SCoPE: Towards a Systolic Array for SVM Object Detection

Christos Kyrkou, *Student Member, IEEE* and Theocharis Theocharides, *Member, IEEE*

***Abstract*—This paper presents SCoPE (Systolic Chain of Processing Elements), a first step towards the realization of a generic systolic array for support vector machine (SVM) object classification in embedded image and video applications. SCoPE provides efficient memory management, reduced complexity, and efficient data transfer mechanisms. The proposed architecture is generic and scalable, as the size of the chain, and the kernel module can be changed in a plug and play approach without affecting the overall system architecture. These advantages provide versatility, scalability and reduced complexity that make it ideal for embedded applications. Furthermore, the SCoPE architecture is intended to be used as a building block towards larger systolic systems for multi-input or multi-class classification. Simulation results indicate real-time performance, achieving face detection at ~ 33 frames per second on an FPGA prototype.**

***Index Terms*— Field-programmable gate array (FPGA), Support Vector Machine (SVM), Systolic Array, Object Detection.**

I. Introduction

Support Vector Machines (SVMs) have been widely adopted since their introduction by Cortes and Vapnik [1]. They are considered one of the most powerful classification engines due to their high classification accuracy rates, which in many cases outperform well established classification algorithms such as neural networks. Consequently, there has been growing interest in utilizing SVMs in embedded object detection and other image classification applications. While software implementations of SVMs yield high accuracy rates, they cannot efficiently meet real-time requirements of embedded applications (failing to take advantage of the inherent parallelism of the algorithms). Hence, SVM hardware architectures emerged as a potential solution for real-time performance. The majority of these emerging architectures presented in literature is directed towards specific applications and cannot be easily modified to adapt in other scenarios. Only few works have looked into generic and versatile architectures to satisfy the variety of embedded object-detection applications. Literature suggests that existing general-purpose SVM architectures do not scale well in terms of required hardware resources, complexity, data transfer (wiring) and memory management, primarily because of two important constraints; the number of support vectors (SVs) and their dimensionality [2]. An SVM with a small number of SVs, requires only a few computational modules, however, several applications require a large number of high dimensional SVs. As such, a general-purpose SVM architecture with a large number of modules faces design challenges such as fan-out and routing, increased complexity, and other performance limitations. An efficient SVM architecture needs to be scalable, provide real-time performance, and be easy to design and implement. Issues such as parallel memory access, distribution of the inputs to the computation modules while taking into consideration fan-out requirements, and efficient computation of the kernel operations (requires the most computational power), need to be considered.

This work presents a distributed pipelined architecture that can be expanded in a systolic manner, providing efficient management for memory and data (transfer/wiring) resources. The Systolic Chain of Processing Elements (SCoPE) is a modular architecture consisting of almost identical processing elements (PEs). The architecture can be scaled according to the available hardware budget, without affecting the operation or frequency of the system. The architecture also addresses the wiring and fan-out complexity of routing the input vector to the classification modules. A prototype chain of the SCoPE architecture was implemented on a Virtex 5 FPGA and was evaluated using training data from a face detection application. However, the architecture is suitable for several SVM image classification applications.

This paper briefly explains the basic principles of SVM classification, and provides a brief overview of other hardware implementations that are present in the literature in section II. The SCoPE architecture is presented in section III and an evaluation of a SCoPE prototype is presented in section IV. Finally Section V provides some insight on the ongoing work and future considerations.

II. SVM Background & Related Work

A. Review of Support Vector Machines

An SVM is a binary classification algorithm where each sample in the data set is viewed as a *k*-dimensional vector [3]. The SVM takes a data set that contains samples from two classes (labeled -1 and +1), and constructs separating hyperplanes between them. The separating hyperplane that best separates the two classes is called the *maximum-margin hyperplane* and forms the decision boundary for classification. The *margin* (shown in Fig. 1 (a)) is defined as the distance between two parallel hyperplanes that lie at the boundary of each class. The data samples that lie at the boundary of each class and determine how the *maximum-margin hyperplane* is

formed, are called *support vectors* (SVs) [3]. The SVs are obtained during the training phase and are then used for classifying new (unknown) data. When two data classes are not linearly separable, a kernel function is used to project data to a higher dimensional space (feature space), where linear classification is possible. This is known as the *kernel trick* [3] and allows an SVM to solve non-linear problems. An input is classified at runtime using the *classification decision function*,

