# Parallelization of Machine Learning Methods by Using CUDA

Goran Velkoski[1], Monika Simjanoska[2], Sasko Ristov[3] and Marjan Gusev[4]

*Abstract* – **Image analysis, data mining, protein folding and gene sequencing are some examples of high-intensive bioinformatics applications that require high computing resources. In this paper we present a problem of computationally intensive methodology for microarray data analysis, whose performance needs to be improved by using high performance computing techniques. Parallelization is a key computing technique for reducing the time required for the analyses and the classification procedure. GPU provides great level of parallelization based on throughput of vast amount of data needed for machine learning problems. Therefore, we propose a model for machine learning problems parallelization based on GPU programming that will increase the speedup of several stages of the machine learning process.**

*Keywords* –**CUDA, Bioinformatics, Parallelization**

## I. INTRODUCTION

Scientific computing involves the construction of mathematical models and numerical solution techniques to solve scientific and engineering problems that often require a huge number of computing resources to perform large scale experiments, or to cut down the computational complexity into a reasonable time frame [1]. The image analysis, data mining, protein folding and gene sequencing are important tools for biomedical researchers, and examples of high compute and resource intensive scientific applications [2]. When comparing the DNA sequencing throughput to the computer speed, sequencing wins at a rate of about 5-fold per year [3], while computer performance generally follows the Moore's Law, doubling only every 18 or 24 months [4]. The exponential growth of biomedical data requires large storage databases and computing resources. However, the need for computing capacity in the biomedical applications varies dramatically for different stages, i.e. sometimes very big computing power with huge storage space is needed, whereas in the following stage these computationally expensive applications may not require as much computing power as in the previous steps [5].

Bioinformatics researchers are now confronted with analysis of ultra large-scale data sets, a problem that will only increase at an alarming rate in coming years [6]. Therefore, the parallelization seems to be a key computing technique for reducing the time required for the bioinformatics analyses.

GPU is a powerful technology created for graphics 3D rendering towards meeting the need of the 3D gaming industry. Single-threaded processor performance is no longer scaling at historic rates. A GPU that is optimized for throughput delivers parallel performance much more efficiently than a CPU that is optimized for latency [7].

Nowadays a GPU has become an important part of today's computing systems. The GPU's rapid increase in both programmability and capability has spawned a research community that has successfully mapped a broad range of computationally demanding, complex problems to the GPU [8].

CUDA™ is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the GPU [9]. CUDA provides several key abstractions, thus the GPU programming model has proven quite successful at programming multithreaded many code GPUs, to achieve high speedups for research codes and productive solutions. Therefore, hybrid CUDA OpenMP, and Message Passing Interface (MPI) programming approaches emerge in order to create GPU enabled clusters [10].

In this paper we present a CUDA GPU enabled parallelization method for a bioinformatics problem, i.e., a methodology for biomarkers detection and classification analysis of microarray gene expression data. We decided to use CUDA to parallelize this methodology since it is an example of computationally intensive problem whose demand for computing resources varies in different stages of the analyses.

The rest of the paper is organized as follows. In Section II we give an overview of the latest literature for parallel solutions of bioinformatics problems. Our parallel machine learning (ML) approach is presented in Section III. The conclusion and our plans for future implementation of the analysis are discussed in Section IV.

## II. RELATED WORK

In this section we briefly present the latest frameworks for distributed computing and parallelization solutions of bioinformatics problems.

[1]Goran Velkoski is with Innovation LLC, Vostanichka 118, 1000, Skopje, Republic of Macedonia, E-mail: goran.velkoski@innovation.com.mk.

[2]Monika Simjanoska is with the Faculty of Computer Science and Engineering at Ss. Cyril and Methodius University - Skopje, Rugjer Boshkovic 16, Skopje, Republic of Macedonia, E-mail: m.simjanoska@gmail.com.

[3]Sasko Ristov is with the Faculty of Computer Science and Engineering at Ss. Cyril and Methodius University - Skopje, Rugjer Boshkovic 16, Skopje, Republic of Macedonia, E-mail: sashko.ristov@finki.ukim.mk.

