Performance Comparison of GPU and FPGA architectures for the SVM Training Problem

Markos Papadonikolakis [1], Christos-Savvas Bouganis [2], George Constantinides [3]

Department of Electrical and Electronic Engineering, Imperial College London
Exhibition Road, South Kensington, London, SW7 2AZ, UK
[1] markos.papadonikolakis07@imperial.ac.uk
[2] christos-savvas.bouganis@imperial.ac.uk
[3] g.constantinides@imperial.ac.uk

***Abstract*—The Support Vector Machine (SVM) is a popular supervised learning method, providing high accuracy in many classification and regression tasks. However, its training phase is a computationally expensive task. In this work, we focus on the acceleration of this phase and a geometric approach to SVM training based on Gilbert's Algorithm is targeted, due to the high parallelization potential of its heavy computational tasks. The algorithm is mapped on two of the most popular parallel processing devices, a Graphics Processor and an FPGA device. The evaluation analysis points out the best choice under different configurations. The final speed up depends on the problem size, when no chunking techniques are applied to the training set, achieving the largest speed up for small problem sizes.**

I. Introduction

Support vector machines (SVMs) [1] have proven to be very effective for supervised classification with a wide range of applicability. Much research has focused on the training phase of SVMs, which is a computational expensive problem and needs of accelerating the problem arise. Moreover, especially in online training problems where the time constraints are tight, the performance criteria play the most vital role for the implementation decisions of the application.

The SVM training problem can be formulated as a Quadratic Programming (QP) problem. Due to the QP problem complexity, several decomposition methods have been proposed, such as SMO [2] and SVM$^{\text{LIGHT}}$ [3]. Some other works like [4] approach the problem from a geometric point of view. This solution is characterized by a simple algorithmic flow and offers significant parallelization potential which makes it a favourable target for a parallel processing device.

In the last decade, mainly because of the continuously increasing graphics processing demands of the video game industry, Graphics Processing Units (GPUs) have evolved into massively parallel computing engines. NVidia's release of the Compute Unified Device Architecture (CUDA) [5] enabled the ease of parallel programmability for GPUs and has been the major step forward to draw research and industry attention. GPUs have already been applied to numerous applications with parallel computing characteristics.

On the other hand, FPGA solutions can also offer high throughput to numerous data-intensive applications with critical time constraints. Modern FPGA devices offer a large number of LUTs, DSP blocks and a hierarchy of different memory sizes, providing high level of design flexibility. Moreover, FPGA runtime reconfigurability allows the design to be scalable and adaptive to different types of input data.

This work focuses on a performance comparison between a GPU and an FPGA implementation for the SVM training. A geometric approach to the SVM training based on Gilbert's Algorithm [6] is targeted. The main objective is the exploitation of the algorithm's parallelism and its efficient mapping on these two popular parallel processing devices, in order to accelerate the algorithm's iteration for the training phase of SVMs. A GPU implementation is presented and is compared to a previously proposed FPGA work [7]. The motivation of this work is to provide an evaluation analysis performance of the two implementations and to identify the most favourable one, under different input configurations and resource constraints.

The rest of this paper is organized as follows: Section II gives the background on the SVM training and presents the targeted algorithm. In Section III, the GPU implementation is presented, along with an overview of the FPGA work proposed in [7]. Section IV provides the performance comparison. The conclusion is presented in Section V.

II. SVM Training

For a two-class classification problem, the SVM objective is to construct a separating hyperplane $\mathbf{w} \cdot \mathbf{x} - b = 0$ to attain maximum separation between the classes, as shown in Fig. 1. The classes' hyperplanes are parallel to the separating one, lying on each of its sides. The Euclidean Distance between the two hyperplanes is $2/||\mathbf{w}||_2$, thus the objective of SVMs is to maximize the distance between the classes' hyperplanes or, in other words, to minimize $\|\mathbf{w}\|_2$:

$$\min \tfrac{1}{2}\|\mathbf{w}\|_2^2, \text{ s.t. } y_i(K(\mathbf{w}, \mathbf{x}_i) - b) \geq 1, \ 1 \leq i \leq N, \quad (1)$$

where $K(\mathbf{x}_i, \mathbf{x}_j)$ denotes a kernel function, $\mathbf{x}_i$ is the training data, label y_i denotes the belonging class of datum $\mathbf{x}_i$ and takes the values -1, 1, $\mathbf{w}$ is the perpendicular vector to the hyperplane direction, b is the offset to the origin and N is the training set size. We focus on the training phase, whose task is the identification of those training samples that lie closest to the hyperplane and determine its direction; these samples are called *Support Vectors* (SVs).

