FPGA implementations of data mining algorithms

P. Škoda*, B. Medved Rogina*, V. Sruk**
* Ruđer Bošković Institute, Zagreb, Croatia
** Faculty of electrical engineering and computing, University of Zagreb, Zagreb, Croatia

Abstract - In recent decades there has been an exponential growth in quantity of collected data. Various data mining procedures have been developed to extract information from such large amounts of data. Handling ever increasing amount of data generates increasing demand for computing power. There are several ways of dealing with this demand, such as multiprocessor systems, and use of graphic processing units (GPU). Another way is use of field programmable gate array (FPGA) devices as hardware accelerators. This paper gives a survey of the application of FPGAs as hardware accelerators for data mining. Three data mining algorithms were selected for this survey: classification and regression trees, support vector machines, and k-means clustering. A literature review and analysis of FPGA implementations was conducted for the three selected algorithms. Conclusions on methods of implementation, common problems and limitations, and means of overcoming them were drawn from the analysis.

I. INTRODUCTION

Thanks to development of computer systems and its applications, the last several decades have been marked by exponential growth of collected data in all areas of human activity. To deal with this continuous and increasing influx of data it was necessary to develop computational methods for extracting information and discovering knowledge. Computational process of non-trivial information extraction is called data mining. Data mining makes use of methods from closely related fields such as statistics, artificial intelligence, machine learning, pattern recognition and databases.

With most data mining methods, the quantity of data directly impacts computational load. High computational loads occur because many problems include large quantities of data and require carrying out complex computations in many-dimensional space. The issue of computational load is a significant one. Quantity of collected data continually increases which implies that available compute power must increase to keep up with it.

Since CPU clock frequency no longer increases by the Moore's law, increase in compute power must be achieved by other means. One approach to the problem is using multi-core processors, multiprocessor systems, and computer clusters, where the program is still executed on general purpose CPUs. Another approach is using hardware accelerators, where the compute-intensive parts are executed on special purpose units. Currently available accelerator units are graphics processor units (GPU) and field programmable gate array (FPGA) devices, which are the focus of this paper. Both platforms have their advantages and disadvantages, and a review of their capabilities in compute-intensive applications is presented in [1].

FPGAs are integrated circuits designed to be configured by the user after manufacturing. An FPGA contains a matrix of configurable logic blocks and a hierarchy of reconfigurable interconnects that allow the blocks to be connected together. A configurable logic block contains look-up tables, multiplexers and flip-flops which together with interconnect allow performance of complex combinatorial and sequential functions and implementation of a wide variety of digital systems as well. Modern FPGAs also contain specialized memory, arithmetic, and communication blocks which enable more efficient implementations of digital systems. By allowing implementation of custom computational architectures FPGAs provide opportunities for exploitation of parallelism inherent in implemented algorithm.

This paper gives a review of current development in FPGA based hardware acceleration of select data mining algorithms. There are many algorithms developed for, and being used in data mining. From the multitude of algorithms Wu et al. in [2] present the top ten identified by the IEEE International Conference on Data Mining in December 2006. Of those ten, three have been selected for this survey: classification and regression trees, support vector machines, and k-means clustering. For the selected algorithms a survey of literature had been conducted, and found implementations have been analyzed.

This paper is organized as follows: Short introduction to selected algorithms is given in Section II; Results of literature survey are presented in Section III; Analysis of characteristics common to most FPGA implementations are presented in Section IV; Conclusions are given in section V.

II. SELECTED ALGORITHMS

To limit the scope of this survey, three algorithms have been selected from the ten identified by Wu et al. [2]. Data mining tasks of primary interest were classification and clustering as arguably the most common tasks. Of the top ten algorithms, five—C4.5, classification and regression trees, support vector machines, k-nearest neighbors, and naïve Bayes—are used for classification and two—k-means and expectation maximization—are used for clustering. As consequence of this count, two classification and one clustering algorithms were selected. The final criterion for selection was perceived amenability to fine-grained parallelization. The seven algorithms were analyzed with special attention paid to repeated use of same arithmetic and logic operations over multiple data,

and possibilities of pipelining the algorithms' implementations. From this analysis classification and regression trees, support vector machines, and k-means clustering emerged as algorithms with potentially highest gains from fine-grained parallelization.

