EFFICIENT FPGA MAPPING OF GILBERT'S ALGORITHM FOR SVM TRAINING ON LARGE-SCALE CLASSIFICATION PROBLEMS

Markos Papadonikolakis, 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, ccb98@imperial.ac.uk

ABSTRACT

Support Vector Machines (SVMs) are an effective, adaptable and widely used method for supervised classification. However, training an SVM classifier on large-scale problems is proven to be a very time-consuming task for software implementations. This paper presents a scalable high-performance FPGA architecture of Gilbert's Algorithm on SVM, which maximally utilizes the features of an FPGA device to accelerate the SVM training task for large-scale problems. Initial comparisons of the proposed architecture to the software approach of the algorithm show a speed-up factor range of three orders of magnitude for the SVM training time, regarding a wide range of data's characteristics.

1. INTRODUCTION

In Machine Learning, classification is the task of matching inputs to one predefined category. In the case of a supervised problem, the machine needs to know the training data's classes a priori. Among supervised learning methods, the most popular are Neural Networks, Linear Discriminant Analysis, Support Vector Machines, Gaussian Mixture Models and Decision Trees.

Support vector machines [1] have been proven to be a very effective method for supervised classification with a wide range of applicability, offering state-of-the-art solutions for numerous applications. SVMs are efficiently used in classification tasks such as face detection and recognition [2], time series forecasting [3], medical applications and more.

Much research has focused on the training phase of SVMs, whose task is to train a machine using a set of training data. The SVM training problem can be formulated as a Quadratic Programming (QP) Problem. Due to the complexity of the QP problem for large data sets, decomposition methods have been proposed, such as SMO [4], $\text{SVM}^{\text{LIGHT}}$ [5], LIBSVM [6] and SVM^{PERF} [7]. Recently, a geometric-based approach to the SVM training problem has been introduced [8], [9] and [10], which is based on Gilbert's Algorithm [11].

Most well known algorithms for SVM training are superlinear to the size of the training data set. Thus, SVM training is considered as a computational expensive task for applications that require large training data sets. For instance, the training time for tasks with training sizes of millions of data points, such as the Covertype data set or Forest data set example, is in the order of days, for a single SVM machine ([12], [13]). Moreover, in online training applications with real-time performance constraints, the use of dedicated hardware resources could significantly accelerate the SVM training, especially when considering large-scale problems. However, to the best of authors' knowledge, marginal research work has been focused on hardware solutions to reduce SVM training time.

In this work, a geometric-based approach that employs Gilbert's Algorithm to the SVM training is chosen, due to its plainness and its high parallelization potentials. An FPGA device is targeted because of its runtime reconfigurability, which allows the design to adapt to different types of input data. This paper proposes a scalable architecture which focuses on maximizing the utilization of FPGA resources and takes advantage of the special features of a modern FPGA device in order to speed-up the SVM online training for large scale classification problems.

The rest of this paper is organized as follows: Section 2 presents a brief description of SVM decomposition algorithms, focuses on the geometric approach of Gilbert's algorithm and provides an insight to the reasons why it was chosen as targeted algorithm. Section 3 includes the detailed description of the proposed FPGA architecture, while Section 4 presents the results for various implementations. Finally, the paper concludes in section 5.

2. SVM TRAINING ALGORITHMS

2.1. Overview

The objective of SVMs is to construct a seperating hyperplane $\mathbf{w} \cdot \mathbf{x} - b = 0$ to attain maximum separation between

978-1-4244-1961-6/08/$25.00 ©2008 IEEE.

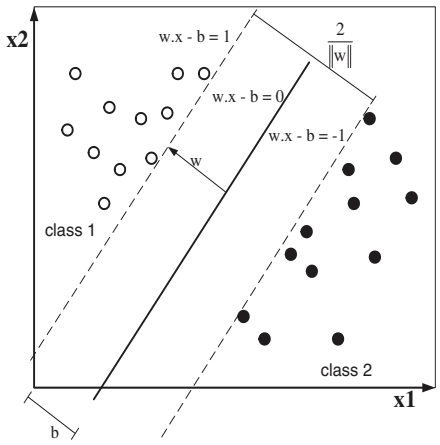

Fig. 1. SVM seperating hyperplane.

