Optimizing SVM Classifier through Approximate and High Level Synthesis Techniques

Konstantina Koliogeorgi, Georgios Zervakis, Dimitrios Anagnostos, Nikolaos Zompakis, Kostas Siozios*
School of Electrical & Computer Engineering, National Technical University of Athens, Greece
**Department of Physics, Aristotle University of Thessaloniki, Greece*
{konstantina, zervakis, anagnostos.d, nzompaki}@microlab.ntua.gr
ksiop@auth.gr

***Abstract*—Leveraging the inherent error resilience of a large number of application domains, approximate computing is established as an efficient design alternative to improve their performance. Support Vector Machine (SVM) classifier is a widely adopted machine learning algorithm, that exhibits high error resilience and requires real-time execution. In this paper, we propose a highly optimized approximate SVM FPGA accelerator, utilizing arrhythmia detection in ECG signals as a case study. The proposed methodology applies two algorithmic approximation techniques, i.e., precision scaling and loop perforation, implemented in a coordinated manner in High-Level Synthesis (HLS). As a second level of performance enhancement, an exploration of the in-build optimization techniques of the HLS tool, with respect to the applied approximation, is also performed. Experimental evaluation shows that the proposed approximate SVM classifier attains a 15× speedup, while maintaining an accuracy of 96.7%.**

I. Introduction

The ever increasing computational demands in today's embedded systems call for radical changes to conventional approaches, in order to sustain and further improve the efficiency of our systems. A very promising novel approach lies in the field of approximate computing. Recent research by Intel, IBM and Microsoft has demonstrated that there is a large number of application domains that exhibit an intrinsic error tolerance [1]. Approximate computing, exploits this inherent error resilience to trade accuracy for gains in other metrics (e.g., performance, energy) and manages to be established as an alternative for efficient systems design [1]. Driven by this high potential for performance and utilization efficiency, designing approximate circuits has attracted significant research interest. In hardware design, algorithmic and logic approximations are applied [1]–[7]. Approximate design mainly targets arithmetic units (e.g., adders [1], [2] and multipliers [4], [5]), but their efficient application in complex accelerator circuits is not comprehensively analyzed and remains arguable. Despite significant results for approximate accelerators, e.g., [6]–[8], research activities on approximate FPGA accelerators are still limited. Considering that signal/image processing applications and neural networks (such as classifiers) are perfect candidates for both FPGA design and approximation, we examine the combined impact of two well-known and efficient algorithmic approximation techniques (precision scaling [7] and loop perforation [9]) in designing approximate FPGA accelerators.

A well established representative of machine learning kernels is Support Vector Machines (SVM) classifier. SVM-based classifiers [10] have grown very popular in many machine learning applications and have been extensively used for classification tasks in fields such as bio-medicine, image processing, and deep learning [11], [12]. These are data-intensive applications and often need to operate under real-time constraints. Therefore, they are in high need of acceleration. Due to their error resilience, approximate computing can be leveraged to that objective. Targeting approximate SVM implementations, [13] introduces an approximate adder and multiplier, and incorporates them to the SVM. The approximation is based on truncating the less significant bits of operands and results. In [14], a different approach is proposed, that applies approximation by removing control signals in their proposed multi-stage ripple carry adder.

In this paper, we propose an approximate FPGA-based SVM accelerator developed in Vivado HLS. We use as a case study arrhythmia detection based on ECG signal analysis and performed through SVM classification. The SVM prediction model was trained and tested on real ECG data [15] from MIT-BIH Arrhythmia Database [16]. The performance enhancement achieved is two-fold since speedup is acquired by both approximations and high level synthesis optimization techniques. The approximation techniques employed target both precision scaling and loop perforation that reduce the computational latency. This approximate version is further tuned by performing design space exploration of high level synthesis optimization techniques.

II. Proposed Approximate SVM Classifier

Considering the data dependencies of the kernel and the complexity of the performed computations, we apply approximate techniques, that reduce both the number and complexity of the performed operations, as well as high level synthesis optimization techniques that boost performance. For the rest of the paper, the accuracy of the approximate SVMs refers to the percentage of correct classifications in a set of heart beats.