[4]Marjan Gushev is with the Faculty of Computer Science and Engineering at Ss. Cyril and Methodius University - Skopje, Rugjer Boshkovic 16, Skopje, Republic of Macedonia, E-mail: marjan.gushev@finki.ukim.mk.

Altekar et al. [11] present a parallel algorithm for Metropolis Coupled Markov Chain Monte Carlo method used in phylogeny. The proposed parallel algorithm retains the ability to explore multiple peaks in the posterior distribution of trees while maintaining a fast execution time. The algorithm has been implemented using two popular parallel programming models: message passing and shared memory. Performance results indicate nearly linear speed improvement in both programming models for small and large data sets.

Multiple sequence alignment (MSA) is an important step in comparative sequence analyses. Katoh and Toh [12] parallelized the three calculation stages, all-to-all comparison, progressive alignment and iterative refinement, of the MAFFT MSA program. They implemented two natural parallelization strategies, best-first and simple hill-climbing. Based on comparisons of the objective scores and benchmark scores between the two approaches, they selected a simple hill climbing approach as the default.

ClustalW [13] is a tool for aligning multiple protein or nucleotide sequences. The alignment is achieved via three steps: pairwise alignment, guide-tree generation and progressive alignment.

Taylor [6] is giving an overview of the Hadoop - MapReduce framework and its current applications in bioinformatics. He concludes that Hadoop and the MapReduce programming paradigm already have a substantial base in the bioinformatics community, especially in the field of next-generation sequencing analysis. This is due to the cost-effectiveness of Hadoop based analysis on commodity Linux clusters, and in the cloud via data upload to cloud vendors who have implemented Hadoop/HBase; and due to the effectiveness and ease-of-use of the MapReduce method in parallelization of many data analysis algorithms.

Considering CUDA, the parallel computing platform of our interest, we present the following applications.

The Smith Waterman algorithm [14] for sequence alignment is one of the main tools of bioinformatics. It is used for sequence similarity searches and alignment of similar sequences.

Ligowski and Rudnicki [15] present an efficient implementation of the Smith Waterman algorithm on the Nvidia GPU. The algorithm achieves more than 3.5 times higher per core performance than the previously published implementation of the Smith Waterman algorithm on GPU, reaching more than 70% of theoretical hardware performance.

Markov clustering (MCL) [16] is becoming a key algorithm within bioinformatics for determining clusters in networks. However, with increasing vast amount of data on biological networks, performance and scalability issues are becoming a critical limiting factor in applications. Bustaman et al. [16] introduce a very fast MCL using CUDA to perform parallel sparse matrix-matrix computations and parallel sparse Markov matrix normalizations, which are at the heart of MCL. They utilized ELLPACK-R sparse format to allow the effective and fine-grain massively parallel processing to cope with the sparse nature of interaction networks data sets in bioinformatics applications. As the results show, CUDA MCL is significantly faster than the original MCL running on CPU.

In context of microarray analysis studies, Shterev et al. [17] have developed a CUDA based implementation, permGPU that employs GPU in microarray association studies. They illustrate the performance and applicability of permGPU within the context of permutation resampling for a number of test statistics. An extensive simulation study demonstrates a dramatic increase in performance when using permGPU on an NVIDIA GTX 280 card compared to an optimized C/C++ solution running on a conventional Linux server.

Zhang et al. [18] develop Parallel Multicategory Support Vector Machines (PMC-SVM) based on the sequential minimum optimization-type decomposition method for SVM (SMO-SVM). It was implemented in parallel using MPI and C++ libraries and executed on both shared memory supercomputer and Linux cluster for multicategory classification of microarray data. PMC-SVM has been analysed and evaluated using four microarray datasets with multiple diagnostic categories, such as different cancer types and normal tissue types.

