SVM Parallelization in GPU

Henry E.L. Cagnini[1], Ana T. Winck[1], Rodrigo C. Barros[2]

[1]Federal University of Santa Maria, Santa Maria, Brazil
{henry, ana}@inf.ufsm.br

[2]Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil
rodrigo.barros@pucrs.br

Abstract

Support Vector Machines (SVMs) is one of the most efficient methods for data classification in machine learning. Several efforts were dedicated towards improving its performance through source-code parallelization, particularly within the Graphics Processor Unit (GPU). Those studies make use of the well-known CUDA framework, which is provided by NVIDIA for its graphics cards. Nevertheless, the main disadvantage of CUDA-based solutions is that they are specific to NVIDIA cards, reducing the applicability of such solutions in heterogeneous environments. In this work, we propose the parallelization of SVMs through the OpenCL framework, which allows the generated solution to be portable to a wide range of GPU manufacturers. The proposed approach parallelizes the most costly steps that are performed when training an SVM. Our solution achieves a considerable speed-up regarding the sequential version of SVM, with little to no loss of predictive performance.

Introduction

As the size of a database increases, the computational power required to run a machine learning algorithm over this database also increases. However, at a certain point the required computational power is too high to be practical. For such large problems, new computational approaches may be taken into account. One of this approaches is the parallelization of machine learning's source code in GPU.

A parallelized source code can run in several devices simultaneously, such as machines in a computer network or stream processors of a Graphics Processor Unit (GPU). The GPU is more convenient for this task mainly for its easiness of access: all contemporary computers have a GPU, whether it is on-board (at the CPU chip area) or off-board (attached to the computer's motherboard through a bus).

In this work it's proposed to improve performance of the binary classification process, through parallelization of training phase of SVM at the GPU, enhancing SVM overall performance in relation to its sequential (original) version. The framework used for parallelization is the OpenCL [2], and the SVM source code library is the LIBSVM [1].

Methodology and Implementation

The methodology used refers to the identification of most costly functions in the training phase of classification process of SVM, and subsequent implementation of these functions in GPU. When analyzing the original source code of LIBSVM [1], two functions use up to 80% of CPU time: dot product between instances and Radial Basis Function, a type of kernel function.

Two main modifications were made to adapt these functions to the GPU: new representation of datasets and replacement of auxiliary functions. The new representation of datasets seeks to take advantage of one particularity of the datasets used (provided by UCI Machine Learning Repository [3] and showed at Table 1), in which only binary values of predictive attributes are used. With this modification, the default value storage approach of LIBSVM, which uses 12 bytes to store one single attribute, could be replaced by a bit storage, where each instance would use only 16 bytes.

The replacement of auxiliary functions replaces the mathematical functions used by Radial Basis Function in original LIBSVM implementation by OpenCL [2] equivalents. Although this does not modify the functioning of the kernel function, it interferes in the accuracy of the generated predictive models, as shown in Table 2. More details are presented in section 1.

Característica	Bases de dados				
	a1	a6	a7	a8	a9
Classes	2	2	2	2	2
Atributos	123	123	123	123	123
Número de instâncias de treino	1.605	11.220	16.100	22.696	32.561
Número de instâncias de teste	30.956	21.341	16.461	9.865	16.281

Table: Datasets used in the identification of LIBSM's most costly functions and parallelization of source code. Avaiable at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

Results and discussions

Two optimized versions were generated: one with the new representation of the dataset, executing sequentially in CPU, and other with the new representation and parallel processing in GPU (the former is avaiable at `https://bitbucket.org/hcagnini/parallel_libsvm_train`). Both versions achieve better performance than the original version, being the GPU version faster than the CPU optimized. However, GPU version achieves worse accuracy than the CPU optimized version; this happens because the auxiliary functions are implemented distinctly in the original LIBSVM's source code and OpenCL's equivalent. Results for processing performance and accuracy may be viewed at Table 2.