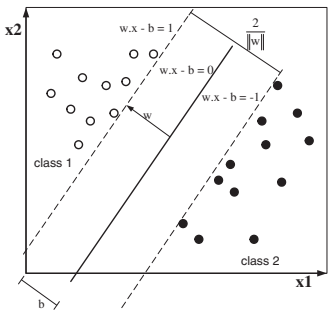

Fig. 1. SVM separating hyperplane.

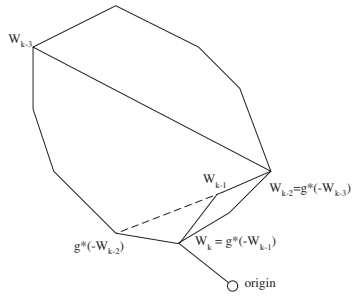

Fig. 2. Iteration steps of Gilbert's Algorithm.

In many real-world classification problems, it is often not feasible to linearly separate the data in the original space. SVMs can overcome this problem by mapping the input space to a higher dimensional one, where a linear separation may be feasible, using kernel functions. Out of many possible kernel functions, of special interest are those which satisfy Mercer's condition [8] and can be expressed as an inner product in the high-dimensional space. These kernel matrices $K(\cdot,\cdot)$ are called valid, provided they are continuous, symmetric and positive semi-definite. By applying the kernel, there is no need to explicitly map the data to the higher dimensional space. The most widely used kernel functions in SVMs are the Polynomial: $K(\mathbf{x}_i,\mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + c)^d$, the Gaussian: $K(\mathbf{x}_i,\mathbf{x}_j) = \exp^{-\frac{|\mathbf{x}_i-\mathbf{x}_j\|_2^2}{\sigma^2}}$ and Sigmoid: $K(\mathbf{x}_i,\mathbf{x}_j) = \tanh(u(\mathbf{x}_i \cdot \mathbf{x}_j + r))$, where $d \in \Re$, $c \in \Re$, $u \geq 0$ and $r \geq 0$ are kernel parameters and σ^2 is the expected or desired variance of the targeted training data.

A. Gilbert's Algorithm on SVMs

In [4], the SVM training is approached from a geometric view of the problem. The idea is based on the application of a nearest point algorithm, Gilbert's Algorithm [6], to the geometric expression of the SVM training. The algorithm is based on the concept that the problem of identifying the Support Vectors can be mapped to a problem of finding the point on a convex hull which lies closest to the origin. If Gilbert's Algorithm is applied to the secant convex hull $S = X - Y = \mathbf{x}_i - \mathbf{x}_j : y_i = 1, y_j = -1$ (Fig. 2), where X and Y are the classes of the training data set, then, according to [4], the geometric solution provided leads to the solution for the SVM training. Gilbert's Algorithm can locate the point of S that lies closest to the origin with linear recurring steps, without the need of explicitly constructing the secant.

Considering a single algorithmic iteration, starting from point $\mathbf{w}_{k-1}$, the algorithm locates the end point $\mathbf{g}^*(-\mathbf{w}_{k-1})$ of the line segment $[\mathbf{w}_{k-1}, \mathbf{g}^*(-\mathbf{w}_{k-1})]$, which belongs to the secant's perimeter. This point $\mathbf{s}_m$ is the one that maximizes the kernel $K(\mathbf{w}_{k-1}, \mathbf{s}_m)$. The next step is to locate the point $\mathbf{w}_k$ of the line segment which lies closest to the origin.