- $D(\vec{x}) = sign(\sum_{i=1}^{m} a_i y_i K(\vec{x}, \vec{s_i}) + b)$ (1)

in which a_i are the *alpha* coefficients (weight of each SV), y_i are the class labels of the SVs, $\vec{s_i}$ are the SVs, $\vec{x}$ is the input vector, $K(\vec{x}, \vec{s_i})$ is the chosen kernel function and b is a bias value. The flow of this computation is illustrated in Fig. 1 (b). The kernel computation (steps 2 & 3 in Fig. 1 (b)) is split into two parts, the vector operation and kernel-dependent additional operations on the produced scalar value. Both parts depend on the choice of kernel function (popular kernel functions are shown in (2)-(5)). For example, the vector operation for kernels (2), (3) and (4) involves a dot-product computation, whereas for (5), it involves the computation of the squared norm of the difference of two vectors.

Popular Kernel Functions:

- **Linear**: $K(\vec{x}, \vec{z}) = (\vec{x} \cdot \vec{z})$ (2)
- **Polynomial**: $K(\vec{x}, \vec{z}) = (\gamma(\vec{x} \cdot \vec{z}) + r)^d, \gamma > 0$ (3)
- **Sigmoid**: $K(\vec{x}, \vec{z}) = \tanh((\vec{x} \cdot \vec{z}) + \theta)$ (4)
- **Radial Basis Function**: $K(\vec{x}, \vec{z}) = exp(\|(\vec{x} - \vec{z})\|^2/(2\sigma^2))$ (5)

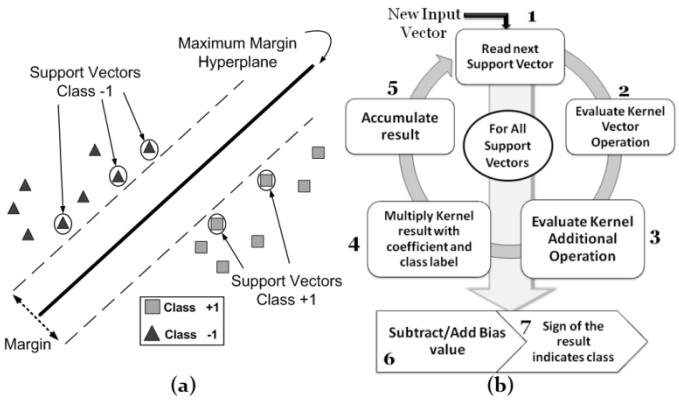

Fig. 1. (a) Maximum-margin hyperplane and margin for an SVM trained with samples from two classes. (b) Data flow for the computation of (1).

B. Related Work on Support Vector Machines

Hardware implementations of SVMs have gained noticeable interest in recent years, primarily because of the potential real-time performance benefits they offer. Most implementations have targeted either specific applications, or require a large amount of computational resources, and thus are not suitable for general-purpose embedded environments. An attempt to design a massively parallel architecture to accelerate learning algorithms was presented in [4], which utilized arrays of vector processing elements controlled through a host CPU in SIMD fashion. The centralized nature of the CPU, however, creates a bottleneck, emphasizing the need for distributed control. An FPGA implementation utilizing a Microblaze processor as a control unit and custom hardware coprocessors to perform the SVM classification was presented in [5]. The design was restricted to 8-dimensional vectors, limiting the application space. An integrated vision system for object detection and localization based on SVM was presented in [6]; however, the system offered limited scalability. An analog custom processor was presented in [7] and an implementation of a training algorithm for SVMs was presented in [8]. SVM implementations have also been explored using embedded processors in [9] and [10]. Considering the limitations of previous works, SCoPE was designed to provide scalability, and reduced memory and data flow complexity, making it suitable for general purpose embedded classification applications.

III. SCoPE: The Proposed Architecture

The proposed architecture consists of a chain of PEs that operates in a pipelined manner and can be expanded vertically with minor modifications (introduction of vertical data flow) to operate in a systolic manner. This paper focuses on describing the design and flow of operation of a single horizontal chain; the full architecture is part of ongoing work.

A. The SCoPE Components

SCoPE consists of three main regions (Fig. 2); the *front-end* (input), the *middle* (computation) and *back-end* (output) regions. The *front-end* includes the input vector memory, address generator units and the *front-end* PE, which is the input part of the chain. The *middle* region bears the bulk of the computation where each PE receives data from the previous PE, processes the data, and propagates it to the next PE in pipelined manner. The *back-end* operates as the output of the system; it consists of the *back-end* PE that transfers all the data outside of the chain, a kernel unit, a MAC unit, and the alpha coefficients memory. All three regions are supervised by a finite state machine control unit.