Salinas and Karmaker [19] have presented a set techniques used to analyse a microarray dataset by computing correlation coefficients between gene expression profiles and transcription factor expression profiles across tissues. Its goal is to find multiple transcription factors that bind together and have a target gene whose transcription is modulated. The technique involves hypothetical heteromeric transcription factor profiles whose expressions are estimated by taking minima for each tissue. A scoring function based on a comparison among the correlation coefficients is used to sort and prioritize combinations of genes and transcription factors. The higher scoring combinations are thought to be more likely to form transcription factor complexes for the gene. By using CUDA enabled NVIDIA GPUs to speed up the computations, they achieved speedups of about 6x.

## III. PARALLEL ML APPROACH

In this section we discuss the ML approach, separate it in several sequential stages and determine the level of parallelization that can be employed on each of them.

### *A. ML Process*

In general, the ML process can be abstracted in three distinct stages: Data Preprocessing, Data Modelling and Data Classification, as depicted in Figure 1.

*1) Data Preprocessing:* The input data is usually retrieved from the publicly available databases, or gathered from experiments done by hospitals, research centers, meteorological stations, etc. The input data is raw and inclined to noise. Therefore, it must be preprocessed in order to be applicable in the methods for knowledge extraction. Since the raw data sets are often measured in gigabytes, the preprocessing process is considered to be the slowest part in the ML procedure. However, a sequential approach of preprocessing and independence in its steps are the features that can be easily parallelized.

*2) Data Modelling:* As soon as the data is preprocessed, it is ready to be an input in a series of methods for the purpose of creating a suitable model for further classification analysis. This part of the process can also benefit from the parallelization, since it can be slow and inefficient when being executed sequentially.

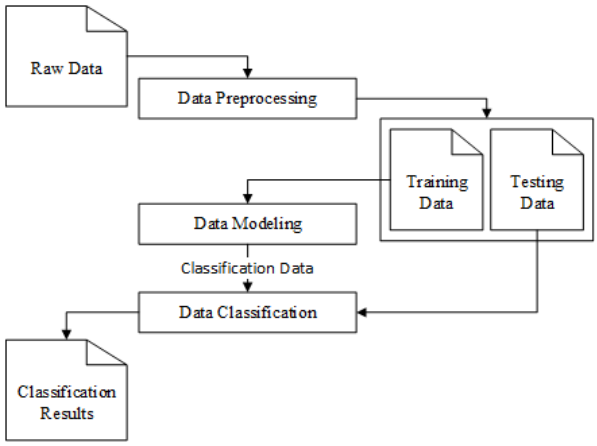


Fig. 1. Machine Learning Approach

3) *Data Classification*: The final goal is achieved when a reliable classifier is created. The testing procedure is usually less computation demanding than the training procedure; however, it depends on the type of classification method used and on the problem dimensionality.

### *B. Bioinformatics ML Methodology*

In this section we present the computation intensive segments of our methodology for microarray analysis presented in [20].

The methodology is developed for two different microarrays technologies and as discussed in Section III-A can also be mainly separated in three parts: preprocessing, building a classification model and the classification process itself.

*1) Preprocessing:* The preprocessing consists of few methods executed in the following order: Quantile Normalization (QN), Low Entropy Filter (LEF), T-test, False Discovery Rate (FDR) and Volcano Plot (VP).

The QN is a gene expression normalization method for making two distributions identical in statistical properties. It can be easy parallelized since we normalize each column of the input matrix distinctively.

LEF is a filter that removes the genes with low variability in their expression levels across the samples. It is based on the entropy computations of each gene (row) in the data matrix, and therefore, can also be parallelized.

The T-test computes the value of a t-statistics for the difference between means of the two columns of data.

FDR uses the p-values from the T-test in order to discover the false positive genes, i.e. the genes that were found to be significant when in reality there is no statistical significance. It is computed for each gene and the process can be parallelized.

When applying the VP method, we consider that the number of genes has significantly decreased from thousands to hundreds and therefore, there is no need of parallelization.

*2) Building the classification model:* In order to build the classification model, we perform hypothesis tests to determine the most probable distributions of the genes. This process can be parallelized since we perform tests for four types of distributions for each of the genes. During the testing, the parameters of the distributions are estimated by using the Maximum Likelihood Estimation (MLE) method, which are then used in the Chi-square goodness-of-fit test. The input of Chi-square goodness-of-fit is also a vector of the expression values of a given gene.