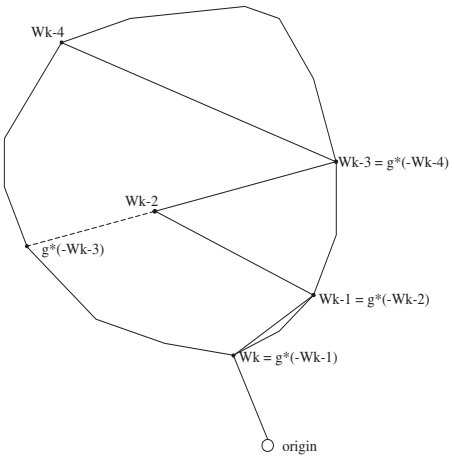

Fig. 2. Iteration steps of Gilbert's Algorithm.

the classes, as shown in Fig. 1. The classes' hyperplanes are parallel to the separating one, lying on each side of it. The Euclidean Distance between the two hyperplanes equals to 2/||**w**||, thus the objective of SVMs is to maximize the distance between the classes' hyperplanes or, in other words, to minimize $\|\mathbf{w}\|$:

$$\begin{aligned} &\min \tfrac{1}{2}\|\mathbf{w}\|^2, \\ &s.t. y_i(\mathbf{w}\cdot\mathbf{x}_i - b) \geq 1, 1 \leq i \leq n. \end{aligned} \tag{1}$$

, where $(\cdot)$ denotes an inner product, $\mathbf{x}_i$ is the training data, y_i is the belonging class taking values -1,1, **w** is the perpendicular vector to the hyperplane's direction and b is the offset to the origin. We are focusing on the training phase of the algorithm, whose task is the identification of those training samples that lie closest to the margin and determine its direction; these samples are called Support Vectors. In the case where the data are not linearly separable, SVMs can map the input feature space to a higher dimensional one, by replacing every dot product $\mathbf{x}\cdot\mathbf{x}'$ in the algorithm by a kernel function $K(\mathbf{x},\mathbf{x}')$. In this non-linear SVM case, the problem is transferred to a higher dimensional space, where linear separation can be obtained.

Solving the SVM training problem by using QP techniques is a demanding and computational expensive task, especially for large high-dimensional datasets. Thus, many algorithms have been proposed to decompose this large QP problem into smaller ones. Sequential Minimum Optimization (SMO) [4] solves the QP problem analytically and it applies to linear and non-linear SVM. In each iteration, it chooses two data points from the training set as the working set and performs the optimization. Similar algorithms like SVM$^{\text{LIGHT}}$ [5] and LIBSVM [6] are based on the same idea of the QP problem decomposition. Their main difference is the way the working set is chosen on each iteration. These algorithms scale superlinearly with the size of training set. SVM$^{\text{PERF}}$ [7] uses a Cutting Plane algorithm for linear SVM training that succeeds in scaling linearly with the number of the samples in the training set. However, the linear behavior regards only linear SVMs; in the non-linear case, the performance of SVMperf worsens by a factor n, equal to the size of training data.

2.2. Gilbert's Algorithm on SVM

Recently, various research works [8], [9], [10] approach the SVM training from a geometric view of the problem. These proposed methods are based on the application of a nearest point algorithm, Gilbert's Algorithm [11], to the geometric expression of the SVM training problem. According to [8], the solution to the SVM problem is the point $\mathbf{w}_k$, which belongs to the secant convex hull's perimeter and is closest to the origin. Gilbert's Algorithm locates the point of a convex hull closest to the origin with recurring linear steps.

In [8] and [10], Gilbert's Algorithm is applied on the secant convex hull S = X - Y = $\mathbf{x}_i$ - $\mathbf{x}_j$: y_i = 1, y_j = -1, where X and Y are the convex hulls of each class of training data, as shown in Fig. 2. It should be noted that the algorithm does not require the explicit construction of the secant.