The *front-end* and *back-end* PEs differ with the *middle* PEs by having additional control signals and modules in order to interact with the I/O interfaces at each end. Each PE consists of a data transfer unit, a processing unit and a memory unit, along with glue logic assisting in synchronization and control (Fig. 3). The main processing unit is a multiplication-accumulation (MAC) unit that performs the kernel's vector operation between the input vector and SV elements. The transfer unit propagates data from one PE to the next. The memory unit holds a group of SVs for which the PE performs the kernel vector operation with the input vector. Distributing the SV data to each PE, allows for a scalable and manageable system, and also contributes to smaller and faster memory units. Each PE has three operational states: PROCESSING, IDLE and TRANSFERRING. In the PROCESSING state the PE is involved with the computation of the vector operation, and simultaneously transfers data/control values to the next PE. These values are encoded in a 25-bit word as shown in Fig. 4. In the IDLE state, the PE remains idle. Lastly, in the TRANSFERRING state, the 25-bit scalar value computed by each PE is transferred to the next PE. Data switching is done through a 2-1 multiplexer and data propagation through a register.

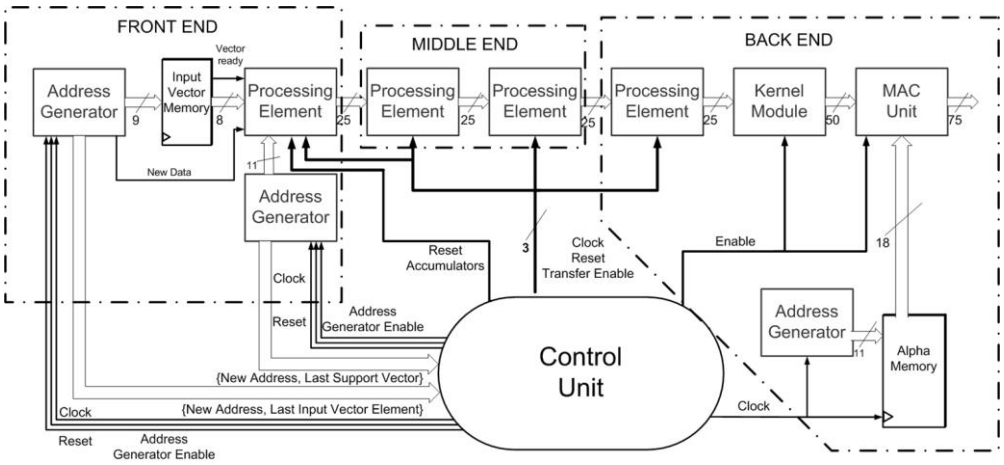

Fig. 2. Block diagram of the SCoPE architecture illustrating the three main regions and their basic components.

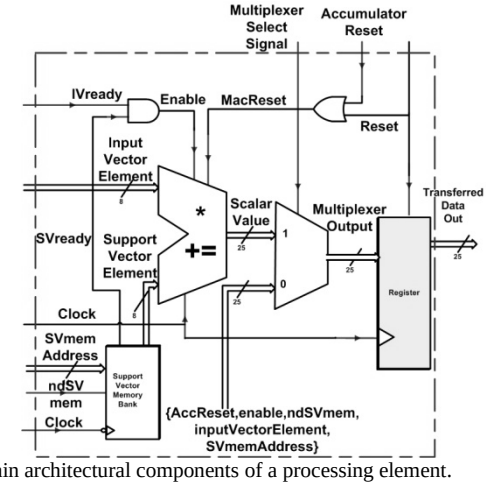

Fig. 3. Main architectural components of a processing element.

24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
6			5	4	3	2																1		

Fig. 4. In the PROCESSING state, data values and control signals are transferred in an encoded 25-bit word: (1) Address for SV memories. (2) Input vector element. (3) New address signal for SV memory. (4) Enable signal for PE's MAC unit. (5) Reset signal for PE's MAC unit. (6) Left unused for future implementations.

The kernel and *back-end* MAC units make up the remaining components and require the most processing resources. The type of kernel does not impact the data flow, but influences the maximum achievable system performance. The kernel can be implemented either as a Look-Up Table (LUT) or by using custom logic. The latter may reduce the maximum operating frequency due to increasing computational complexity, while the former impacts the accuracy of the classification and increases memory demands, depending on the size and precision of the LUT. The *back-end* MAC unit's bitwidth (precision) is determined by the choice of kernel (i.e. the kernel's output bitwidth), the number of SVs (determines the number of accumulations and consequently the precision of the accumulator) as well as the chosen precision for the alpha coefficients. One of the main advantages of the proposed architecture is that these two resource-hungry components are shared amongst the PEs, thus reducing the overall hardware overhead and complexity.