More details on the targeted algorithm can be found in [4]. As a conclusion, it should be mentioned that the kernel evaluations are the most time-consuming tasks in the algorithm. The step of computing the kernel functions and identifying the maximum kernels consumes more than 90% of the algorithm's iteration time, even for small datasets of 70K training data points. The number of these calculations is linearly dependent on the training data size N and input dimensionality D. Those tasks however are highly parallelizable. Moreover, the decisions for the algorithm's next step are not based on heuristics and can be performed simultaneously with the projection task. These characteristics make the algorithm a favourable target for a parallel processing machine.

III. Hardware Mappings

This section presents the details for the GPU and the FPGA mapping approaches for the SVM training based on Gilbert's Algorithm. The rationale of the architectures is driven by the need to exploit the parallelization potential of the algorithm, in order to speed-up the most time-consuming tasks of the algorithmic flow.

The most time-consuming task of the algorithm on a single Gilbert iteration is the evaluation of the kernel functions. However, all the training data N are projected along a line, which is designated by a single data point and the origin. Thus, this time consuming task is easily parallelized and we can decompose the original problem into smaller ones and solve all subproblems in parallel.

A. GPU Implementation

GPUs are Single Instruction Multiple Data (SIMD) computing devices. Parallelizable tasks are executed on the GPU as kernels, which can be considered as arrays of threads that operate on different sets of data. Threads are organized into blocks, and many blocks can be launched in a single kernel execution. Intra-block communication between threads is possible through the shared memory available in each GPU microprocessor, inter-block communication is however not possible. The shared memory has a size of 16KB per microprocessor and it provides very fast access. Access to the GPU on-chip global memory, however, presents large latency; thus, threads operate with a *load-process-store* model: chunks of data are loaded from the global memory to the microprocessors' shared memory, threads process them in parallel and finally, all threads store back the results to the global memory.

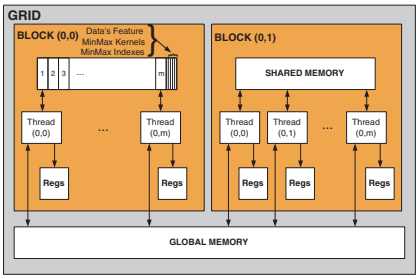

Fig. 3. GPU Kernel Programming Model for Gilbert's Algorithm.

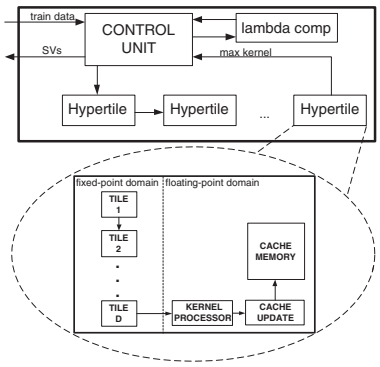

Fig. 4. FPGA architecture for Gilbert's Algorithm on the SVM Training.

Under CUDA, the GPU is treated as co-processor serving the host CPU. The serial algorithmic flow is executed on the CPU and parallelizable tasks are sent to the GPU. Each device maintains its own memory space, and Direct Memory Access provides fast communication between CPU and the graphics processor. The CPU is responsible for allocating, freeing, copying and storing data on the GPU memory.

As described earlier, the kernel evaluations for the projection are performed between one single data point and all the training data N. Thus, this task maps ideally to the GPU's hardware and its special characteristics. Moreover, the location of the next $\mathbf{g}^*(\mathbf{w}_{k-1})$ point requires the identification of the maximum kernel produced by each class, a task which is also easily parallelizable.

The implementation model of the kernel is illustrated in Fig. 3. Each thread uses a fragment of the available shared memory of the multiprocessor into which its block resides, in order to store a single dimension of a training point, as well as the maximum and minimum computed kernels and their corresponding data indices. It loads the features of its current data point sequentially and computes the kernel value. The result is then stored back to the global memory.