Dataset	Original		Optimized Sequential			Optimized Parallel		
	Accuracy	Time	Accuracy	Time	*speedup*	Accuracy	Time	*speedup*
a1a	83,59%	0,63s	83,59%	0,69s	0,91×	83,59%	2,75s	0,23×
a6a	84,17%	17,33s	84,17%	14,23s	1,22×	75,87%	18,24s	0,95×
a7a	84,56%	34,28s	84,58%	28,24s	1,21×	76,17%	28,68s	1,19×
a8a	85,01%	66,83s	85,01%	51,73s	1,29×	76,33%	40,73s	1,64×
a9a	84,82%	133,88s	84,82%	103,83s	1,29×	76,38%	80,74s	1,66×

Table: Time and accuracy for original, optimized sequentially and optimized parallel versions of the training phase of SVM classification process. The speedup column refers to how much the optimized version is faster than the original; a speedup less than 1 means that it runs slower than the original version.

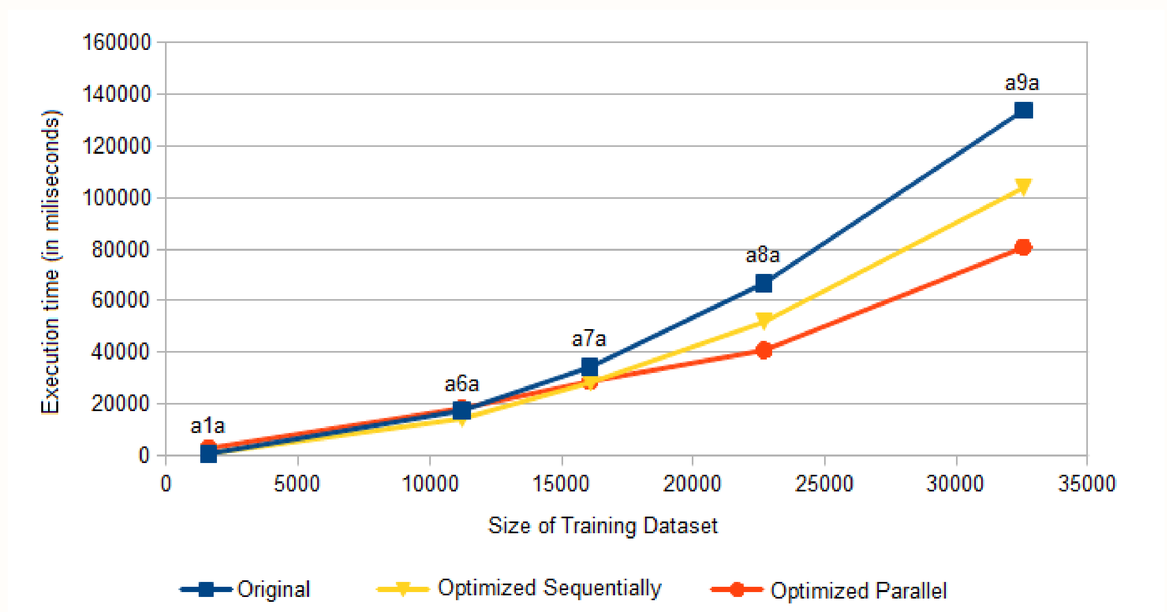

Figure: Comparison of original, optimized sequentially and optimized parallel versions of LIBSVM's code. As the size (number of instances) of dataset increases, the speedup of the optimized versions also increases.

Future works

The proposed implementation of LIBSVM in GPU is successful in enhancing its processing performance. Although it suffers from a loss in accuracy, some changes can be made to correct this discrepancies, while keeping the gained performance:

1. Implementation of Radial Basis Funcion's auxiliary functions, preventing the loss of accuracy of the generated predictive models;
2. Parallelization of other LIBSVM's kernel methods, not only Radial Basis Function;
3. Matrix multiplication, leaving the new representation of datasets behind and allowing a wider range of datasets to be used by the implementation;
4. Use of bigger datasets, in the instance number as well as in the attribute number, to analyze better the effects of parallelization in the performance of LIBSVM's training phase.

References

[1] Chang, Chih-Chung and Lin, Chih-Jen. LIBSVM: A library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology*. Vol. 2, pp. 27:1-27:27, 2011.

[2] Khronos: The OpenCL 1.2 Specification. <http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf>, 2012.

[3] K. Bache, M. Lichman: UCI Machine Learning Repository. <http://archive.ics.uci.edu/ml>, 2013.