The algorithm's iteration loop is illustrated in Fig. 3. Starting from a random point w_{k-1}, the algorithm locates the point $\mathbf{g}^*(-\mathbf{w}_{k-1})$, whose projection on the direction of $-\mathbf{w}_{k-1}$ is the closest to the origin; this point will lie on the secant's perimeter [11]. Thus, $\mathbf{g}^*(-\mathbf{w}_{k-1})$ is the point of S that maximizes the inner product with $\mathbf{w}_{k-1}$. This value can be computed by finding $\mathbf{g}^*_X$ and $\mathbf{g}^*_Y$, which are the points $\mathbf{x}_i$ and $\mathbf{x}_j$ of classes X and Y respectively that maximize the inner products $-\mathbf{w}_{k-1}\cdot\mathbf{x}_i$ and $\mathbf{w}_{k-1}\cdot\mathbf{x}_j$:

$$\mathbf{g}^*(-\mathbf{w}_{k-1}) = \mathbf{g}^*_X(-\mathbf{w}_{k-1}) - \mathbf{g}^*_Y(\mathbf{w}_{k-1}) \tag{2}$$

The next step is to locate the next $\mathbf{w}_k$ as in (3), which is the point on the line segment $[\mathbf{w}_{k-1}, \mathbf{g}^*(-\mathbf{w}_{k-1})]$ closest to the

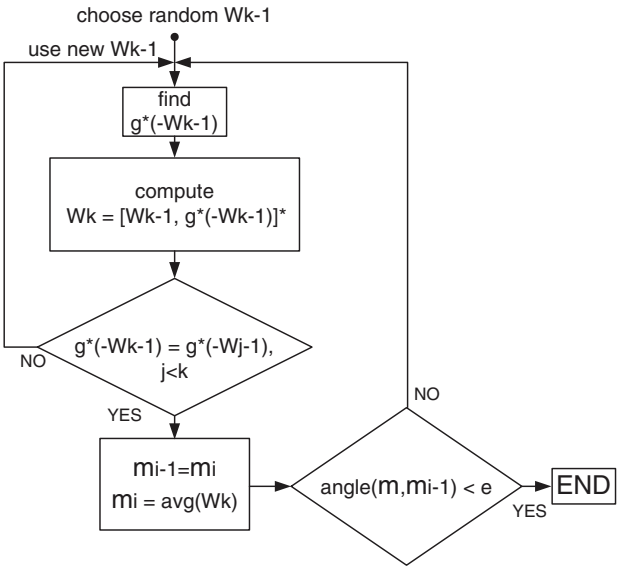

Fig. 3. Gilbert's Algorithm flow.

origin, which may not be part of the secant.

$$\mathbf{w}_k = \begin{cases} \mathbf{w}_{k-1} & \text{, if } top \leq 0, \\ \mathbf{g}^*(-\mathbf{w}_{k-1}) & \text{, if } bot \leq top, \\ \mathbf{w}_{k-1} + \lambda(\mathbf{g}^*(-\mathbf{w}_{k-1}) - \mathbf{w}_{k-1}) & \text{, else} \end{cases} \tag{3}$$

, where:

$$\begin{aligned} top &= -\mathbf{w}_{k-1} \cdot (\mathbf{g}^*(-\mathbf{w}_{k-1}) - \mathbf{w}_{k-1}) \\ bot &= \|\mathbf{g}^*(-\mathbf{w}_{k-1}) - \mathbf{w}_{k-1}\|^2 \\ \lambda &= \tfrac{top}{bot} \end{aligned} \tag{4}$$

This algorithm's loop iterates numerous times over the convex hull's points. The average $\overline{\mathbf{m}}_i$ of $\mathbf{w}_k$ points is calculated and the algorithm terminates if the angle between the latest two averages is smaller than e, which is the algorithm's convergence tolerance and is user specified.