B. FPGA Architecture

This section focuses on the FPGA architecture for the SVM training based on Gilbert's Algorithm presented in [7]. Following the same idea of fragmenting the projection task into smaller ones, a highly scalable architecture is targeted, by employing a processing unit for the kernel evaluations. Multiple instantiations of this unit perform parallel computations and each solves a subproblem of the overall task. The proposed design of this module fully utilizes the device features and maximizes the parallelization. The targeted architecture is a homogeneous one, using the same bit precision P for all D dimensions of the training data. The chosen precision P needs to satisfy the largest dynamic range requirements among all dimensions.

Fig. 4 illustrates the proposed architecture. The hypertile is partitioned into fixed- and floating-point domains, in order to exploit the diversity of the dynamic range requirements between the inner products and the kernel functions. The modules which lie in the floating-point domain are fully pipelined, in order to maximize the throughput. The kernel functions include inner product calculations and thus, built-in blocks for the inner products are embedded in the kernels hardware implementation. These blocks introduce a delay which is directly proportional to D. In order to maximize the pipeline throughput, a scheme with many parallel tiles attached to the kernel pipeline is employed. Each tile performs the kernel evaluations for the update of a fragment of the overall problem. The fixed-point inner products are converted into IEEE754 single precision format before being fed into the kernel pipeline.

IV. Performance Comparison

In the following analysis, the processing times of the FPGA core and the GPU kernel are normalized, assuming one hypertile instance in the FPGA design and one GPU multiprocessor. The normalization was applied in order to allow for the generalization of the results to larger devices. The targeted device for the FPGA architecture was an Altera Stratix III EP3SE110-F780C2, while the GPU measurements regard the NVidia GeForce 8500GT chip.

Let us first examine the scenario where the training problem fits in the FPGA RAM blocks. The training data can then be loaded entirely in the device and be processed many times as the algorithm iterates. In this example, the size of the dataset was 315,000 and $P = 8$. Fig. 5 shows the relative speed up of the FPGA versus the GPU for various dimensionalities. For this comparison, the time to load the FPGA with training data was ignored. It can be observed that the hypertile's speed-up compared to the GPU's microprocessor increases linearly with the problem's dimensionality. This is expected, since for the case that $N \times D \times P \leq C_{FPGA}$, where C_{FPGA} is the memory capacity of the device's M9K blocks, the hypertile is processing the kernel values of D training points in parallel. In this case, the speed-up of the FPGA processing unit compared to the GPU microprocessor can be expressed as $K_{acc} = \frac{T_{GPU} \times f_{FPGA}}{N}$, where f_{FPGA} is the FPGA circuit's operating frequency and T_{GPU} is the measured execution time for a GPU with one microprocessor available.

Fig. 6 illustrates the normalized execution time of the FPGA design compared to the GPU for various dataset sizes, including the overhead time for the download of the training data to the FPGA. All datasets are four-dimensional. One important

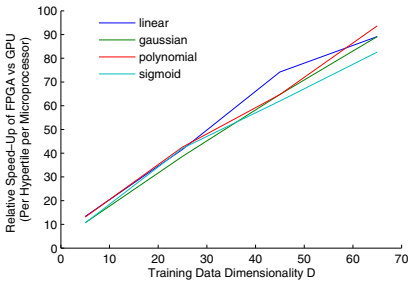

Fig. 5. Normalized speed-up comparison for $N \times D \times P \leq C_{FPGA}$.

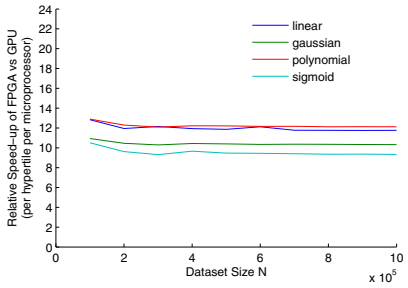

Fig. 6. Normalized speed-up comparison for $N \times D \times P \leq C_{FPGA}$.

observation is that, for fixed dimensionality, the FPGA speed-up remains constant as the dataset size N increases. This is expected, since the execution time of the FPGA and the GPU, as well as the load time are linear functions of N.

