An FPGA Implementation of Decision Tree Classification

Ramanathan Narayanan Daniel Honbo Gokhan Memik Alok Choudhary Joseph Zambreno[†]

Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208, USA
{*ran310, dkh301, memik, choudhar*}*@eecs.northwestern.edu*

[†]Electrical and Computer Engineering
Iowa State University
Ames, IA 50011, USA
zambreno@iastate.edu

Abstract

Data mining techniques are a rapidly emerging class of applications that have widespread use in several fields. One important problem in data mining is Classification, which is the task of assigning objects to one of several predefined categories. Among the several solutions developed, Decision Tree Classification (DTC) is a popular method that yields high accuracy while handling large datasets. However, DTC is a computationally intensive algorithm, and as data sizes increase, its running time can stretch to several hours. In this paper, we propose a hardware implementation of Decision Tree Classification. We identify the compute-intensive kernel (Gini Score computation) in the algorithm, and develop a highly efficient architecture, which is further optimized by reordering the computations and by using a bitmapped data structure. Our implementation on a Xilinx Virtex-II Pro FPGA platform (with 16 Gini units) provides up to $5.58\times$ *performance improvement over an equivalent software implementation.*

1 Introduction

Data mining is the process of transforming raw data into actionable information that is nontrivial, previously unknown and is potentially valuable to the user. Data mining techniques are used in a variety of fields including marketing and business intelligence, biotechnology, multimedia, and security. As a result, data mining algorithms have become increasingly complex, incorporating more functionality than in the past. Consequently, there is a need for faster execution of these algorithms, which creates ample opportunities for algorithmic and architectural optimizations.

Classification is an important problem in the field of data mining. A classification problem has an input dataset called the training set which consists of example records with a number of attributes. The objective of a classification algorithm is to use this training dataset to build a model which can then be used to assign unclassified records into one of the defined classes [6]. Decision Tree Classification (DTC) is a simple yet widely-used classification technique. In DTC, inferring the category (or class label) of a record involves two steps. The first task involves building the decision tree model using records for which the category is known beforehand. The decision tree model is then applied to other records to predict their class affiliation.

Decision trees are used for various purposes, such as detecting spam e-mail messages, categorizing cells as malignant or benign based upon the results of MRI scans, and classifying galaxies based on their shapes. They yield comparable or better accuracy when compared to other models such as artificial neural networks, statistical, and genetic models. Decision tree-based classifiers are attractive because they provide high accuracy even when the size of the dataset increases [4].

Recent advances in data extraction techniques have created large data sets for classification algorithms. However conventional classification techniques have not been able to scale up to meet the computational demands of these inputs. Hardware acceleration of classification algorithms is an attractive method to cope with the increase in execution times and can enable algorithms to scale with increasingly large and complex data sets. This paper analyzes the DTC algorithm in detail and explores techniques for adapting it to a hardware implementation. We first isolate the compute-intensive kernel in the decision tree induction process, called Gini score calculation, and then rearrange the computations in order to reduce hardware complexity. We also use a bitmapped index structure for storing class IDs that minimizes bandwidth requirements of the DTC architecture. To the best of our knowledge, this is the first published hardware implementation of a classification algorithm.

We implement our design on an FPGA platform, as their reconfigurable nature provides the user ample flexibility, al-

This work was supported in part by the National Science Foundation (NSF) under grants NGS CNS-0406341, IIS-0536994, CNS-0551639, CCF-0621443, CCF-0546278, and NSF/CARP ST-HEC program under grant CCF-0444405, and in part by Intel Corporation.

978-3-9810801-2-4/DATE07 © 2007 EDAA

lowing for customized architectures tailored to a specific problem and input data size. Another property of FPGAs that is important for our design is that they allow the design to scale upward easily as process technology allows for ever-larger gate counts. Overall, our system is able to achieve a speedup of $5.58\times$ as compared to software implementations on the experimental platform we selected.

The remainder of this paper is organized as follows. Section 2 contains the related work regarding hardware implementations of data mining algorithms. Section 3 describes the DTC algorithm and Gini score calculation in detail. A description of our architecture and techniques used to accelerate the Gini score computation are given in Section 4. Section 5 contains implementation details and results, followed by a summary of the overall effort in Section 6.

2 Related Work

There has been prior research on hardware implementations of data mining algorithms. However, to the best of our knowledge, ours is the first attempt to implement decision tree classification in hardware. In [5] and [9], k-Means clustering is implemented using reconfigurable hardware. Baker and Prasanna [2] use FPGAs to implement and accelerate the Apriori [1] algorithm, a popular association rule mining technique. They develop a scalable systolic array architecture to efficiently carry out the set operations, and use a 'systolic injection' method for efficiently reporting unpredicted results to a controller. In [3], the same authors use a bitmapped CAM architecture implementation on a FPGA platform to achieve significant speedups over software implementations of the Apriori algorithm. Compared to our work, these implementations target different classes of data mining algorithms.

Several software implementations of DTC have been proposed (e.g., SPRINT [8], ScalParC [7]), which use complex data structures for efficient implementation of the splitting and redistribution process. These implementations focus on parallelizing DTC using coarse-grain parallelization paradigms. Our approach is complementary to these methods, as we tend to use a fine-grained approach coupled with reconfigurable hardware to improve performance.