*3) Classification procedure:* The classification method that we proposed is based on a calculation of the Bayesian posterior probability P (Ci| $\vec{x}$ ):

$$p(C_i \mid \vec{x}) = \frac{p(\vec{x} \mid C_i) * P(C_i)}{\sum_{1}^{2} p(\vec{x} \mid C_i) * P(C_i)} \quad (1)$$

### *C. Parallel execution on CUDA GPU*

We already mentioned that large number of methods in the ML process can be parallelized. Since almost always the methods are without data dependency, we can easily partition the data and create full data parallelization.

Because of the large number of multiple distinctive microarray data portions on which several distinctive operations are used as single instructions sequentially, we conclude that the GPU SIMD architectures might be an appropriate platform.

In Figure 2 we present the parallelization procedure. For each of the above methods a distinctive kernel is created on the GPU. Each kernel is executed on optimal number of cores on GPU and each core gets a single vector data to work on.

If the time needed for each core to do its job is $t_c$, and $N_v$ and $N_c$ denote the number of vectors and the number of cores available correspondingly, then the time needed for a single method execution $T_m$ is calculated by using the following equation:

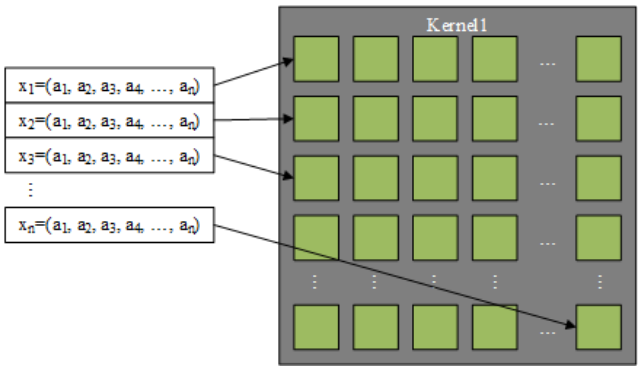


Fig. 2. Parallelization on GPU Architectures

$$T_m = \frac{N_v * t_c}{N_c} \quad (2)$$

In the best case scenario the number of vectors $N_v$ will be equal to the number of cores $N_c$ and therefore, $T_m = t_c$, i.e., the time needed for a method execution equals the time needed for a single core to finish its task.

Since no GPU provides unlimited number of cores, we can conclude that this is just an isolated case. Most commonly the CUDA enabled GPU devices are packed up with $N_c$ =512, or $N_c$ = 1024 cores. Therefore, the expected speedup $S$ according to Gustafson's law when executed on CUDA enabled GPU is:

$$S = \frac{T_{ns}}{T_{mp}} = \frac{N_c * T_{mp}}{T_{mp}} = N_c \quad (3)$$

where $T_{ms}$ is the method execution time when implemented sequentially and $T_{mp}$ is the method execution time executed in parallel.

Since the GPU involves additional latency for data transfer and core utilization, we hypothesize that the speedup will not reach 512, or 1024 with fully utilized GPU; however, this will be part of our future work on the topic.

In the end, each of the kernels will be executed sequentially and therefore the overall speedup will abide the speedup for each of the methods.

## IV. Conclusion

In this paper we present a computation intensive bioinformatics methodology for biomarkers detection and classification analysis of microarray gene expression data. Our ML approach is comprised in three distinctive parts, where each part consists of one or more different methods. Considering the methods used in the methodology, we can conclude that the analyses are usually performed in context of genes, or patients, but never depend on both.

We propose a parallelization procedure to reduce the time for execution of the distinct methods. Because of the nature of the problem we decided that a parallelization by using GPU computing may be convenient. Consequently, we discussed the advantages of using CUDA technology for parallelization.

Based on CUDA GPU properties we modelled a kernel based solution for ML process theoretically. In our solution each kernel is an implementation of a single method. Since the methods are sequentially executed, the kernel execution is also sequential.