A. Theoretical Background

Support Vector Machines (SVMs) are supervised machine learning models used for data-driven modelling and classification. An SVM is trained to classify an input feature vector into one of two classes. The training phase of the algorithm is performed offline and its outcome is a set of N_sv support vectors, i.e. critical points of the different classes that define the classification hyper-planes. A new input vector with D_sv features, is classified according to its distance from the support vectors. A kernel function K, in this case the Radial Basis

Function (RBF) is used to map input vectors to a space where different classes are linearly separable. The classification result is calculated based on Eq. 1, 2:

$$Class = sgn(\sum_{i=1}^{N\_sv} (y_i * a_i * K(\mathbf{x}, \mathbf{sup\_vector}_i)) - b), \quad (1)$$

$$K(\mathbf{a}, \mathbf{b}) = exp(-\gamma ||\mathbf{a} - \mathbf{b}||^2), \quad (2)$$

whrere K is the kernel function, $\mathbf{sup\_vector}_i$ is the i-th support vector, y_i, a_i are coefficients and b a bias value derived during training.

Listing 1: SVM original prediction code

```
const float sv_coef[N_sv];
const float sup_vectors[D_sv][N_sv];
void SVM_predict (float test_vector[D_sv], int * y){
 loop_i:for (i=0; i<N_sv; i++){
   loop_j:for (j=0; j<D_sv; j++){
     diff=test_vector[j]-sup_vectors[j][i];
     norma = norma + diff*diff;}
   sum = sum + exp(-gamma*norma)*sv_coef[i];
   norma=0;}
 sum = sum - b;
 if (sum<0) *y = -1;
 else *y = 1;}
```

Listing 1, introduces the C-language based implementation of equation 1. The input is a new feature vector to be classified and the outcome is the class label of the input vector. The SVM model utilized features 1222 support vectors and 18 test vector width, with 99.88% classification accuracy.

B. Approximate Techniques

1) Loop Perforation: The first approximate technique that we examine is loop perforation [9], i.e., omitting loop iterations. Loop perforation was introduced in software approximation, but since it is an algorithmic technique it can be equivalently applied in HLS. Although it can be applied to any algorithm, its optimal tuning is application-specific. In this case, the data dependencies of the SVM accommodate loop perforation. As seen in Listing 1, the squared euclidean distance of a single $test\_vector$ and each sup_vector is computed ($norma$), filtered through the RBF kenrel and eventually accumulated to a variable that defines the classification result. The contribution of each support vector to the total sum is irrelevant to the contribution of the others. Hence, loop perforation appears to be a very promising approximation candidate for SVM that allows us to eliminate operations (and thus the total latency of the kernel), as well as the area footprint, at a small accuracy loss.

In order to efficiently apply loop perforation, we utilize a greedy approach to identify the support vectors to be omitted. In this study, we consider loop perforation ranging from 2% up to 10% with step 2%. Given a target of p% perforation, our greedy algorithm implements an iterative procedure. In each iteration, we perforate an additional support vector. The iterations terminate when p% perforation is reached. Given the support vector matrix of the previous iteration, we evaluate the classification accuracy when also perforating each one of the remaining support vectors. Then we perforate the respective vector that achieves the highest accuracy. Table I, presents the accuracy and speedup over the exact-SVM, of the SVMs produced by our greedy approach. The attained accuracy ranges from 98.0% for 2% perforation target to 90.4% for 10% perforation target. The respective speedups range from 1.41× to 1.54×. As it can be seen, in the latter case extra perforation comes at a significant accuracy loss without compensating with significant additional speedup. Further perforation is not of interest in this application due to lower accuracy rates.

TABLE I: Impact of Loop Perforation on SVM Accuracy.

Perforation %	Accuracy%	Speedup
2%	98.02%	1.41
4%	97.12%	1.45
6%	96.27%	1.47
8%	92.85%	1.51
10%	90.43%	1.54

TABLE II: Fixed point Data types Initial Configuration in Bits

Variable	Bit-Width	Integer Bits	Decimal Bits
$test\_vector$	24	2	22
$sup\_vectors$	24	2	22
sv_coef	32	10	22
$diff$	25	3	22
$norma$	31	9	22
sum	32	10	22

2) Precision Scaling: Vivado HLS provides fixed-point precision data types for C/C++ kernels. We can leverage this feature to apply precision scaling, i.e., implement the SVM kernel with smaller bit-widths. This will result in smaller hardware operators and thus faster circuit.

