A Scalable FPGA Architecture for Non-Linear SVM Training

Markos Papadonikolakis and Christos-Savvas Bouganis
Department of Electrical and Electronic Engineering
Imperial College London
Exhibition Road, South Kensington, London, SW7 2AZ, UK
Email: markos.papadonikolakis07@imperial.ac.uk, christos-savvas.bouganis@imperial.ac.uk

Abstract

Support Vector Machines (SVMs) is a popular supervised learning method, providing state-of-the-art accuracy in various classification tasks. However, SVM training is a time-consuming task for large-scale problems. This paper proposes a scalable FPGA architecture which targets a geometric approach to SVM training based on Gilbert's algorithm using kernel functions. The architecture is partitioned into floating-point and fixed-point domains in order to efficiently exploit the FPGA's available resources for the acceleration of the non-linear SVM training. Implementation results present a speed-up factor up to three orders of magnitude of the most computational expensive part of the algorithm compared to the algorithm's software implementation.

1. Introduction

In a classification problem, the objective of SVM [7] training task is the construction of a linear hyperplane which will provide maximum separation between the classes. The SVM training problem focuses on identifying the data instances that lie closest to the separating hyperplane, thus designating its direction and position. These data points are called Support Vectors (SVs).

In most real world applications, the data are not linearly separable, i.e. there is no feasible linear separation between the two classes. In such cases, SVMs provide the flexibility to map the input data to a higher-dimensional feature space, where a linear separation can be obtained. This becomes feasible by employing kernel functions. In the general case, the optimization problem can be formulated as follows:

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

where $K(\cdot,\cdot)$ denotes a kernel function, $\mathbf{x}_i$ is the training data, y_i is the class label taking values -1,1, $\mathbf{w}$ is the perpendicular vector to the hyperplane's direction, b is the offset to the origin and N is the training set's size. The most important characteristic of non-linear SVM is that by using kernel functions that satisfy Mercer's condition, the classification can be performed in the higher-dimensional feature space without explicitly computing it [6]. The SVM training problem (1) is a Quadratic Programming (QP) problem, which is computational expensive for large-scale data sets. Thus, several training algorithms have been proposed, such as decomposition methods (SMO [5]) or geometric-based approaches [3].

A first approach which was focused on linear SVMs can be found in [4], where a scalable FPGA architecture of the Gilbert's algorithm [1] for linear SVM was proposed. In this work, the research is focused on the non-linear case of SVMs, leading to the re-addressing of the initial architecture in [4]. The wide applicability of non-linear SVMs to real world classification problems and the need for acceleration are strong motivations to design a scalable FPGA architecture for the non-linear SVM training problem. The original contributions of this work are: (i) the introduction of the hardware-mapped kernel functions which are most widely used in SVMs, (ii) an efficient implementation of a processing tile, with respect to the problem's characteristics and the available FPGA resources and (iii) the kernel modules' embodiment into a new scalable FPGA architecture. The rest of the paper is organized as follows: Section 2 presents a brief overview of kernel functions and a description of Gilbert's Algorithm for non-linear SVM training. Section 3 focuses on the proposed FPGA architecture, Section 4 provides the implementation results and the paper concludes in Section 5.

2. Non-Linear SVM Training

2.1. Kernel Functions

In many real-world classification problems, it is often not feasible to linearly separate the data in the original space.

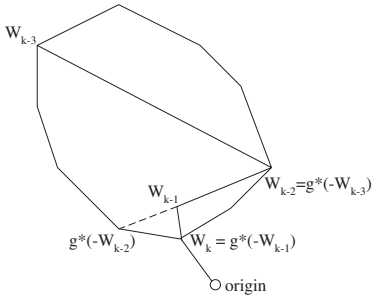

Figure 1. Gilberts Algorithm iterations on SVM secant hull.

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.

