducted on recently published papers dealing with GPU ML implementations, we noticed that few ML algorithms have been implemented on the GPU and many of those are not openly shared. This poses a hard-barrier to researchers and engineers aiming to develop ML GPU-based systems. To circumvent this problem, we presented an open source GPU Machine Learning Library (GPUMLib) that currently implements the BP, MBP, NMF and RBF algorithms.

Experimental results using the GPUMLib algorithms on benchmark datasets demonstrated the potential and usefulness of this library, that allows researchers and practitioners to easily select the building blocks necessary to build ML software tailored to meet different requirements and to provide significant speedups. Thus, GPUMLib can be used to create software capable of dealing with real-world complex and computationally demanding problems. The amount of speedup provided by the library algorithms as compared with the CPU counterparts depends on the quantity of data that the algorithms need to process. Assuming data can be processed in parallel, the more data is required to be processed the greater will be the speedup provided by the GPU library algorithm implementations. Finally, we hope to attract the efforts of other reserchers on the endeavor of extending this library. GPUMLib has a broad modular architecture with algorithms ranging from neural networks, support vector machines to genetic algorithms. Future work will extend the GPUMLib to other algorithms opening up the development of hybrid intelligent systems.

Acknowledgments

FCT (Fundação para a Ciência e Tecnologia) is gratefully acknowledged for funding the first author with the grant SFRH/BD/62479/2009.

References

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

[2] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

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

[4] N. Lopes and B. Ribeiro, "Fast pattern classification of ventricular arrhythmias using graphics processing units," in *Proceedings of the 14th Iberoamerican Conference on Pattern Recognition*, vol. 5856 of *LNCS*, pp. 603–610, Springer, 2009.

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

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

[7] Z. Luo, H. Liu, Z. Yang, and X. Wu, "Self-organizing maps computing on graphic process unit," in *Proceedings of the 13th European Symposium on Artificial Neural Networks*, pp. 557–562, 2005.

[8] M. Garland and D. B. Kirk, "Understanding throughput-oriented architectures," *Communications of the ACM*, vol. 53, pp. 58–66, November 2010.

[9] 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.

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

[11] 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.

[12] C.-A. Bohn, "Kohonen feature mapping through graphics hardware," in *Proceedings of the 1998 International Conference on Computational Intelligence and Neurosciences (ICCIN '98)*, (N. Carolina, USA), pp. 64–67, 1998.

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

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

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

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

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

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

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

[20] F. Bernhard and R. Keriven, "Spiking neurons on GPUs," in *Proceedings of the 2006 International Conference on Computational Science*, pp. 236–243, Springer, 2006.

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

[22] S. Harding and W. Banzhaf, "Fast genetic programming on GPUs," in *Proceedings of the 10th European Conference on Genetic Programming (EuroGP '07)*, vol. 4445 of *LNCS*, (Valencia, Spain), pp. 90–101, Springer-Verlag, 2007.

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

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

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

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

[28] S. A. Shalom, M. Dash, and M. Tue, "Efficient k-means clustering using accelerated graphics processors," in *Proceedings of the 10th International Conference on Data Warehousing and Knowledge Discovery*, pp. 166–175, Springer, 2008.

[29] T.-N. Do, V.-H. Nguyen, and F. Poulet, "Speed up SVM algorithm for massive classification tasks," in *Proceedings of the 4th International Conference on Advanced Data Mining and Applications (ADMA '08)*, vol. 5139 of *LNCS*, (Chengdu, China), pp. 147–157, Springer-Verlag, 2008.

[30] S. Grauer-Gray, C. Kambhamettu, and K. Palaniappan, "GPU implementation of belief propagation using CUDA for cloud tracking and reconstruction," in *Proceedings of the 5th IAPR Workshop on Pattern Recognition in Remote Sensing (PRRS '08)*, 2008.

[31] P. A. Sheetal Lahabar and P. J. Narayanan, "High performance pattern recognition on GPU," in *Proceedings of the 2008 National Conference on Computer Vision Pattern Recognition Image Processing and Graphics*, (Gandhinagar, India), pp. 154–159, 2008.

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

[33] 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.

[34] N. Lopes and B. Ribeiro, "GPU implementation of the multiple back-propagation algorithm," in *Proceedings of the 2009 Intelligent Data Engineering and Automated Learning*, vol. 5788 of *LNCS*, pp. 449–456, Springer, 2009.

[35] D. Robilliard, V. Marion-Poty, and C. Fonlupt, "Genetic programming on graphics processing units," *Genetic Programming and Evolvable Machines*, vol. 10, no. 4, pp. 447–471, 2009.

[36] M. Čerňanský, "Training recurrent neural network using multistream extended kalman filter on multicore processor and CUDA enabled graphic processor unit," in *Proceedings of the 19th International Conference on Artificial Neural Networks*, pp. 381–390, Springer, 2009.

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

[38] Y. Xu, H. Chen, R. Klette, J. Liu, and T. Vaudrey, "Belief propagation implementation using CUDA on an NVIDIA GTX 280," in *Proceedings of the 22nd Australasian Joint Conference on Advances in Artificial Intelligence (AI'09)*, vol. 5866 of *LNCS*, (Melbourne, Australia), pp. 180–189, Springer-Verlag, 2009.

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

[40] N. Lopes and B. Ribeiro, "Non-negative matrix factorization implementation using graphic processing units," in *Intelligent Data Engineering and Automated Learning IDEAL 2010*, vol. 6283 of *LNCS*, pp. 275–283, Springer, 2010.

[41] N. Lopes, B. Ribeiro, and R. Quintas, "GPUMLib: A new library to combine machine learning algorithms with graphics processing units," *10th International Conference on Hybrid Intelligent Systems*, pp. 229–232, 2010.

[42] S. Ryoo, C. Rodrigues, S. Baghsorkhi, S. Stone, D. Kirk, and W. Hwu, "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA," in *Proceedings of the 13th ACM Symposium on Principles and practice of parallel programming*, pp. pp.73–82., 2008.

[43] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, and K. Skadron, "A performance study of general-purpose applications on graphics processors using CUDA," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1370–1380, 2008.

[44] A. Asuncion and D. Newman, "UCI machine learning repository," 2007.

[45] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization.," *Nature*, vol. 401, pp. 788–791, October 1999.

[46] M. Qiao, A. H. Sung, and Q. Liu, "Feature mining and intelligent computing for MP3 steganalysis," in *Proc. of the International Joint Conference on Bioinformatics, Systems Biology and Intelligent Computing*, pp. 627–630, IEEE Computer Society, 2009.

[47] R. Quintas, "GPU implementation of RBF neural networks in audio steganalysis," Master's thesis, University of Coimbra, 2010.

Author Biographies

Noel Lopes is Assistant Professor, at the Polytechnic Institute of Guarda in Portugal. He received the MSc in Computer Science from the University of Coimbra, Portugal. He is currently doing his PhD at the University of Coimbra. His main areas of interest are machine learning algorithms and Graphics Processing Unit (GPU) computing.

Bernardete Ribeiro is Professor at the Informatics Engineering Department, Faculty of Science and Technology, University of Coimbra in Portugal. She received a MSc degree in Computer Science and a PhD in Informatics Engineering both from the Informatics Engineering Department, University of Coimbra. Her main publications are in the areas of neural networks and their applications to engineering systems, computational intelligence and support vector machines. She is a member of ACM and IEEE.