A cache scheme is employed to accelerate the algorithm. The inner products $-\mathbf{w}_{k-1} \cdot \mathbf{x}_i$ and $w_{k-1} \cdot \mathbf{x}_j$ are stored in a cache and hence, $\mathbf{g}^*$ of the next iteration is easily calculated by the maximum values stored in the cache. The values of the inner product cache are updated according to the value of λ in equations (4), to reflect the new value of w_k in equation (3), as in (5).

$$cache = (1 - \lambda)cache + \lambda(\mathbf{w}_{k-1} \cdot \mathbf{x}) \tag{5}$$

Concluding the above analysis, several interesting characteristics of the algorithm can be remarked. The iteration loop of the algorithm and the termination criteria are easily mapped in hardware, while there are no complicated decisions for the point of the next iteration, leading to a simple to implement control flow. The dominant mathematical functions of the algorithm are the inner products and only one

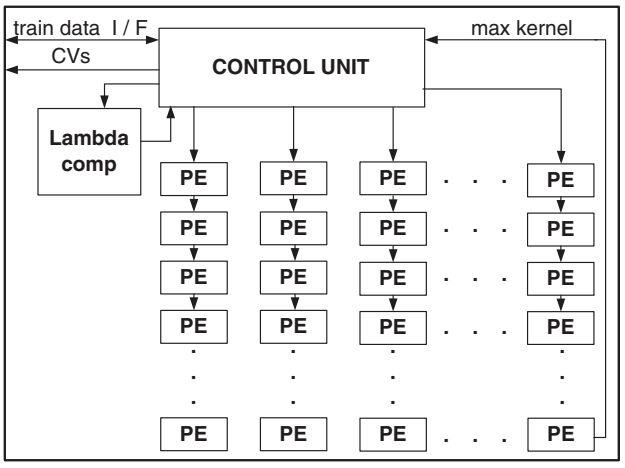

Fig. 4. Top Architecture.

division per iteration step is required. Therefore, the algorithm can be implemented using fixed-point precision. The most computational demanding step of the loop is the evaluation of the cache data points, which scales linearly with the training data size and the input's dimensionality. However, the inner products evaluations are independent and many parallelization techniques can be applied to speed-up the algorithm's iteration loop.

3. FPGA MAPPING

3.1. Architecture's Description and Considerations

As already mentioned, the most time-consuming task of the loop in Gilbert's Algorithm is the update of its inner product cache. Hence, a significant step to accelerate the algorithm is to introduce a Processing Element (PE) module, which could segment the cache update problem into smaller ones. Implementing the PE efficiently, with respect to the available resources of a specific FPGA device, contributes in better utilization of the hardware resources and more PE instantiations. Thus, the acceleration of the algorithm is maximized. This case study of the architecture refers to a DSP-oriented FPGA. Initial exploration of the problem showed that the main critical factor is the number of slices per PE; thus, keeping PE's area small will result in a larger number of PEs fitting in the device, since DSPs and block RAMs available resources will add softer constraints.

The proposed architecture is shown in Fig. 4. First, the training data are downloaded to the FPGA, assuming that the training set fits into the device's block RAMs. In the case where the problem is too large, chunking or clustering methods can be applied and solve the subsets of the problem iteratively. Then, each PE updates a segment of the inner product cache, while at the same time evaluates the maximum inner product of its stored data. Each PE compares