Migrating from standard C types to arbitrary precision types is not trivial in terms of maintaining the correctness. For that reason, we perform an exploration to refine the utilized data types to their optimal size. Six different data types are defined for variables $test\_vector$, $sup\_vectors$, sv_coef, $diff$, $norma$, sum of Listing 1. We use fixed point representation for each data type and examine varying precision for the decimal part that ranges from 12 up to 22 bits. In our exploration we evaluate all the possible combinations with respect to the data type and its precision. For example Table II shows the precision assigned to each data type for the most accurate configuration resulted by this exploration. The accuracy of this configuration is 99.82% and its attained speedup is 2× with respect to the exact implementation.

C. Design Space Exploration of HLS Directives

Performance of the SVM kernel can be further improved by meticulously tuning the HLS directives available by Vivado HLS. The tool performs some optimizations by default and also allows the user to impose directives and constraints of his own choice. The set of directives provided by HLS, aim at performance and area optimization and can be applied on functions, loops, arrays and regions containing one or more of the above. Table III summarizes the HLS directives that have been incorporated in the design exploration.

The directives and their assignment to specific elements of the code, were selected after considering the inherent parallelism of the algorithm. On a coarse-grain level, parallelism is exploited by calculating the contribution of different support vectors simultaneously. On a fine-grain level, parallelism is

found within the computations required for calculating the contribution of a single support vector, i.e. $loop\_j$. The computations regarding each element of the vectors in $loop\_j$ can be performed independently of each other and thus in parallel.

Taking these into account, the directives examined are **Loop pipeline** and **Unroll**, **Array partition** and **Reshape** (Table III) [17]. Both outer and inner loop can be unrolled by applying $unroll$ directive and directives $partition$ and $reshape$ can be applied to the arrays accessed by the respective iterators, to increase the number of their ports or their word-width and allow simultaneous access to their elements. A full search space exploration consists of all valid combinations of the directives and their configuration. To avoid an exhaustive design space exploration, we utilize the framework proposed in [18], which applies effective pruning strategies and reduces the search space from 100000 to approximately 1000 configurations. The proposed pruning guidelines are based on an efficient arrangement of data on BRAMS, which guarantees that the elements which are required simultaneously, are either packed together into the same word or accessible from an adequate number of BRAM ports.

D. Methodology

The techniques described so far should be combined in a synergetic and strategic manner to deliver a highly optimized approximate SVM classifier. The non-linear and nontrivial inference effects between the applied approximation techniques as well as the HLS optimization directives regarding both performance as well as accuracy require the use of heuristic algorithms in practice. A brute-force evaluation of all possible combinations of the examined techniques, would lead to an enormous design space. In order to avoid an exponentially growing search space, we propose a framework that incrementally exploits each technique and gradually builds up to the implementation of an efficient FPGA SVM classifier, accelerated through both approximate and high level synthesis optimization techniques.

As a first step, we apply loop perforation, carefully adjusted to the kernel's requirements. Specifically, loop perforation technique is applied to the exact SVM classifier, using the greedy algorithm described in Section II. This step results in constructing five approximate SVM classifiers, each one exhibiting a perforation percentage of 2%,4%,6%,8% and 10% respectively and an initial boost in performance.

As a second level of optimization, we explore the performance enhancement that can be extracted by meticulously tuning the in-build optimization knobs of HLS tool with respect to the previously conducted approximations. To avoid an exhaustive design space exploration, we utilize the framework proposed in [18], which elegantly prunes the design space and efficiently converges towards the Pareto front. The basis of the proposed pruning strategies lies in customizing the memory architecture according to the computation and memory access patterns of the algorithm. Since different perforation percentages result in different memory layouts and thus different optimal configurations of HLS directives, each approximate-perforated SVM should be evaluated separately. Therefore, we perform an efficient design space exploration and acquire a resource utilization-speedup Pareto-front for each approximate SVM resulted by loop perforation. Note that all design points within the same Pareto exhibit the same classification accuracy.

On the last optimization level, we further boost the performance by applying precision scaling. For each Pareto point, an exploration is performed to refine the utilized data types to their optimal precision. We evaluate all the possible combinations (Section II-B2), and a new Pareto Front is extracted. The aproximate designs in this Pareto front, apply both loop perforation and precision scaling, as well as HLS optimization.