A. *Classification and Regression Trees*

Classification and regression tree (CART) is a decision and regression tree learning algorithm. In decision trees the output is a prediction on class to which the data item belongs. In regression trees the output is a real number, and the tree represents an approximation of the function that maps input data to predicted outcome. One other well known decision tree learning algorithm is C4.5. Decision trees are easy to interpret, can readily be converted into a set of "if-then" rules, and can work with incomplete data.

Decision trees consist of nodes and leaves. Nodes represent simple attribute test which splits the data and determines branching of the tree into two or more branches. Leaves represent predicted class labels of the input data—the classification of the data. Classification is performed passing the data through the tree starting at the root node. Attributes are tested at each node and braches followed as determined by the test. When a leaf is reached its associated class label is the classification output.

CART algorithm constructs binary trees, and can work with categorical and numerical attributes. It works with "raw" data, i.e. it doesn't require preprocessing of data. The learning process is based on finding a test that defines the best split of the data. Test for categorical attributes is defined as "attribute $X_i = C$", and for numerical as "attribute $X_i \leq C$". Split quality is estimated by some information metric. Commonly used metrics are Gini impurity, and entropy. Once the attribute test giving the best split is found, the data is split according to it and procedure repeated recursively on the obtained subsets. To prevent overfitting, the splitting process can be stopped early by some rule, or pruning process may be run after the tree is built.

B. *Support Vector Machines*

Support vector machine (SVM) is one of the most popular methods for classification. The main idea of this method is finding a classification function that separates points from two classes from by finding a hyperplane that separates them optimally. For training set $\{\mathbf{x}_i, y_i\}$, where $\mathbf{x}_i \in \mathbb{R}^n$, and $y_i \in \{-1, 1\}$, classification function is of form:

$$f(\mathbf{t}) = \operatorname{sgn}\left(\sum_i \alpha_i y_i \, \mathrm{K}(\mathbf{t}, \mathbf{x}_i) + b\right) \qquad (1)$$

where: $\mathbf{t}$ is point (vector) to be classified, α_i are positive real constants, b is a real constant, and $\mathrm{K}(\mathbf{t}, \mathbf{x}_i)$ is a kernel function. Kernel function maps points from original space into a space where they become linearly separable. Some commonly used kernels are linear, polynomial, and Gaussian.

Hyperplane that optimally separates the sets is the one that maximizes the margin, i.e. which maximizes the distance from the hyperplane to the nearest vectors from both sets. For sets that cannot be completely separated the soft margin is introduced, which allows some level of misclassification of the training set. The points closest to the hyperplane are named support vectors, and they define the classifier. Finding classifier parameters α_i is a quadratic optimization problem. There are several algorithms for learning SVMs, e.g. sequential minimal optimization, and gradient projection.

C. *K-means Clustering*

K-means clustering is a simple iterative procedure for dividing a set of points into a predefined number k of clusters. Algorithm is executed on a set of d-dimensional vectors $\{\mathbf{x}_i, \mathbf{x}_i, \ldots, \mathbf{x}_n\}$, where $\mathbf{x}_i \in \mathbb{R}^d$. Clusters $\{S_1, S_2, \ldots, S_k\}$ are represented by their centers $\boldsymbol{\mu}_j$.

The algorithm starts by choosing k points from $\mathbb{R}^d$ which will be the starting values for centers $\boldsymbol{\mu}_j$. These points can be chosen in several ways, e.g. randomly of using some heuristic. After initialization of centers, the following two steps are iterated until the algorithm converges: (a) clusters are formed by assigning points to their closest center, and (b) new centers are found by computing the mean of all points in a cluster. Algorithm converges when points no longer change clusters. For distance metric, Euclidian distance is most common.

III. Literature Survey Results

A. *Classification and Regression Trees*

Survey of literature uncovered only one publication on FPGA implementation of decision tree learning. Narayanan et al. [3] have used FPGA to implement decision tree induction system for binary classification.