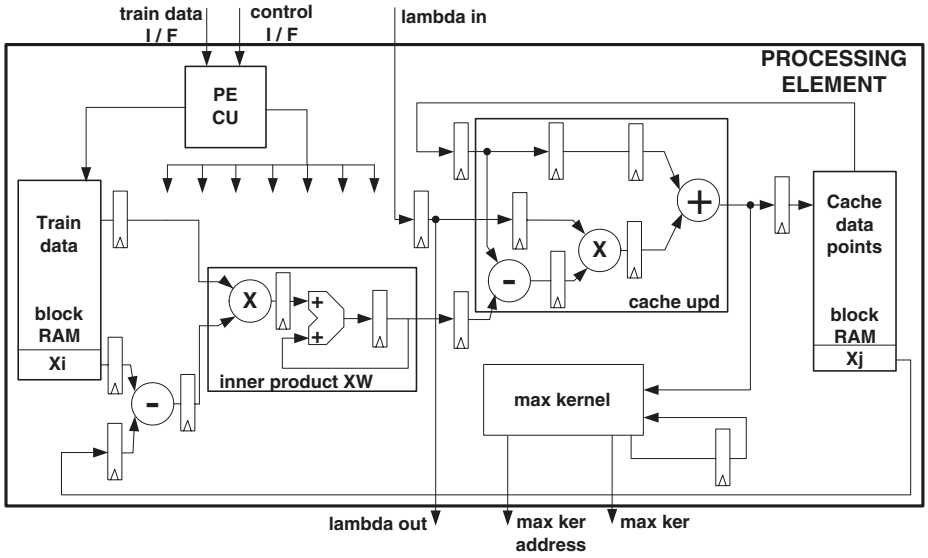

Fig. 5. Processing Element Architecture.

its max inner product with the inner product of previous PE and propagates the result to the next one, in order to find $\mathbf{g}^*(-\mathbf{w}_{k-1})$. The λ value is calculated by a special module parallel to the cache updates and is propagated through PEs to reduce fanout and routing delays. The control unit is responsible for the bus interface, the algorithm's iterations steps, as well as the storage of $\overline{\mathbf{m}}_i$ average values and the checking of termination criteria.

To complete the investigation of the proposed architecture, the rationale of some design choices should be remarked. In order to propose an efficient design, which could take advantage of the special features of a modern FPGA device, several issues needed to be addressed. The first decision to make is how the machine accesses the training data. Downloading segments of training data into block RAMs can offer significant increase of the device's throughput and take full advantage of the FPGA's available bandwith, by utilizing all block RAMs in parallel. In addition, the memory bus is relieved from sparse or sequential accesses to an external RAM while the algorithm's iteration are performed. Moreover, the allocation of data in the block RAMs is an important issue; the FIFO scheme is the simplest to implement and offers the easiest access to data, since one extra bit to denote the class of data is only needed. The propagation of variables like λ and maximum kernels was kept single directional through the PEs in order not to add hard placement constraints for the element's instances.

3.2. Processing Element Analysis

The Processing Element of the design is shown in Fig. 5. The PE's data path is fully pipelined, except the two inner product modules that operate in parallel. This inner products scheme has a latency of m clock cycles, where m is the dimensionality of the data. The proposed PE architecture follows a distributed control approach, in order to reduce routing delays of signals among PEs and keep fanout low. Each PE has its own control unit, which is responsible of the control signals for the pipeline as well as the memory control. The latency of PE is dependent on the data's dimensionality and the number of DSPs used for the cache update multiplier.

The training data are stored in block RAM, as well as the inner products between each data point and $\mathbf{w}_{k-1}$. Moreover, the current $\mathbf{w}_{k-1}$ is kept in the two block RAMs instead of using registers to minimize PE's area. The computation of the inner product $\mathbf{w}_{k-1} \cdot \mathbf{x}$ is implemented with a sustracter and one DSP. The function (5) for the cache update is transformed and implemented as in (6).

$$cache = cache + \lambda(\mathbf{w}_{k-1} \cdot \mathbf{x} - cache) \qquad (6)$$

The multiplier for the cache update is implemented with DSP blocks instead of using LUTs or Hybrid Multiplier, leading to small area utilization for the PE.

3.3. Lambda Computation Module

The module for the λ variable evaluation is shown in Fig. 6. *Lambda computation module* takes as inputs $\mathbf{w}_k$, $\mathbf{g}^*(-\mathbf{w}_{k-1})$ and their inner product, which has been already computed and stored in the inner product cache memory. All inner products are implemented in DSPs. The divider was implemented for high frequency rather than high throughput, since the computation of λ is performed in parallel to the

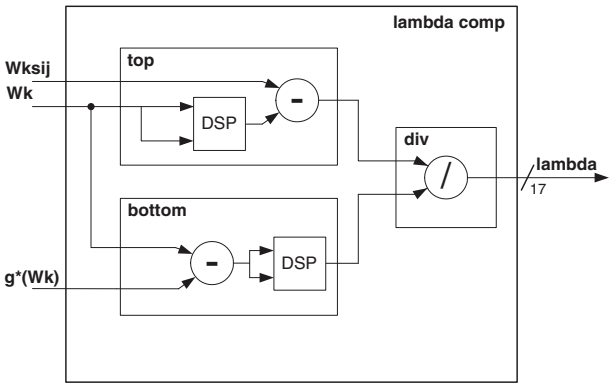

Fig. 6. Lambda computation module.