III. Experimental Evaluation

In this section, an evaluation of the proposed framework is provided. Xilinx Vivado-HLS(v2018.2) and the Zynq7 ZC706 Evaluation Board are used to implement all SVM accelerators. Two different datasets are used in our evaluation. A training set of 47856 inputs is utilized for the implementation of the approximate SVMs and the design space exploration, and a separate testing set of 52291 inputs is utilized to evaluate the classification accuracy of the proposed approximate SVMs. Both sets are composed of feature vectors, extracted from real ECG signals of the MIT-BIH Arrhythmia Database.

Fig. 1 depicts the Pareto Front, which is the output of our methodology. This study explores the trade-off between speedup and classification accuracy when all optimization levels have been applied. Accuracy ranges from 90% to 99% and speedup from 1× to 18×. The highest speedup is achieved for aggressive approximation, i.e. 10% perforation and significant precision scaling. The trade-off of course is reduced accuracy, which is crucial for an application such as arrhythia detection. Fig.2 presents the speedup achieved for the fastest configuration resulting by each technique with a target accuracy of 95%. Perforation technique satisfies this constraint for 6% perforation, 1.47× speedup and 96.27% accuracy, whereas precision scaling delivers a configuration with 4× speedup and 97.32% accuracy. Our proposed methodology, delivers a configuration that combines 2% perforation, precision scaling and HLS optimization techniques and outperforms them by delivering 15× speedup and 96.7% accuracy. Finally, Fig. 3 presents the resources utilization for the corresponding SVMs. The precision scaling technique is the most efficient one in resources utilization, since reducing the bit width leads to fewer BRAMS for storing the support vectors and smaller-width operators. The small decrease in resources utilization due to perforation, is attributed to storing less support vectors at BRAMS. Our proposed technique shows greater utilization, due to the HLS directives, that require more resources to increase parallelism and dominate over the resources savings achieved by perforation and precision scaling. This increase is compensated by the greater speedup.

IV. Conclusion

In this paper, we present a highly optimized approximate SVM accelerator implemented in Vivado HLS. The SVM model assists in arrhytmia detection in real ECG signals from MIT-BIH Arrhythmia Database, and requires acceleration to meet its real-time constraints. To boost its efficiency, the proposed SVM combines, in a co-ordinated manner, two approximate computing techniques and in-build tuning knobs provided by the HLS tool. Our experimental evaluation shows

TABLE III: HLS directives

Directive	Description
PIPELINE	Reduces the initiation interval by concurrent execution of operations within a loop or function.
DATAFLOW	Task level pipelining. Functions and loops are executed concurrently. Used to minimize interval.
INLINE	Inlines a function, removing all function hierarchy. Used to enable logic optimization across function boundaries and improve latency/interval by reducing function call overhead.
UNROLL	Unrolls for-loops to create multiple independent operations rather than serial executed ones.
ARRAY_PARTITION	Partitions large arrays into multiple smaller arrays or into individual registers, to improve access to data and remove block-RAM bottlenecks.
ARRAY_MAP	Combines multiple smaller arrays into a single large array to help reduce block-RAM resources.
ARRAY_RESHAPE	Reshape an array from one with many elements to one with greater word-width. Useful for improving block-RAM accesses without using more block-RAM.

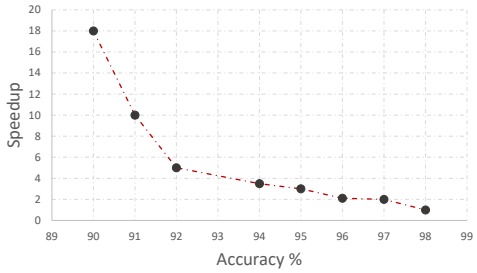

Fig. 1: Pareto Front for Accuracy and Speedup Metrics

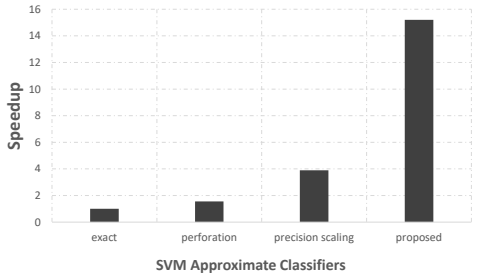

Fig. 2: Speedup for Fastest SVM of each technique.

