GPU acceleration for support vector machines

Article · January 2011

CITATIONS
57

READS
508

4 authors, including:

Anastasios Dimou
The Centre for Research and Technology, Hellas
34 PUBLICATIONS **412** CITATIONS
SEE PROFILE

Ioannis (Yiannis) Kompatsiaris
The Centre for Research and Technology, Hellas
792 PUBLICATIONS **8,226** CITATIONS
SEE PROFILE

Proc. 12th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2011), Delft, The Netherlands, April 2011.

GPU ACCELERATION FOR SUPPORT VECTOR MACHINES

Andreas Athanasopoulos, Anastasios Dimou, Vasileios Mezaris, Ioannis Kompatsiaris

Informatics and Telematics Institute / Centre for Research and Technology Hellas
6th Km Charilaou-Thermi Road, Thermi 57001, Greece
{athanaso, dimou, bmezaris, ikom}@iti.gr

ABSTRACT

This paper presents a GPU-assisted version of the LIBSVM library for Support Vector Machines. SVMs are particularly popular for classification procedures among the research community, but for large training data the processing time becomes unrealistic. The modification that is proposed is porting the computation of the kernel matrix elements to the GPU, to significantly decrease the processing time for SVM training without altering the classification results compared to the original LIBSVM. The experimental evaluation of the proposed approach highlights how the GPU-accelerated version of LIBSVM enables the more efficient handling of large problems, such as large-scale concept detection in video.

1. INTRODUCTION

Nowadays, the Support Vector Machine (SVM) methodology is widely used among the research community for training classifiers. Its popularity can be attributed to the good results it produces on a variety of classification problems; in the field of image/video processing and analysis, for example, SVMs are used for video segmentation to shots [1], concept detection in images and videos [2],[3] and other tasks. The standard SVM is a non-probabilistic, binary, linear classifier. To add non-linear classification capabilities, the kernel trick was introduced that transforms the problem to a higher dimensional feature space, in order to make classes linearly separable [4]. Current SVM implementations (eg. LIBSVM [5]) offer a variety of features such as different kernels, multiclass classification and cross-validation for model selection.

Classification problems are becoming more and more demanding and challenging for the classifier in the quest for higher accuracy. The size of the problem increases with the training data available and the size of the input vectors. Moreover, techniques such as SVM parameter tuning and cross validation, which improve classification, also increase computational costs. In some case, the processing time for training a SVM model can be in the order of days. Thus, it is crucial to speed-up the process of model training without sacrificing the accuracy of the results.

The computational cost can be reduced either by using multiple PCs to parallelize the work, as in [6], or by using the processor in the graphics cards as a co-processor. The latter option is particularly appealing because modern Graphics Processing Units (GPUs) have become faster and more efficient than traditional CPUs in certain types of computations. They are optimized for rendering real-time graphics, a highly computational and memory intensive problem with enormous inherent parallelism; thus, relative to a CPU, a much larger portion of a GPU's resources is devoted to data processing than to caching or control flow. In order, though, to realize large performance gains compared to a CPU, parallelism has to be uncovered in the desired application. NVIDIA has introduced a programming framework for their GPUs, Compute Unified Device Architecture (CUDA) [7], that can be used to exploit the advantages of their architecture.

In this paper we try to give an insight on past work on the GPU-accelerated SVMs and suggest a new solution for which we make source code publicly available[1]. The remainder of the paper is organized as follows: in section 2, previous work on SVMs and using GPUs is discussed. In section 3 the methodology of model training and the algorithm implementation are presented. Experimental results are reported in section 4 and finally conclusions are drawn in section 5.

2. RELATED WORK

A number of CPU-based implementations of the SVM methodology are publicly available today. LIBSVM [5], SVM light [8], mySVM [9] and TinySVM [10] are only some examples. LIBSVM is one of the most complete implementations, providing many options and state-of-the-art performance. Therefore, it has become the SVM implementation of choice for a large part of the research community. Thus, it was chosen as the basis of our work on GPU-accelerated SVMs.

Several approaches to exploiting the GPU power have been proposed. Catanzaro et al [11] presented a SVM solver that works on GPUs. The implementation used the Sequential

This work was supported by the European Commission under contract FP7-248984 GLOCAL.