B. The SCoPE Flow of Operation

The classification procedure begins with the PEs in the IDLE state. The system receives the first input vector element and generates the address for the SV memory, leading the *front-end* PE in the PROCESSING state. The SV memory in the *front-end* PE outputs the corresponding SV element and the computation begins. The rest of the PEs in the chain follow this computation pattern, one after another. When the scalar value is computed in all PEs, all PEs enter the TRANSFERING state simultaneously, propagating the computed scalar value towards the *back-end* PE. The kernel and *back-end* MAC units then begin processing each scalar value that they receive from the *back-end* PE. The kernel outcome is computed, multiplied with its respective preloaded coefficient and accumulated. When the accumulation is completed, the control unit asserts an *accumulation reset* signal. This signal is propagated through the chain to reset the MAC unit values in each PE. Each PE will then again enter the PROCESSING state to begin a new kernel vector operation. When all SV operations finish, the bias is processed. The total number of cycles required for the classification of an input vector is given by:

- $(n + k + (n + 2)) * \lceil m/n \rceil$ (6)

The first input vector element takes *n (number of PEs)* cycles to reach *the back-end* PE (i.e. to fill the pipeline) and the PEs then take *k (number of elements in vector)* cycles to compute the scalar value. The *front-end* PE's scalar value (which is the furthest away from the *back-end* region) takes an additional *n+2* cycles to reach the kernel unit and be accumulated. The chain processes *n* SVs in parallel, so if we assume *m* total SVs in a training set, we need $\lceil m/n \rceil$ repetitions in total, for all SVs.

IV. Evaluation & Results

A prototype of the proposed architecture with a chain of 100 PEs was implemented on a Virtex 5 FPGA and evaluated using face detection, a popular embedded application. The MAC units were mapped both on the FPGA's embedded DSP units and on the slice logic, and the internal SV memory in each PE was implemented using the dedicated FPGA block RAM. The SVM was trained in MATLAB, using 6,800 samples (4,400 non-faces, 2,400 faces) with training

information from [11]. The training phase produced m=818 SVs that were distributed amongst the n=100 PEs (some PEs had 9 SVs and some had 8 SVs in their SV memory). Each vector consisted of k=400 8-bit elements, corresponding to an input 20x20 grayscale image. The polynomial kernel was used, with parameters r=0, γ=1 and d=2, (i.e. a square operation). The alpha coefficients had a fixed point 18-bit format. Table I shows the synthesis results of the proposed architecture, which is limited by the multiplier resources (available slice logic and DSP units) and operating frequency of the FPGA. The design can be scaled to the available hardware budget and its modularity and ability to be super-pipelined allow for further performance improvement.

To the best of our knowledge, there is limited work dealing with general SVM hardware implementations, and performance metrics used in existing works relate to the specific benchmark applications used in each implementation. Therefore, comparing our preliminary results to other works is impractical, as several factors affect the resulting frame rate, such as the search window size, window overlap and the source image size. Table II shows the parameters chosen in the experimental framework and the computation of the frame rate achieved for a single chain. The targeted FPGA can, however, fit 5 parallel SCoPE modules with shared SV memory units, raising the frame rate up to ~ 33 fps, shown also in Table II. The experimental architecture achieves 88% detection accuracy, which can be further increased by improving the training set with more training samples. However, this may result in an increased number of SVs.

TABLE I: SYNTHESIS RESULTS FOR THE XILINX VIRTEX 5-LX110T FPGA

Slice Registers (69,120)	*5,162 (7%)*	**Slice LUT (69,120)**	*8,887 (12%)*
DSP48Es (64) **(64 of the multipliers were implicitly mapped on the DSP units)**	*64 (100%)*	**PE Utilization (multiplier mapped on slice logic)**	*145 of 69,120 (0.2%)*
Memory (666KB)	*329KB (49%)*	**Block Rams (148)**	*74 (50%)*