As future work we will implement large variety of methods for ML by using CUDA and we will test if our CUDA implementations achieve the expected speedup level.

## References


[1] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on. IEEE, 2009, pp. 4–16.

[2] A. Rosenthal, P. Mork, M. H. Li, J. Stanford, D. Koester, and P. Reynolds, "Cloud computing: A new business paradigm for biomedical information sharing," Journal of Biomedical Informatics, vol. 43, no. 2, pp. 342–353, 2010.

[3] L. D. Stein et al., "The case for cloud computing in genome informatics," Genome Biol, vol. 11, no. 5, p. 207, 2010.

[4] G. E. Moore et al., "Cramming more components onto integrated circuits," Proc. of IEEE, vol. 86, no. 1, pp. 82–85, 1998.

[5] H. Chae, I. Jung, H. Lee, S. Marru, S.-W. Lee, and S. Kim, "Bio and health informatics meets cloud: Biovlab as an example," Health Information Sci. and Systems, vol. 1, no. 1, p. 6, 2013.

[6] R. C. Taylor, "An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics," BMC bioinformatics, vol. 11, no. Suppl 12, p. S1, 2010.

[7] J. Nickolls and W. J. Dally, "The GPU computing era," IEEE micro, vol. 30, no. 2, pp. 56–69, 2010.

[8] 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.

[9] NVIDIA. (2014, May) NVIDIA CUDA - Parallel Programming and Computing Platform. [Online]. Available: http://www.nvidia.com/object/ cuda home new.html

[10] C.-T. Yang, C.-L. Huang, and C.-F. Lin, "Hybrid cuda, openmp, and mpi parallel programming on multicore gpu clusters," Computer Physics Communications, vol. 182, no. 1, pp. 266–269, 2011.

[11] G. Altekar, S. Dwarkadas, J. P. Huelsenbeck, and F. Ronquist, "Parallel metropolis coupled markov chain monte carlo for bayesian phylogenetic inference," Bioinformatics, vol. 20, no. 3, pp. 407–415, 2004.

[12] K. Katoh and H. Toh, "Parallelization of the mafft multiple sequence alignment program," Bioinformatics, vol. 26, no. 15, pp. 1899–1900, 2010.

[13] K-B. Li, "ClustalW-MPI: ClustalW analysis using distributed and parallel computing," Bioinformatics, vol. 19, no. 12, pp. 1585-1586, 2003.

[14] F. T. Smith and S. M. Waterman, "Identification of common molecular subsequences," in Journal of molecular biology, vol. 147, no. 1, pp. 195-197, Elsevier, 1981.

[15] L. Ligowski and W. Rudnicki, "An efficient implementation of smith waterman algorithm on gpu using cuda, for massively parallel scanning of sequence databases," in Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on. IEEE, 2009, pp. 1–8.

[16] A. Bustamam, K. Burrage, and N. A. Hamilton, "Fast parallel markov clustering in bioinformatics using massively parallel computing on gpu with cuda and ellpack-r sparse format," IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), vol. 9, no. 3, pp. 679–692, 2012.

[17] I. D. Shterev, S.-H. Jung, S. L. George, and K. Owzar, "permgpu: Using graphics processing units in RNA microarray association studies," BMC bioinformatics, vol. 11, no. 1, p. 329, 2010.

[18] C. Zhang, P. Li, A. Rajendran, Y. Deng, and D. Chen, "Parallelization of multicategory support vector machines (pmc-svm) for classifying microarray data," BMC bioinformatics, vol. 7, no. Suppl 4, p. S15, 2006.

[19] E. A. Salinas and A. Karmaker, "Cuda-accelerated data-mining for putative heteromeric transcription factors and target genes using microarray gene expression profiles."

[20] M. Simjanoska, A. Madevska Bogdanova, and Z. Popeska, "Bayesian posterior probability classification of colorectal cancer probed with Affymetrix microarray technology," in Information & Communication Technology Electronics & Microelectronics (MIPRO), 2013 36th International Convention on. IEEE, 2013, pp. 959–964.