[1] http://mklab.iti.gr/project/GPU-LIBSVM

Minimal Optimization (SMO) [12] algorithm also used by LIBSVM, but their GPU implementation used solely single precision floating point arithmetics, rather than double precision that is used by LIBSVM. Consequently, the results showed a performance deterioration compared to LIBSVM (e.g 23,18% difference in accuracy for the "FOREST" dataset as shown in [13]). Carpenter [13] presented another SVM implementation that uses the same SMO solver but uses a mixture of double and single precision arithmetics to enhance the accuracy of the results. The performance was boosted, especially for dense datasets, but still could not duplicate the LIBSVM results. Both solutions presented above also lack a number of features that LIBSVM offers, e.g. cross-validation, that are valuable for achieving state-of-the-art results. In [14], relevant work has been done for the x^2 kernel. In our work, LIBSVM is used as the basis and it is modified to use the GPU to accelerate parts of the procedure, most notably the computation of the RBF kernel. The functionality of LIBSVM is fully preserved, all original features are available (such as cross-validation, probability estimation, weighting etc.) and results identical to those of the original LIBSVM are produced.

3. ACCELERATING SVM COMPUTATIONS

3.1. Methodology

The SVM classification problem can be separated in two different tasks; the calculation of the kernel matrix (KM) and the core SVM task of decomposing and solving a series of quadratic problems (SMO solver) to extract a classification model. The increasing size of the input data leads to a huge KM that cannot be calculated and stored in memory. Therefore, the solver needs to calculate on-the-fly portions of the KM, a processing and memory-bandwidth intensive procedure. Calculating the KM values has to be repeated multiple times when cross validation is adopted to enhance the classification process. Moreover, using the SVM parameter optimization requires a huge number of recalculations (e.g. for 5-fold cross validation and 110 different parameter sets tested, which are the default parameters for the "easy.py" script included in LIBSVM, that would lead to 550 repetitions). For large datasets this procedure can account for up to 90% of the total processing time required [14]. Acceleration can be achieved in two ways: i) the KM is pre-calculated and it is passed to the SVM to avoid recalculation during the cross-validation loops, and ii) the parallelism and the memory bandwidth of a GPU are exploited to speed up the process. The solution of the quadratic problems can also benefit from parallelism, but accuracy matters are raised. LIBSVM is using double precision for calculations but only the latest GPUs do support double precision, leading to accuracy inconsistencies. Therefore, we chose to deal with the matrix calculations only.

The methodology that was followed combines an architectural change and the GPU-acceleration. The differences are visualized in Fig. 1. In the original LIBSVM the KM is recalculated in every iteration of the cross-validation function, exploiting only previously cached results in the system memory. On the other hand, in the modified version that we propose the KM is calculated only once for every set of parameters tested. This architectural change accelerates the procedure by a factor of n, where n is the number of cross-validations employed. Furthermore, the combination of CPU and GPU is used to speed-up the calculation of the KM elements, as discussed in the sequel. In contrast to other similar methods (e.g. [11], [13]), the core SVM computations are kept intact, to preserve the accuracy of the results.

3.2. Calculation of the Kernel Matrix on the GPU

The pre-calculation is performed combining CPU and GPU to achieve maximum performance as suggested in [13]. LIBSVM offers a variety of classification modes and kernels. We chose to work with C-SVC mode and the RBF kernel.

The RBF kernel function $\Phi(x, y)$ is calculated as

$$\Phi(x, y) = \exp^{-\gamma||x-y||^2}, \quad (1)$$

where $x = x_i, y = y_i$ are two training vectors. We can expand

$$||x - y||^2 = \sum(x_i)^2 + \sum(y_i)^2 - 2 * \sum(x_i * y_i). \quad (2)$$

Algorithm 1 shows in detail how the KM is computed, using this kernel.

Algorithm 1 Kernel Matrix Computation

1: Pre-calculate on the CPU the sum of squares of the elements for each training vector.
2: Convert the training vectors array into column wise format, i.e. create the translated matrix of the array, a step required for the work on the GPU.
3: Allocate memory on the GPU for the training vectors array.
4: Load the training vectors array, in the translated format, to the GPU memory.
5: FOR (each training vector) DO

- Load the training vector to the GPU (because the version on the GPU is in a translated format).
- Perform the matrix-vector multiplication, i.e. calculate the dot products, using CUBLAS.
- Retrieve the dot products vector from the GPU.
- Calculate the line of the KM by adding the training vector squares, then calculating $\Phi(x, y)$ according to Equation 1.

END DO
6: De-allocate memory from GPU.