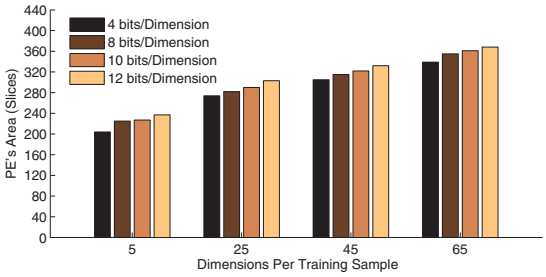

Fig. 7. PE's Area.

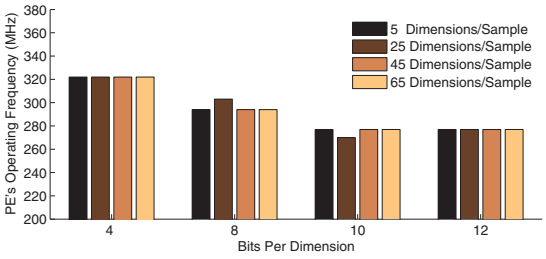

Fig. 8. PE's Operating Frequency.

cache updates and the overall delay of this module is not critical to the algorithm's performance. The selected precision for λ is 18 bits for the fractional part, since λ is only used for the cache update when: $\lambda \in (0, 1)$. This precision is considered sufficient in order to avoid abnormal terminations or useless iterations over the same points on the same line segment $[\mathbf{w}_{k-1}, \mathbf{g}^*(-\mathbf{w}_{k-1})]$.

4. IMPLEMENTATION RESULTS

The proposed architecture was designed for a wide input bit range and various data's dimensionalities. The targeted device was the Xilinx Virtex-4 XCV4VSX55-FF1148-12, since it features a large number of DSP blocks (512), while keeping a good ratio between DSPs and block RAMs (1.6:1). The memory usage is 2 block RAMs/PE and the DSP usage is 3 DSPs/PE for 4 bits/dimension and 4 DSPs/PE for 8, 10 and 12 bits/dimension. Since the RAM and DSP requirements per PE are kept low, the number of required slices adds the most critical constraint and governs the potential number of PE instantiations in the design. It should be mentioned that all bit-width refer to the worst-case, as if all data's dimensions have equal bit-range requirements.

The scaling of PE's area with dimensionality and bit-width is illustrated in Fig. 7. It can be concluded that PE's area scales linearly with data's bit-width and dimensionality. By fitting a linear function to the results, an expression of required slices to data's bit-width and dimensionality can be obtained (7), which allows the estimation of PE's area for any other possible configuration.

$$\text{Num. of slices} = 2.1 \cdot \text{Dims} + 3.6 \cdot \text{Bits} + 191 \quad (7)$$

Fig. 8 shows the scaling of PE's operating frequency with the increase of data's bit-width and dimensionality. The circuit's maximum operating frequency decays with the bit-width increase, although routing delays also affect this relation. However, it can be seen that PE's performance remains satisfactorily high for various bit-widths. Moreover, Table 1 includes results for the lambda computation module for various implementations. The critical path of the architecture is located in the division for the evaluation of λ. Nevertheless, this module's performance is roughly the same to the PE's and does not impose any extra constraint to the overall design. The module's latency increases with the dimensionality and the precision, although its effect on the iteration's overall latency is negligible, since these operations are performed in parallel to the inner products evaluations.