TABLE II: EXPERIMENTAL SETUP & PERFORMANCE RESULTS

Experimental Framework:			
Image Size	*320x240*	**Window Overlap**	*5 pixels*
FPGA Frequency	*100 MHz*	**# of input vectors (# of 20x20 windows)**	*2,745*
Classification Cycles:			
Single input vector (using (6))	*5,418 (~54μs)*	**Whole Image**	*14,872,410 (~0.14s)*
Frames per second:			
1 SCoPE module	*~ 6.72*	**5 SCoPE modules**	*~ 33.61*

V. ONGOING AND FUTURE WORK & CONCLUSIONS

The regular and modular nature of SCoPE provides scalability, reduced complexity, and efficient memory and data/control transfer management, especially when dealing with applications requiring a large number of high dimensional SVs. Work in progress involves evaluating the architecture using other kernel functions as well as other object detection applications. However, emphasis is being placed on expanding the architecture into a systolic array.

The future systolic array implementation will utilize both a horizontal (currently employed in SCoPE) and a vertical pipeline, where input vector elements will move along the horizontal pipeline and SV elements along the vertical. One of the rows of the array will be identical to SCoPE and will contain the distributed SV memories. The other rows will not have a memory unit, but will receive the SV elements from that row in a vertical systolic manner, as illustrated in Fig. 5.

SCoPE can potentially support multiclass classification problems with minor modifications. Each row in the array will follow the SCoPE architecture, with its own set of SVs and kernel function (if necessary), so that each row can perform different classification from the other rows, with no need for vertical data movement. However, further work is necessary to address the increasing memory demands.

SCoPE provides a modular and regular way of performing SVM classification and can be used in a variety of embedded applications. The expected systolic array implementation will take advantage of the inherent parallelism, to increase SCoPE's performance capabilities while maintaining versatility and scalability.

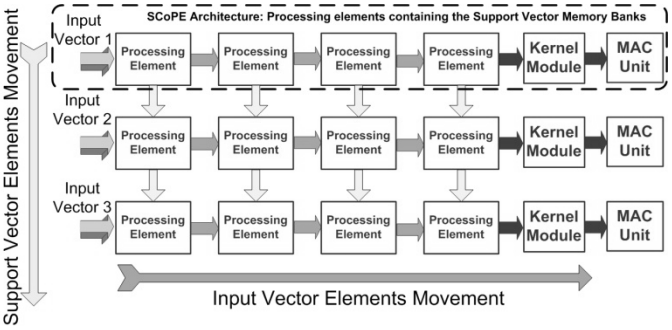

Fig. 5. Block diagram of a possible future systolic array implementation.

REFERENCES

[1] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, no. 3, 1995, pp. 273-297.

[2] R. Reyna-Rojas, D. Houzet, D. Dragomirescu, F. Carlier and S. Ouadjaout, "Object Recognition System-on-Chip Using the Support Vector Machines," EURASIP Journal on Applied Signal Processing, vol. 2005, no. 7,2005, pp. 993-1004.

[3] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," Data Mining and Knowledge Discovery, vol. 2, no. 2, 1998, pp. 121-16.

[4] H. P. Graf, et. al., "A Massively Parallel Digital Learning Processor," NIPS 2008, pp. 529-536.

[5] I. Biasi, A. Boni, and A. Zorat, "A reconfigurable parallel architecture for SVM classification," IEEE International Joint Conference on Neural Networks, vol. 5, 2005, pp. 2867-2872.

[6] R.A. Reyna, D. Esteve, D. Houzet, and M.-F. Albenge, "Implementation of the SVM Neural Network Generalization Function for Image Processing," 5th IEEE International Workshop on Computer Architectures for Machine Perception, 2000, p.147.

[7] R. Genov and G. Cauwenberghs. "Kerneltron: Support Vector Machine in Silicon," IEEE Transactions on Neural Networks, vol. 14, no. 5, 2003, pp. 1426-1434.

[8] D. Anguita, A. Boni, and S. Ridella. "A Digital Architecture for Support Vector Machines: Theory, Algorithm, and FPGA Implementation," IEEE Transactions on Neural Networks, vol. 14, no. 5, September 2003, pp. 993-1009.

[9] R. Pedersen and M. Schoeber, "An Embedded Support Vector Machine," International Workshop on Intelligent Solutions in Embedded Systems, June 2006, pp. 1-11.

[10] S. Dey, M. Kedia, N. Agarwal, and A. Basu, "Embedded support vector machine: Architectural enhancements and evaluation," Proc. of the 20th International Conf. on VLSI Design, 2007, pp. 685-690.

[11] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: an application to face detection," IEEE Conference on Computer Vision and Pattern Recognition, 1997, pp. 130-136.