Gini impurity computation was implemented in hardware, as computationally most intensive part of learning process. One peripheral unit contains 16 Gini units working in parallel, as well as additional logic for finding the minimum Gini impurity and its associated data split. The peripheral unit is connected to the CPU's local bus. The implementation uses 16.16 fixed-point arithmetic. Algorithm is executed on a PowerPC CPU embedded in the FPGA device. The system is implemented on Xilinx Virtex-II Pro platform, with 100 MHz clock rate. The achieved speedup is 5.58× compared to pure software implementation running on the same platform.

B. *Support Vector Machines*

The research in hardware implementations of SVM is closely related research in embedded systems. As consequence using floating-point arithmetic is usually avoided in order to reduce energy consumption, and to reduce usage of logic resources in FPGAs. Computationally most intensive part of SVM training process is the matrix-vector multiplication.

Anguita et al. propose in [4] what was probably the first FPGA implementation of SVM training. In the paper they present a concise analysis quantization error and its influence on performance of SVM, with emphasis on fixed-point arithmetic and its influence. They present a

new SVM training algorithm suited for hardware implementation using fixed-point arithmetic. Algorithm has two phases which are executed alternately. In the first phase the quadratic optimization problem of finding parameters α_i with fixed parameter b is solved. In second phase the found parameters α_i are fixed and parameter b is found using bisection method. The training process is executed in hardware only. Calculation of sum $\sum_i q_{ij} \alpha_j$ is parallelized using up to 32 processing units. Using fixed-point arithmetic (8.8 to 16.13 formats), classifiers achieved performance comparable to the classifiers with floating-point training. The systems was implemented on Xilinx Virtex-II platform, with 19 to 35 MHz clock rate, depending on word width.

Pedersen et al. [5] have experimented with speeding up classifying with and training SVMs. Their focus was on accelerating scalar product computation by introducing hardware multiply-accumulate unit (MAC). The program was run on a hardware Java virtual machine [6] with the MAC unit attached to it. The system was implemented on Altera Cyclone platform, with 100 MHz clock rate. Speed-up achieved by using MAC unit was up to 59.8 % for training, and up to 6.5 % for classification.

Cadambi et al. in [7] present and advanced architecture of SVM training coprocessor. They use fixed-point arithmetic in 16.16 format, and on two of the datasets in 16.4 format. The system consists of a computer that executes training algorithm and an FPGA coprocessor for scalar product computation. Coprocessor contains 128 processing elements, which execute multiplication and accumulation, organized in 4 clusters, each containing 4 fields of 8 elements. Coprocessor uses local memory to store training set vectors, and uses buffers to enable the computation to run simultaneously with data transfer to and from the computer. Buffers are used to hide the data transfer latency and prevent stalling the coprocessor. The system was implemented on Xilinx Vitex-5 platform. Training time was compared to scalar and vector software implementations running on dual-core 2.2 GHz Opteron CPU. Speed-ups were 18.2× compared to scalar, and 6.5× compared do vector implementation.

Cao el al. in [8] present an implementation developed for Keerthi's training algorithm [9]. Keerthi introduces error matrix:

$$e_i = \sum_j a_j y_j \, \mathrm{K}(\mathbf{x}_i, \mathbf{x}_j) - y_j \qquad (2)$$

which is used for correction of parameters α_j, and b. To find the lower and upper bound of parameter b, a comparison is made over all e_i. This two parts of the algorithm—computation and comparison of e_i—are parallelized by means of error cache update (ECU) units. ECUs implement computation of error matrix elements and tracking of upper and lower bound of parameter b. ECU contains Gaussian kernel look-up tables, and MAC units for *ei* computation. Additionally, each ECU has its own local memory used for storing training vectors and temporary parameter values. The authors have investigated classifier performance achieved with different combinations of word widths and sizes of Gaussian kernel look-up table. Classifier with 16-bit arithmetic and table size of 1024 values, achieved classification accuracy equal to that of reference software classifier with floating-point arithmetic. The system was implemented on Xilinx Virtex-4 platform, with 75 MHz clock rate.