Nevertheless, the explicit construction of this higher dimensional space mapping for all data points is a computational expensive task, especially in large-scale problems. SVMs can avoid this by employing kernel functions $K(\mathbf{x}_i, \mathbf{x}_j)$ in the optimization problem (1). Out of many possible kernel functions, of special interest are those which satisfy Mercer's condition and can be expressed as an inner product in the high-dimensional space. This means that if Ω is the input space, there is a space H and a mapping $\phi : \Omega \rightarrow H$ such that $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_H$. A kernel matrix $K(\cdot, \cdot)$ which satisfies Mercer's condition is called valid and it is continuous, symmetric and positive semi-definite. By applying the kernel trick to the SVM, there is no need to explicitly map the data to the higher dimensional space H. 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 Radial basis function: $K(\mathbf{x}_i, \mathbf{x}_j) = \exp^{-\gamma \cdot \|\mathbf{x}_i - \mathbf{x}_j\|^2}$ and Sigmoid: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(u(\mathbf{x}_i \cdot \mathbf{x}_j + r))$, where $d \in \Re$, $c \in \Re$, $\gamma \in \Re$, $u \geq 0$ and $r \geq 0$ are kernel parameters. The Gaussian kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \exp^{-\frac{|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}}$, widely used in SVMs, is a special case of the RBF kernel for $\gamma = 1/\sigma^2$, where σ^2 is the expected or desired variance of the targeted training data.

2.2. Gilbert's Algorithm Overview

Let us denote by X and Y the convex hulls of each class of the training data set. The secant convex hull is defined as $S = X - Y = \mathbf{x}_i - \mathbf{x}_j : y_i = 1, y_j = -1$. According to [3], the closest point $\mathbf{w}_k$ to the origin is the solution for the SVM training problem (Fig. 1). This point is provided by the application of Gilbert's Algorithm on the secant convex hull S. Gilbert's Algorithm is an iterative process which locates the point of a secant hull which lies closest to the origin [1]. Considering an iteration of the algorithm, 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 $K(\mathbf{w}_{k-1}, \mathbf{s}_m)$ and it can be evaluated as $\mathbf{g}^*(\mathbf{w}_{k-1}) = \mathbf{g}^*_X(\mathbf{w}_{k-1}) - \mathbf{g}^*_Y(\mathbf{w}_{k-1})$. The next step is to locate the point $\mathbf{w}_k$ of the line segment which lies closest to the origin. A variable λ geometrically expresses the position of the next point $\mathbf{w}_k$ on the line segment. The algorithm abundantly iterates over the secant and computes subsequent averages of $\mathbf{w}_k$ points. The algorithm terminates when the angle between the latter two averages $\overline{\mathbf{m}}$ is smaller than a convergence tolerance e, which is user-specified.

Regarding the implementation of the algorithm, one interesting characteristic is that the algorithm can be considerably accelerated by storing the kernels $K(\mathbf{w}_{k-1}, \mathbf{x}_i)$ and $K(\mathbf{w}_{k-1}, \mathbf{x}_j)$ for each iteration in a cache. This allows $\mathbf{g}^*$ to be located by just identifying the maximum kernels stored in the cache. The cache values are updated on each iteration, according to the new value of λ.

More detailed description of the algorithm can be found in [3] and [4]. Concluding, it can be observed that the evaluations of kernels (or inner products in the linear case) are the most time-consuming tasks in an algorithm's iteration. The number of these calculations are linearly dependent of the training data size and input's dimensionality.

3. FPGA Architecture

3.1. Top-Level Architecture

The algorithm's analysis highlighted several important issues, which should be taken under consideration when designing the hardware architecture of the SVM training algorithm. In multi-dimensional large-scale problems, the kernel evaluations govern the algorithm execution time. However, in each iteration of the algorithm, the kernel evaluations are performed between one single point and all the training data N. Thus, we target a high scalable architecture, by employing a processing unit for the kernel evaluations. Multiple instantiations of this processing unit, which performs parallel computations and solves a subproblem of the overall kernel evaluations' task, significantly speed-up the algorithm's iteration. The efficient design of this module, with respect to the available resources of a modern FPGA, fully utilizes the device's features and maximizes the parallelization factor.

The top-level architecture is illustrated in Fig. 2. Each hypertile focuses on a fragment of the overall problem and communicates with its neighbors to transmit data information. It performs multiple parallel kernel evaluations and stores them in each own memory. Moreover, the hypertile propagates the maximum evaluated kernel among with the corresponding training data's address, in order to identify

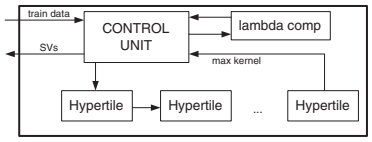

Figure 2. Top level architecture.

the point $\mathbf{g}^*(\mathbf{w}_{k-1})$ for the next algorithm's iteration. The control unit interfaces the external bus and is responsible for the propagation of universal control signals between hypertiles, the execution of algorithm's iteration and the check of termination criteria.