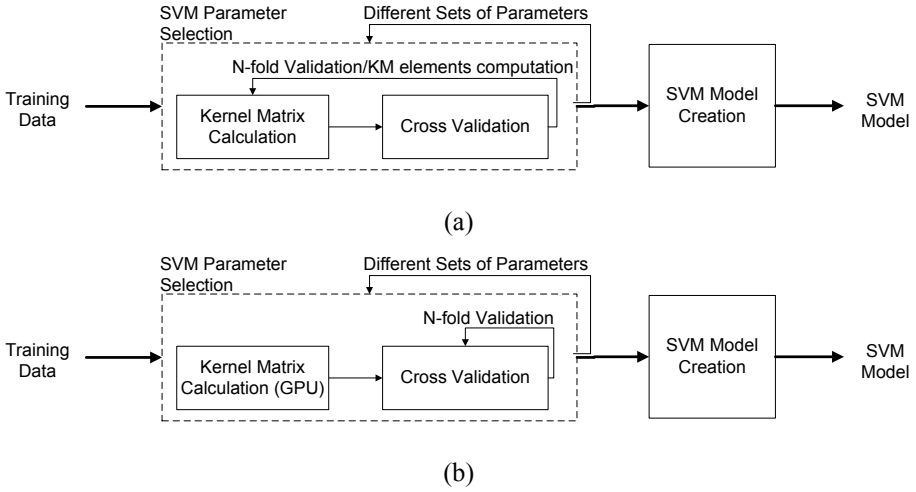

Fig. 1. SVM training procedure for (a) Original, (b) GPU-Accelerated LIBSVM.

As can be seen, the proposed algorithm is based on pre-calculating on the CPU for each training vector x the sum $\sum(x_i)^2$, while calculating the dot-products $\sum(x_i y_i)$ using the CUBLAS [15] library provided from NVIDIA.

4. EXPERIMENTAL RESULTS

The experiments were run on two PCs. The first was equipped with a quad core Intel Q6600 processor with 3GB of DDR2 RAM, Windows-XP 32-bit OS and a NVIDIA 8800GTS graphics card with 512MB of onboard RAM. The second PC was equipped with a quad core Intel Core-i7 950 processor with 12GB of DDR3 RAM, Windows-7 64-bit OS and a NVDIA GTS-250 graphics card with 512MB onboard RAM.

SVMs were trained and used for the detection of high level features in video shots, on the training portion of the TRECVID 2007 dataset which comprises 110 videos of 50 hours duration in total. Two separate low-level visual descriptors were used as input to the SVMs for the experiments; the first is created by extracting SIFT [16] descriptors from a keyframe of every shot of the videos and using the Bag-of-Words (BoW) methodology, resulting to a descriptor vector of dimension 500; the second is created by combining the SIFT-based BoW descriptors of the first set with the Feature Track descriptors of [2] and pyramidal decomposition schemes, resulting to a descriptor vector of dimension 6000. For the experiment, 20 high-level features (e.g. "bus", "nighttime" etc.) were considered, i.e. 20 SVMs were separately trained. A variable number of input training vectors, ranging from 36 up to 3772, was available for each of them according to the ground truth annotations and was used. The script included in the LIBSVM package for parameter tuning, which involves 5-fold cross validation, was used in all experiments. LIBSVM ran in the multi-threaded mode, setting "local workers" number to 4 in both modes, CPU and CPU+GPU.

The resulting processing times for the various high-level features (and, thus, for a variable number of input vectors to the SVM) can be seen in Figs. 2 and 3. These show that GPU acceleration speeds-up the training process, while in all experiments the results of SVM training were shown to be identical. The amount of the acceleration is highly dependent on both the number of input vectors and the size of the vectors. For small input data the whole process is very fast and the GPU-induced acceleration is hardly noticeable. As the input data increases, the processing time with the traditional LIBSVM is increasing at an almost exponential rate (Fig. 2). On the contrary, using the GPU the processing time increases at a much lower rate. Similarly in Fig. 3 (where the time axis is in logarithmic scale), the processing time when exploiting the GPU is up to one order of magnitude lower in comparison to using the CPU alone. These results were expected, since the number of the KM elements is equal to the square of the number of training vectors. Thus, the amount of calculations required to produce the KM increases with the square of the number of training vectors. For large input data, the time spent on calculating the matrix elements becomes dominant, compared to the other processes of LIBSVM, and thus the GPU acceleration becomes more pronounced. It should be stressed that visual concept detection is just one example of LIBSVM usage. Any other application that uses LIBSVM can also benefit, depending on the size of the input data.