For the case of large scale classification problems that cannot fit in the internal FPGA memories, a different strategy should be followed. If no chunking techniques are applied, and the training set is processed entirely for every algorithm pass, it then is preferable to stream the data through the FPGA device rather than download and process them. The DDR2 RAM banks of the FPGA board are used to store the training dataset. Using the Terasic DE3 development board, with 8 DDR2 banks on a 64-bit data bus running under a 400MHz clock, the load time of the FPGA is approximately $\frac{N \times D \times P}{51{,}200} \mu sec$. Fig. 7 shows the relative speed-up of the FPGA device with one hypertile serving the training data stream, compared to the performance of a single GPU microprocessor. In this case, the speed-up of the FPGA processing unit compared to the GPU can be estimated as $K_{acc} = \frac{T_{GPU}}{T_{LOAD}}$, where T_{LOAD} is the overall time for the data transfer from the board RAMs to the FPGA. The whole design is fully pipelined.

The implementation results highlight several remarks, concerning the performance comparison of the two devices. First, it was shown that the speed-up of the FPGA processor increases with the dimensionality, compared to the GPU. On the other hand, high dimensionalities limit the maximum number of hypertile instances on the device and thus, the FPGA parallelization factor. Moreover, the execution times of both the FPGA and the GPU implementations are linear with the training set size N. In problems where the training set can fit in the FPGA RAM blocks, the hypertile offers higher

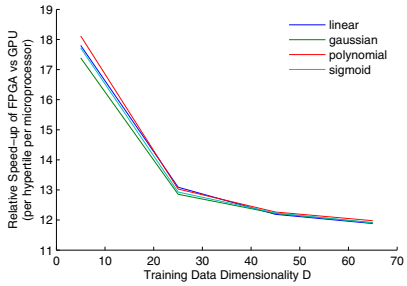

Fig. 7. Normalized speed-up comparison for $N \times D \times P > C_{FPGA}$.

acceleration factors, since the GPU requires the data transfer from the device global memory to the microprocessor shared memory. In large scale problems, Fig. 7 showed that streaming the data through the FPGA presents worse acceleration factors than the case where the data can fit in the internal FPGA memories (Fig. 6). Moreover, since only one hypertile can handle all the training stream, the parallelization factor of the FPGA device does not apply. Thus, it is preferable for an FPGA application to apply chunking techniques and split the dataset into several subsets. Otherwise, data streaming is the only feasible option and the performance worsens significantly.

V. Conclusion

This work presents a performance comparison between two of the most popular parallel-processing hardware devices on the application of Gilbert's Algorithm on the SVM training. The research was focused on the exploitation of the algorithm parallelization potential, in order to use the hardware resources offered by a GPU and an FPGA device efficiently, and accelerate the most time consuming algorithmic part. The performance comparison was normalized to the minimum discrete processing units of the devices and the results are expandable to problems with different constraints.

Acknowledgment

The authors greatfully acknowledge the funding support of EPSRC Grant Number EP/G00210X/1 and EP/C549481/1.

References

[1] V. N. Vapnik, *The Nature of Statistical Learning Theory*. Springer Verlag, 1995.

[2] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," Tech. Rep.

[3] T. Joachims, "Transductive inference for text classification using support vector machines," in *Proceedings of ICML-99, 16th International Conference on Machine Learning*, pp. 200–209.

[4] S. Martin, "Training support vector machines using gilbert's algorithm," in *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*. IEEE Computer Society, pp. 306–313.

[5] NVidia, *NVIDIA CUDA Compute Unified Device Architecture, Programming Guide*, http://www.nvidia.co.uk/cuda, 2008.

[6] E. G. Gilbert, "An iterative procedure for computing the minimum of a quadratic form on a convex set," *SIAM Journal on Control*, vol. 4, no. 1, pp. 61–80, 1966.

[7] M. Papadonikolakis and C.-S. Bouganis, "A scalable fpga architecture for non-linear svm training," in *ICFPT 2008*, pp. 337–340.

[8] J. Mercer, "Functions of positive and negative type and their connection with the theory of integral equations," *Philos. Trans. Roy. Soc. London*, 1909. [Online]. Available: http://en.wikipedia.org/wiki/Mercer's_theorem