3.2. Precision Analysis

Considering the hardware implementation of kernel functions, the representation of their dynamic range should be first addressed. The kernel functions include inner product calculations and thus, built-in blocks for the inner products are embedded in the kernels' hardware implementations. However, the high-dimensionality of a kernel and its increased dynamic range requires a different approach than the inner product's case. For example, for training data with 8-bits precision and 25 dimensions, an application which uses a polynomial kernel with $d = 5$ and $c = 0$ would need $5 \cdot (8 \cdot 2 + \log_2(25)) = 105$ bits for representation of the kernel output. Thus, floating-point precision is essential for the hardware implementation of kernel functions.

Nevertheless, the representation of the dynamic range of an inner product between 2 data points with P-bits precision and D dimensions would only require $2 \cdot P + \lceil \log_2(D) \rceil$ bits. Due to these different requirements, the partitioning of several modules into fixed-point and floating-point domains is necessary. The following analysis concerns a homogenously designed architecture, where the fixed-point domain uses equal bit-width P for all D dimensions/training data point and P satisfies the largest dynamic range among D dimensions.

3.3. Hypertile Analysis

In order to maximize the computational speed and design an efficient in performance terms circuit for the kernels evaluations, a pipeline technique is applied. However, the circuit to implement computations such as the inner products or the squared norms, which appear in the kernel functions, introduces a delay which is directly proportional to D. In order to maximize the pipeline's performance, a scheme with many parallel tiles attached to the kernel pipeline is employed. Moreover, allocating a common kernel processor among many inner product tiles is a more area-efficient solution than a scheme with one kernel per inner product,

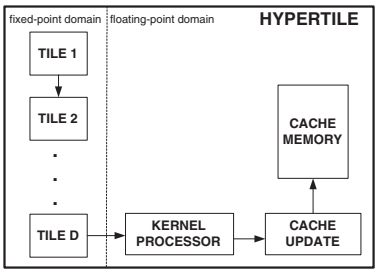

Figure 3. Hypertile's architecture.

since the floating-point operations are area expensive.

The hypertile's structure is shown in Fig. 3. Each tile performs the kernel evaluations for the update of a fragment of the overall problem. D tiles for the inner products feed the kernel processor and the cache update module in parallel with $1/D$ data rate. Fixed-point precision is kept for the tiles' computations, while the kernel pipeline and the module for the update of the cache memory lie in the floating-point domain. The fixed-point inner products are interpreted into IEEE754 single precision format before feed the kernel input.

The proposed architecture targets three kernel types, the polynomial, the gaussian and the sigmoid kernel. For the sigmoid kernel, the hyperbolic tangent function is expressed and implemented as $\tanh(x) = \frac{\exp^{2x}-1}{\exp^{2x}+1}$. The cache update module is fully-pipelined, operating under floating-point precision on the kernel evaluations. The kernel values are stored in a large memory, which is used for all tiles inside a hypertile. This scheme that employs one large cache memory for all tiles in a hypertile provides easier access and better routing than maintaining many small memories per tile.

4. Implementation Results

The targeted device for the architecture was Altera's Stratix III EP3SE110-F780C2. This device was chosen because of its large number of available DSPs, as well as the gradation of its memory capacitance. The design was captured in VHDL and the Altera Floating-Point Compiler [2] was used for the design of the floating-point units of the architecture. The M9Ks were used for the instantiation of tiles' memories, while the cache memory is allocated in M144K blocks.

Table 1 presents the implementation results for the three targeted kernel processors, regarding the area and DSP usage, the achieved operating frequency and the processors' overall latency. The area usage is different for each kernel function implementation and is dependent on the data's bit precision and dimensionality. The hypertile's area cost can be evaluated as $A_{HYPER} = D \cdot A_{TILE} + A_{KER} + A_{CUPD}$,

Table 1. Kernel Processors' Implementation Results.