Papadonikolakis and Bouganis in [10] present an architecture that takes into account attribute data ranges of training datasets. They classify datasets as homogenous, in which dynamic ranges of attributes are equal, and heterogeneous, in which attributes have highly differing dynamic range, They identify evaluation of kernel functions as the most time consuming part of the algorithm and implement this functionality in hypertiles. Hypertiles implement scalar product using parallel fixed-point multipliers and a pipelined adder tree, followed by a floating-point kernel processor. For heterogeneous datasets each individual multiplier's arithmetic format is adapted to its input data format in order minimize its resource usage and to make the hypertile as compact as possible. The training vectors are stored in FPGA's local memory. Their approach requires training set analysis before implementing the hypertiles in order to extract attribute's data ranges. The system was implemented on Altera Straix III platform.

Wang el al. in [11] present a least squares SVM [12] implementation targeted for online SVM training in embedded systems. They partition the system into static and reconfigurable part. The static part contains embedded CPU, memory controllers and peripherals required for algorithm execution and communication with the host computer. They identify two parts of LS-SVM training algorithm as the most time consuming: the kernel matrix formulation and the least square problem solution, which are implemented as the reconfigurable parts of the system. The reconfigurable parts were loaded into the FPGA when the algorithm required them. The kernel matrix formulation is implemented with 8 kernel processing elements working in parallel. The least squares problem solver implements Cholesky matrix decomposition using 8 parallel processing elements for computing scalar product. The algorithm FPGA's external memory was used for storing intermediate results. The entire design uses single precision floating-point arithmetic. The system was implemented on Xilinx Virtex-5 platform with 150 MHz clock rate. Performance was compared to 2.93 GHz Xeon computer. Achieved speedups were in range from 6.02× to 218.45× for 512 to 8192 training samples.

C. K-means clustering

FPGA implementations of k-means clustering are primarily focused on image and video processing applications. Research is for the most part focused on computation of distance metric, which is the most computationally intensive part of the algorithm.

Estlick et al. [13] considered two modifications of the algorithm which could increase the available parallelism: alternative distance metrics, and smaller word width of input vectors. Alternative metrics were experimentally evaluated and the Manhattan distance—$\sum_i |x_i - \mu_i|$—was found to be the most suitable with respect to quality of

clustering and complexity of implementation. Experimentally was demonstrated that input data word width can be significantly decreased without adversely affecting clustering quality. The implementation used Manhattan distance metric, without word width decrease. FPGA uses two private external memories: one for storing pixel values, and the other for storing clustering results. The system is organized in a pipeline which executes distance computation, and finding minimum distance and associated cluster. Software that runs on host computer loads the pixels into the FPGA memory, and sets new cluster center values for each iteration. The system was implemented on Xilinx Virtex platform, with 50 MHz clock rate. The achieved speed-up was 200× compared to software implementation executed on a 500 MHz Pentium III computer.

Gokhale et al. [14] implemented k-means clustering for segmenting hyperspectral images. Their system was implemented on Altera Excalibur platform, which contains Altera Apex FPGA and an ARM CPU. They used two implementations: one with ARM CPU, and one with NIOS soft-core CPU. In the paper they demonstrate development of the system through several steps which illustrate influences of communication between CPU and FPGA, granularity of processing unit, and use of direct memory access (DMA) while executing the algorithm. Algorithm was implemented in several iterations, starting with completely software one and replacing the part of code that requires the most time to execute with a hardware unit. The final implementation uses hardware to execute computation of distances, uses DMA, and writes clustering results into a buffer where they are kept until read by the CPU. Distance computation unit contains 32 processing elements which work in parallel. The system runs on 33 MH clock rate. Speed-up achieved was 11.8×, compared to a 1 GHz Pentium III computer.

Wang and Leeser [15] experiment with adding a hardware floating-point division unit into the hardware implementation of the algorithm. They and a cluster association structure to the distance calculation and minimum distance tracking from [13]. They added accumulators that compute pixel sum for each cluster, and counters that track number of pixels in each cluster. New cluster center is computed by dividing accumulator value with counter value, which gives the mean of the cluster. Accumulators and cluster centers are working with fixed-point arithmetic. Hence the divider has fixed-point/floating-point conversion on its inputs and output. The system was implemented on Xilinx Virtex-II platform. Three implementations were compared: completely software, FPGA with division on CPU, and FPGA with division on FPGA. The Program was executed on a 3.2 GHz Pentium 4 computer. Compared to completely software implementation, for 50 iterations of algorithm a 11× speed-up was achieved, and for 1000 iterations a 174× speed-up was achieved. Differences in execution time for FPGA implementations with division on CPU and division on FPGA were negligible. The reason is that compared to distance computation, the division takes up a very small amount of execution time.