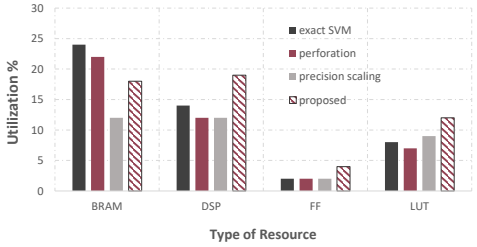

Fig. 3: Resources Evaluation for Fastest SVM of each technique.

that the proposed SVM attains up to 15× speedup compared to the exact design and that it significantly outperforms state-of-the-art approximation techniques.

ACKNOWLEDGEMENT

This work was supported in part by Greece and the European Union (European Social Fund) through the Operational Programme Human Resources Development, Education and Lifelong Learning 20142020 in the context of the project Automated methodology for production and execution of data-centric multi-level approximate equivalent applications for heterogeneous computing platforms under Grant MIS 5005377.

REFERENCES

[1] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *Design Automation Conf.*, June 2015.
[2] H. A. F. Almurib, T. N. Kumar, and F. Lombardi, "Inexact designs for approximate low power addition by cell replacement," in *Design, Automation Test in Europe*, 2016.
[3] G. Zervakis, S. Xydis, K. Tsoumanis, D. Soudris, and K. Pekmestzi, "Hybrid approximate multiplier architectures for improved power-accuracy trade-offs," in *International Symposium on Low Power Electronics and Design*, July 2015, pp. 79–84.
[4] W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, and F. Lombardi, "Design of approximate radix-4 booth multipliers for error-tolerant computing," vol. 66, no. 8, pp. 1435–1441, Aug 2017.
[5] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris, and K. Pekmestzi, "Design-efficient approximate multiplication circuits through partial product perforation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 10, pp. 3105–3117, Oct 2016.
[6] A. Lingamneni, C. Enz, K. Palem, and C. Piguet, "Synthesizing parsimonious inexact circuits through probabilistic design techniques," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, pp. 93:1–93:26, May 2013.
[7] C. Li, W. Luo, S. S. Sapatnekar, and J. Hu, "Joint precision optimization and high level synthesis for approximate computing," in *Design Automation Conference*, 2015, pp. 104:1–104:6.
[8] G. Zervakis, S. Xydis, D. Soudris, and K. Pekmestzi, "Multi-level approximate accelerator synthesis under voltage island constraints," *IEEE Trans. Circuits Syst. II*, pp. 1–1, 2018.
[9] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proc. 19th ACM SIGSOFT Symp. 13th Eur. Conf. Found. Softw. Eng. (ESEC/FSE)*, 2011.
[10] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
[11] Y. Zhang, S. Wang, G. Ji, and Z. Dong, "An mr brain images classifier system via particle swarm optimization and kernel support vector machine," *The Scientific World Journal*, vol. 2013, 2013.
[12] Y. Tang, "Deep learning using linear support vector machines," *arXiv preprint arXiv:1306.0239*, 2013.
[13] Y. Zhou, J. Lin, and Z. Wang, "Energy efficient svm classifier using approximate computing," in *ASIC (ASICON), 2017 IEEE 12th International Conference on*. IEEE, 2017, pp. 1045–1048.
[14] Y. Wu, X. Yang, A. Plaza, F. Qiao, L. Gao, B. Zhang, and Y. Cui, "Approximate computing of remotely sensed data: Svm hyperspectral image classification as a case study," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 12, pp. 5806–5818, 2016.
[15] D. Azariadi, V. Tsoutsouras, S. Xydis, and D. Soudris, "Ecg signal analysis and arrhythmia detection on iot wearable medical devices," in *Modern Circuits and Systems Technologies (MOCAST), 2016 5th International Conference on*. IEEE, 2016, pp. 1–4.
[16] G. B. Moody and R. G. Mark, "The impact of the mit-bih arrhythmia database," *Engineering in Medicine and Biology Magazine, IEEE*, vol. 20, no. 3, pp. 45–50, 2001.
[17] T. Feist, "Vivado design suite," *White Paper*, vol. 5, 2012.
[18] V. Tsoutsouras, K. Koliogeorgi, S. Xydis, and D. Soudris, "An exploration framework for efficient high-level synthesis of support vector machines: Case study on ecg arrhythmia detection for xilinx zynq soc," *Journal of Signal Processing Systems*, vol. 88, no. 2, pp. 127–147, 2017.