KERNEL PROCESSOR	Area (ALUTs/REGs)	DSPs (18-bits)	Frequency (MHz)	Latency (clock cycles)
Polynomial	1385/2963	16	338	50
Gaussian	552/782	18	255	23
Sigmoid	1936/2836	28	260	61

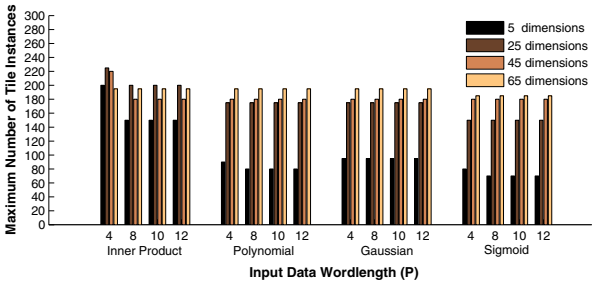

Figure 4. Number of tiles instantiations per D, P and kernel function.

where A_{HYPER}, A_{TILE}, A_{KER} and A_{CUPD} is the resource utilization for the hypertile, the tile, the kernel processor and the cache update module respectively. The DSP utilization of a tile is constant and equals to 4 DSP 18-bit elements and remains invariable while $P < 36$ and the inner product's range $2 \cdot P + \log_2(D) \leq 72$. The hypertile's operating frequency drops with the increase of data's precision P and dimensionality D and ranges between 200-330MHz, depending on the targeted kernel and D.

Fig. 4 illustrates the maximum number of tile instantiations on the device. Each hypertile contains D tiles. For low dimensions, the available area and DSP resources add equivalent constraints. For larger D, the DSP constraint becomes dominant. The tile instantiations increase with D, since the number of tiles per hypertile equals to D.

The architecture's achieved throughput is compared to the algorithm's Matlab implementation of [3] on a PC with a 3GHz Intel CORE 2 DUO and 2GB of RAM. If N equals to the number of training data that can fit in a hypertile and Ts is the measured CPU time consumed by the algorithm for the evaluation of the kernels on these data, the acceleration factor will be: $K_{acc} = \frac{Ts \cdot H \cdot FRE}{D + L_{KER} + L_{CUPD} + 7 + N}$, where H is the number of hypertiles that can be fitted in the device, FRE is the circuit's operating frequency, L_{KER} and L_{CUPD} are the kernel processor's and the cache update module's latency. The denominator is the overall number of clock cycles required for the hypertile to process a whole pass of its subproblem. For $D = [5, 65]$, the achieved speed-up offered of the FPGA architecture ranges from 780-4,000 for the non-linear SVM. In the linear case, the acceleration factor varies from 1,400-2,700. It should be noted that the speed-up factors refer to the algorithm's part for the cache update and not to the whole algorithm's iteration time. However, for the SVM training of a small set with $N = 20,000$ and $D = 45$, the cache update part requires more than 94% of the time per algorithm's iteration. This percentage reaches up to $\sim 98\%$ for datasets of millions of training data. An important remark is that the reported acceleration factors K_{acc} are computed with one M9K block per tile. Exhausting all the M9K blocks and allocating them to tiles would increase the acceleration factor.

4.1. Conclusion

In this paper, a scalable FPGA architecture for the SVM training based on Gilbert's Algorithm has been presented. A partitioning of circuit's modules into fixed-point and floating-point domains was applied, leading to a flexible architecture that can be applied to many different problems. The employed hypertile scheme with D parallel tiles feeding each kernel pipeline maximizes the circuit's performance. The implementation results showed a speed-up of up to three orders of magnitude for the algorithm's inner loop; important optimization, as it consumes a significantly large fraction of the algorithm's execution time.

Future work will focus on the design of an heterogeneous FPGA architecture. By exploiting the diversity of dimensions' dynamic ranges, the resource usage can be reduced and result in even higher parallelization than the current approach. Moreover, chunking methods will be applied to address large-scale SVM problems, when the training set cannot fit entirely in the FPGA.

References

[1] E. G. Gilbert. An iterative procedure for computing the minimum of a quadratic form on a convex set. *SIAM Journal on Control*, 4(1):61–80, 1966.

[2] M. Langhammer. Floating point datapath synthesis for fpgas. In *FPL'08*, 2008.

[3] S. Martin. Training support vector machines using gilbert's algorithm. In *ICDM'05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 306–313, Washington, DC, USA, 2005. IEEE Computer Society.

[4] M. Papadonikolakis and C.-S. Bouganis. Efficient fpga mapping of gilbert's algorithm for svm training on large-scale classification problems. In *FPL'08*, 2008.

[5] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, 1998.

[6] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

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