Sageusa and Maruyama [16], [17] implement real time k-means clustering for color images. The standard clustering procedure with Euclidian distance metric is augmented with kd-tree [18] filter which is used for finding cluster centers candidates for each pixel. Distance computation is then carried out only for this small set of centers. Kd-tree is implemented using memory banks. It's capable of finding centers for 4 pixels in parallel, and outputs 24 center candidates for each pixel. The system is realized as a three-stage pipeline. First stage contains the kd-tree, second stage implements square-of-distance computation, and third stage the cluster center for each pixel is found. Square-of-distance computation units are also used for kd-tree construction, which is carried out at the every iteration. The system was implemented on Xilinx Virtex-II platform, with 66 MHz clock rate. Performance was evaluated by clustering color image pixels into 256 clusters. Achieved performance for 512×512 and 640×480 images was at least 30 fps, and 20 to 30 fps for 768×512 images.

Covington et al. [19] use k-means clustering for document clustering. Documents are represented by vectors, where each dimension represents number of word occurrences in the document. Similar words are counted in the same dimension. Distance metric used is "cosine-theta" distance:

$$\cos\theta = \frac{\mathbf{x} \cdot \boldsymbol{\mu}}{|\mathbf{x}||\boldsymbol{\mu}|} \tag{3}$$

Words are translated into a 4000-dimension feature space using a hash based method. Document vectors and cluster centers are stored in external memory attached to the FPGA. The system contains three key modules. First module realizes distance computation, and consists of a square root, a scalar product, and a division unit. The module is complex and outputs one result every 294 clock cycles. One distance computation module is instantiated for each cluster center. Second module compares distances and assigns input vector to cluster with nearest center. Third module computes new cluster centers. The system performance was compared to a 3.6 GHz Xeon CPU. On Xilinx Virtex-E platform the system works with 80 MHz clock rate and achieves a 26× speed-up. On Xilinx Virtex-4 platform the system works with 250 MHz clock and achieves a 328× speed-up.

Nagarajan et al. [20] are taking a more general approach with intention to define patterns and models common to many algorithms, i.e. design patterns. The idea is that knowing and using these patterns will aid in system design. In their work they present a review of common communication and computing patterns. They begin hardware development with probability density function (PDF) approximation problem. By using structural similarities between algorithms they adapt the PDF approximation hardware to implement correlation computation, and k-means clustering, instead of implementing them from scratch. Implemented k-means clustering hardware is a three-stage pipeline consisting of: (a) square of distance computation, (b) finding minimum distance, and (c) cluster center update. The first two stages are similar to Saegusa and Maruyama's [17] implementation. The system was implemented and tested on Xilinx Virtex-4 platform. Achieved speed-up was 3.2×,

compared to a 3.2 GHz Xeon computer. The authors find the cause of small speed-up in low communication bandwidth between CPU and FPGA.

Hussain et al. [21] present an FPGA k-means clustering implementation for clustering microarray data. All parts of the algorithm are implemented in hardware, and distance computation is parallelized. Their implementation uses fixed-point arithmetic adapted to microarray data ranges. They use 5.10 format for distances, and 15.10 format for accumulators. The number of clusters is fixed to 8, hence 8 cluster distance units are used in parallel. Input data is stored in on-chip RAM. Several cores can be fit on single FPGA to perform clustering on different datasets in parallel. The system was implemented on Xilinx Virtex-4 platform, with 126 MHz clock rate for single core, and 124 MHz clock rate for five cores implementation. Achieved speed-ups were 10.3× for single core, and 51.7× for five cores implementation, compared to 3 GHz Core 2 Duo computer.