Fig. 9 illustrates the maximum possible number of PE instantiations in the device, which is inversely proportional to the PE's occupied area. The number of PE's instantiations is a direct factor of the hardware's speed-up compared to a software implementation of the algorithm. Comparing to the algorithm's Matlab implementation [10] on a PC with a 3GHz Intel CORE 2 DUO and 2GB of RAM, the proposed design presents a speed-up range of 3,000-6,000 on the algorithm's iteration. This range regards two extreme cases of the implementation, 8 bit precision with 5 dimensions/data and 12 bit precision with 65 dimensions/data. For these cases, the maximum PE utilization number is 115 and 64, while the operating frequency of the design is 300 MHz

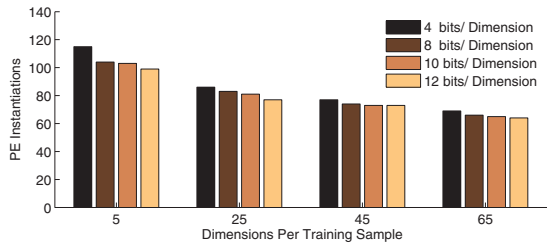

Fig. 9. PE's Instantiations.

Table 1. Lambda Module Implementation Results.

Bits	Dims	Operating Frequency (MHz)	Area (slices)	DSPs	Latency (clocks)
4	5	300	140	2	28
8	5	270	284	2	30
8	10	270	296	2	36
8	16	270	304	2	44
12	65	203	483	2	99

and 203 MHz, respectively.

5. CONCLUSION

A novel scalable architecture for the acceleration of SVM training based on Gilbert's Algorithm has been presented. Even though the hardware mapping concerns a class of FPGA devices, the results can be extended to other case studies with different resources constraints. The results show that the efficient implementation of PE can maximally utilize the FPGA resources and significantly reduce the SVM training time for large-scale problems with high dimensional data. The proposed architecture overperforms the software implementation of the algorithm, accelerating the training problem by 3 orders of magnitude. The PE is designed for linear SVMs, however an extension to the non-linear case can be easily accommodated by the proposed architecture by replacing the inner products with kernel functions.

The results of this work will be futurely exploited for the proposal of a more integrated solution, using chunking methods and clustering to answer SVM online training problems with very large training datasets.

6. REFERENCES

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

[2] J.-C. Terrillon, M. N. Shirazi, M. Sadek, H. Fukamachi, and S. Akamatsu, "Invariant face detection with support vector machines," *icpr*, vol. 04, p. 4210, 2000.

[3] K.-R. Müller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik, "Predicting time series with support vector machines," in *ICANN '97: Proceedings of the 7th International Conference on Artificial Neural Networks*. London, UK: Springer-Verlag, 1997, pp. 999–1004.

[4] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," Tech. Rep., 1998. [Online]. Available: citeseer.ist.psu.edu/platt98sequential.html

[5] T. Joachims, "Transductive inference for text classification using support vector machines," in *Proceedings of ICML-99, 16th International Conference on Machine Learning*, I. Bratko and S. Dzeroski, Eds. Bled, SL: Morgan Kaufmann Publishers, San Francisco, US, 1999, pp. 200–209. [Online]. Available: citeseer.ist.psu.edu/joachims99transductive.html

[6] C. Chang and C. Lin, "Libsvm: a library for support vector machines," Tech. Rep., 2001. [Online]. Available: citeseer.ist.psu.edu/chang01libsvm.html

[7] T. Joachims, "Training linear svms in linear time," in *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2006, pp. 217–226.

[8] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, "A fast iterative nearest point algorithm for support vector machine classifier design," *IEEE-NN*, vol. 11, no. 1, p. 124, January 2000. [Online]. Available: citeseer.ist.psu.edu/keerthi99fast.html

[9] M. E. Mavroforakis, M. Sdralis, and S. Theodoridis, "A novel svm geometric algorithm based on reduced convex hulls," in *ICPR '06: Proceedings of the 18th International Conference on Pattern Recognition*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 564–568.

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

[11] 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. [Online]. Available: http://link.aip.org/link/?SJC/4/61/1

[12] R. Collobert, S. Bengio, and Y. Bengio, "A parallel mixture of svms for very large scale problems," *Neural Comput.*, vol. 14, no. 5, pp. 1105–1114, 2002.

[13] H. Yu, J. Yang, and J. Han, "Classifying large data sets using svms with hierarchical clusters," in *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2003, pp. 306–315.