Since the GPU-accelerated LIBSVM loads the input data to the graphics card memory and current graphics cards have a limited amount of memory on board, it is necessary to give some hints on the memory requirements. GPU-accelerated LIBSVM stores the input elements with single precision, i.e. 4 bytes per element. For example, an input dataset of 20000 vectors of dimension 500 needs 20000 x 500 x 4 = approximately 40MB of video memory. For multiple threads, the memory requirements have to be multiplied by the number of

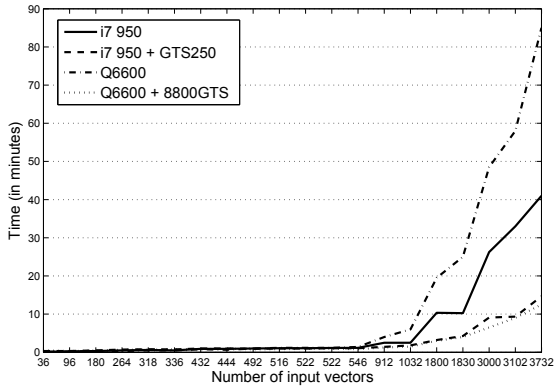

Fig. 2. Processing times for input vectors of size 500.

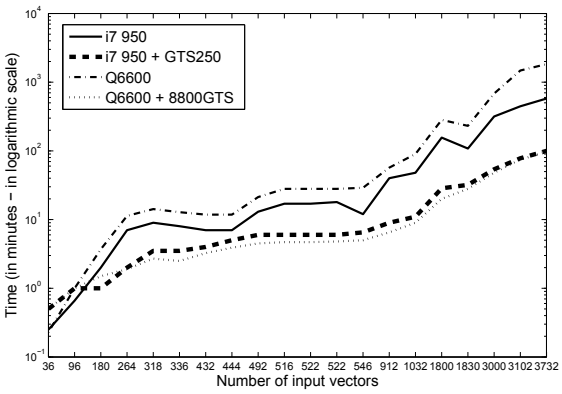

Fig. 3. Processing times for input vectors of size 6000.

threads ("local workers") used.

5. CONCLUSIONS

A modification of the LIBSVM library that takes advantage of the processing power of the GPU was presented in this paper. The modification pre-calculates the KM elements, combining the CPU and the GPU to accelerate the procedure. The modified version of LIBSVM produces identical results with the original LIBSVM. Our experimental setup showed that the GPU-accelerated LIBSVM gives a speed improvement that steeply increases with the size of the input data, enabling the efficient handling of large problems. The experimental evaluation also showed that the GPU speed has a significantly higher impact than the CPU speed in the total processing time, especially for large input data. Using a fast graphics card, even an older PC seems to provide similar performance with a fast PC using the same graphics card.

The GPU-accelerated LIBSVM is available at `http://mklab.iti.gr/project/GPU-LIBSVM`

6. REFERENCES

[1] E. Tsamoura, V. Mezaris, and I. Kompatsiaris, "Gradual transition detection using color coherence and other criteria in a video shot meta-segmentation framework," in *Proc. ICIP-MIR*, San Diego, CA, USA, Oct. 2008.

[2] V. Mezaris, A. Dimou, and I. Kompatsiaris, "On the use of feature tracks for dynamic concept detection in video," in *Proc. ICIP*, Hong Kong, China, Sept. 2010.

[3] K.E.A. van de Sande, T. Gevers, and C.G.M. Snoek, "Evaluation of color descriptors for object and scene recognition," in *Proc. CVPR*, Anchorage, Alaska, USA, June 2008.

[4] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[5] C.-C. Chang and C.-J. Lin, "LIBSVM: a library for support vector machines," 2001, Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[6] E.Y. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, and H. Cui, "PSVM: Parallelizing Support Vector Machines on Distributed Computers," in *Proc. NIPS*, Vancouver, B.C., Canada, Dec. 2007.

[7] NVIDIA, *CUDA Programming Guide 2.0*, 2008.

[8] T. Joachims, "SVM light, http://svmlight.joachims.org," 2002.

[9] S. Rueping, "mySVM-Manual, http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/," 2000.

[10] T. Kudo, "Tinysvm : Support vector machines," *http://www.chasen.org/ taku/software/TinySVM/*.

[11] B. Catanzaro, N. Sundaram, and K. Keutzer, "Fast support vector machine training and classification on graphics processors," in *Proc. ICML*, Helsinki, Finland, 2008.

[12] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," *Technical Report MSR-TR-98-14, Microsoft Research*, 1998.

[13] A. Carpenter, "cuSVM: a CUDA implementation of support vector classification and regression," 2009, Software available at http://patternsonascreen.net/cuSVM.html.

[14] K.E.A. van de Sande, T. Gevers, and C.G.M. Snoek, "Empowering visual categorization with the gpu," *IEEE Trans. on Multimedia*, vol. 13, no. 1, pp. 60–70, 2011.

[15] NVIDIA, *CUBLAS Library 2.0*, 2008.

[16] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. of Computer Vision*, vol. 60, pp. 91–110, 2004.