Singaraju and Chandy [22] implement k-means clustering on FPGA based network switches. The architecture allows data processing to be performed on the data as it flows through tha network in form of Ethernet packets. They implement all parts of the algorithms on the FPGA, and use floating point arithmetic for all computation. They parallelize distance computation, and finding minimum distance. Since used arithmetic operators are heavily pipelined, they use data element interleaving to hide the latencies. For single FPGA their system supports cluster parallelism, distances from multiple cluster centers are computed simultaneously for single data input. For multiple FPGAs, in addition to cluster parallelism, the system also supports data parallelism, where each FPGA operate on separate data points. Single FPGA is limited to up to 8 clusters in parallel. The system was implemented on Xilinx Virtex-II platform. For single FPGA, achieved speedups were up to 10× compared to Opteron 1.8 GHz computer. For multiple FPGAs performance was compared to parallel software implementation with equivalent number of nodes. Achieved speed-ups were up to 9×.

IV. Analysis of Implementations

Following features common to most implementations can be highlighted from this survey of FPGA based hardware accelerators.

For all implementations, the design process begins with algorithm analysis. Algorithm's hotspots are identified, and evaluated for possible gains in speed or execution time from implementing them in hardware. Most often, the hotspots are a short sequence of instructions that are looped through many times.

FPGA's clock rate is much lower than CPU's, usually one to two orders of magnitude lower. This is offset by exploiting available parallelism. FPGA can simultaneously execute several hundred to several thousand operations, depending on type of operation and its implementation.

FPGA implementations are usually pipelined. Apart from the additional "depth" parallelism provided by pipelining, separating the process into several smaller steps enables faster clock rate, and faster execution of algorithm. The algorithm is often rearranged or modified to enable more efficient pipelining.

Use of floating-point arithmetic is usually avoided. Floating-point arithmetic units are more complex and consume more FPGA resources than fixed-point arithmetic units. Higher per-unit resource consumption means less available parallelism because fewer units can be implemented in a single FPGA device. This is the main reason for using fixed-point arithmetic. Due to this constraint, a numerical analysis of algorithm is required to find sufficient range and precision of fixed-point number representation which will ensure correct results.

Only a few implementations perform the entire algorithm in hardware. In general, implementations are carefully partitioned into software and hardware part, and systems completely implemented in FPGA usually use a soft core CPU which runs the main program. This is a consequence of following factors: (a) control of algorithm flow is much easier to implement in software than by designing a hardware finite state machine for this function, and (b) parts of algorithm that execute infrequently can be left in software without adverse impact on overall performance.

A frequent problem in hardware implementations is communication with CPU and memory. In some cases, using hardware acceleration can actually decrease performance because the cost of moving data can be larger than the gain from faster hardware execution [14]. This problem can be tackled in several ways. First, by using direct memory access (DMA) where the CPU is taken out of the data transfer loop. The CPU then controls the data flow only by issuing commands for data transfer. Additional performance is gained by providing significant amount of private local memory to the FPGA, which then eliminates the need for frequent communication with shared main memory. And finally, data transfer latency can be hidden by using buffers, which enable processing of one block of data while the next block is being transferred.

Earlier implementations rarely used multiplication in FPGA since it was fairly expensive in terms of resources. Newer FPGAs families targeted for DSP applications include high-speed hardware multipliers and MAC units. Thanks to this development, later implementations benefit from using multiplication in FPGA hardware.

V. Conclusion

FPGA platform can be used as accelerator in data mining processes. It has great potential for use in data mining applications and computing in general, and especially in embedded systems. FPGA's flexibility and programmability enables implementation of optimal computer architecture for each specific task. From this literature survey we conclude that FPGA platform performs well as a hardware accelerator.

Results in this research area are significant, but we also noticed some important issues that prevent wider use of FPGAs in general purpose computing. First and most

significant is lack of mature high-level development tools. It is still very difficult to develop a hardware accelerator without knowledge of low-level details of hardware development, especially if full potential of FPGA is to be utilized. Second issue is a high cost of advanced FPGA platforms suitable for implementing data mining application. These issues currently limit the use of FPGA platforms to applications in which development effort and cost can be justified with gains in performance.

With development of high-level languages and synthesis tools, and development of hardware-software co-design methodology, designing for FPGAs could become much easier. Together with advancements in FPGA technology making the devices larger in terms of logic resources and providing more high-performance arithmetic blocks, the FPGA platform is expected to become more frequently used in data mining applications, as well as in computing in general.

REFERENCES

[1] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with GPUs and FPGAs," in *Proc. SASP*, 2008, pp. 101-107.

[2] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z. H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowl. Inf. Syst.*, vol. 14, no. 1, pp. 1-37, 2008.

[3] R. Narayanan, D. Honbo, G. Memik, A. Choudhary, and J. Zambreno, "An FPGA implementation of decision tree classification," in *Proc. DATE*, 2007.

[4] D. Anguita, A. Boni, and S. Ridella, "A digital architecture for support vector machines: theory, algorithm and FPGA imlementation," *IEEE Tras. Neural Netw.*, vol. 14, no. 5, Sept. 2003.

[5] R. Pedersen and M. Schoeberl, "An embedded support vector machine," in *Proc. WISES*, 2006

[6] M. Schoeberl, *JOP: A java optimized processor for embedded real-time systems*, PhD thesis, Vienna University of Technology, 2005.

[7] S. Cadambi, I. Durdanovic, V. Jakkula, M. Sankaradass, E. Cosatto, S. Chakradhar, and H. P. Graf, "A massively parallel FPGA-based coprocessor for support vector machines," in *Proc. FCCM*, 2009, pp. 115-122.

[8] K. Cao, H. Shen, and H. Chen, "A parallel and scalable digital architecture for training support vector machines," *Journal of Zhejiang University - Science C*, vol. 11, no. 8, pp. 620-628, 2010.

[9] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, "Improvements to Platt's SMO algorithm for SVM classifier design," *Neural Computation*, vol. 13, no. 3, pp. 637-649, 2001.

[10] M. Papadonikolakis and C.-S. Bouganis, "A Heterogeneous FPGA Architecture for Support Vector Machine Training," in *Proc. FCCM*, 2010, pp. 211-214.

[11] S. Wang, Y. Peng, G. Zhao, and X. Peng, "Accelerating on-line training of LS-SVM with run-time reconfiguration," in *Proc. ICFPT*, 2011, pp. 1-6.

[12] J. A. K. Suykens and J. Vandewalle, "Least Squares Support Vector Machine Classifiers," *Neural Processing Letters*, vol. 9, no. 3, pp. 293-300, 1999.

[13] M. Estlick, M. Leeser, J. Theiler, and J. J. Szymanski, "Algorithmic transformations in the implementation of k-means clustering on reconfigurable hardware", in *Proc. ACM FPGA*, 2001, pp. 103-110.

[14] M. Gokhale, J. Frigo, K. McCabe, J. Theiler, C. Wolinski, and D. Lavenier, "Experience with a hybrid processor: K-means clustering," *J. Supercomputing*, vol. 26, no. 2, pp. 131-148, 2003.

[15] X. Wang and M. Leeser, "K-means clustering for multispectral images using floating-point divide," in *Proc. FCCM*, 2007, pp. 151-159.

[16] T. Maruyama, "Real-time k-means clustering for color images on reconfigurable hardware," in *Proc. ICPR*, 2006, pp. 816-819.

[17] T. Saegusa and T. Maruyama, "An implementation of real-time k-means clustering for color images," *J. Real-Time Image Proc*, vol. 2, no 4, pp. 309-318, 2007.

[18] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: analysis and implementation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 881–892, 2002.

[19] G. A. Covington, C. L. G. Comstock, A. A. Levine, J. W. Lockwood, and Y. H. Cho, "High speed document clustering in reconfigurable hardware," in *Proc. FPL*, 2006, pp. 411-417.

[20] K. Nagarajan, B. Holland, A. D. George, K. C. Slatton, and H. Lam, (2009, Jan.). Accelerating machine-learning algorithms on FPGAs using pattern-based decomposition. *J. Sign. Process. Syst.* [Online] Available: http://dx.doi.org/10.1007/s11265-008-0337-9

[21] H. M. Hussain, K. Benkrid, H. Seker, and A. T. Erdogan, "FPGA implementation of K-means algorithm for bioinformatics application: An accelerated approach to clustering Microarray data," in *Proc. AHS*, 2011, pp. 248-255.

[22] J. Singaraju and J. A. Chandy, "Active Storage Networks for Accelerating K-Means Data Clustering," In *Proc. ARC*, 2011, pp. 102-109.