3 Introduction to Decision Tree Classification

Formally, the classification problem may be stated as follows. We are given a training dataset consisting of several records. Each record has a unique record ID and is made up of several fields, referred to as attributes. Attributes may be *continuous*, if they have a continuous domain, or *categorical* if their domain is a finite set of discrete values. The *classifying attribute* or *class ID* is a categorical attribute. The DTC problem involves developing a model that allows prediction of the class of a record in terms of its remaining attributes.

A decision tree model consists of internal nodes and leaves. Each of the internal nodes has a splitting decision and splitting attribute associated with it. The leaves have a class label assigned to them. Building a decision tree model from a training dataset involves two phases. In the first phase, a splitting attribute and a split index are chosen. The second phase involves splitting the records among the child nodes based on the decision made in the first phase. This process is recursively continued until a stopping criterion is met. At this point, the decision tree can be used to predict the class of an incoming record, whose class ID is unknown. The prediction process is relatively straightforward: the classification process begins at the root, and a path to a leaf is traced by using the splitting decision at each internal node. The class label attached to the leaf is then assigned to the incoming record.

Choosing the split attribute and the split position is a critical component of the decision tree induction process. In various optimized implementations of decision tree induction [8, 7], the splitting criteria used is to minimize the Gini index of the split.

3.1 Computing the Gini Score

The Gini score is a mathematical measure of the inequality of a distribution. Calculating the Gini value for a particular split index involves computing the frequency of each class in each of the partitions. The details of the Gini calculation can be demonstrated by the following example. Assume that there are R records in the current node. Also, assume that there are only 2 distinct values of class IDs, hence there can be only 2 partitions into which the parent node can be split. The algorithm iterates over the R records and computes the frequencies of records belonging to distinct partitions. The Gini index for each partition is then given by $Gini_i = 1 - \sum_{j=0}^{1}(\frac{R_{ij}}{R_i})^2$, where R_i is the number of records in partition i, among which R_{ij} records bear the class label j. The Gini index of the total split is then calculated by using the weighted average of the Gini values for each partition, i.e.,

$$Gini_{total} = \sum_{i=0}^{1} \frac{R_i}{R} \cdot Gini_i \qquad (1)$$

The values of R_{ij} are stored in a count matrix. The partitions are formed based on a splitting decision, which depends on the value of a particular attribute. Each attribute is a possible candidate for being the split attribute. Hence this process of computing the optimal split has to be carried out over all attributes. Categorical attributes have a finite number of distinct class ID values, so there is little benefit in optimizing Gini score calculation for such attributes.

However, the computation cost of the minimum Gini score for a continuous attribute is linear in the number of

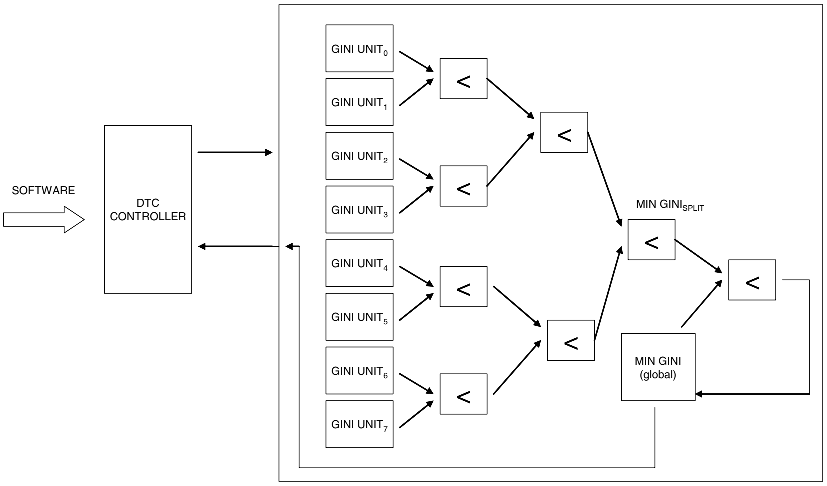

Figure 1. Architecture for Decision Tree Classification

records. In the case of a continuous attribute A, it is assumed that two partitions are formed, based on the condition $A < v$, for some value v in its domain. It is initially assumed that one of the partitions is empty, and the second partition contains the R records. At the end of the Gini calculation for a particular split value, the split position is moved down one record, and the count matrix is updated according to the class ID of the record at the split position. The Gini value for the next split position is calculated and compared to the present minimum Gini value.

Therefore, a linear search is made for the optimum value of v, by evaluating the Gini score for all possible splits. This process is repeated for each attribute, and the optimum split index over all the attributes is chosen. Therefore, the total complexity of Gini calculation is $O(|R| * |A|)$, where $|R|$ and $|A|$ represent the number of records and number of attributes, respectively. Since each attribute needs to be processed separately in linear time, it becomes necessary to maintain a sorted list of records for each attribute. This entails vertically partitioning the record list into several attribute lists, which consist of a record ID and attribute value. Each attribute list is sorted, thus introducing a random order among the records in various attribute lists.

Previous work has shown that the largest fraction of the execution time of representative implementations is spent in the split determining phase [10]. For example, ScalParC [7], which uses a parallel hashing paradigm to efficiently map record IDs to nodes, spends over 40% of its time in the Gini calculation phase. As the number of attributes and records increase, it is expected that the importance of Gini calculation will increase. In this paper, we design an architecture that allows for fast calculation of the split attribute and split index. By using hardware to implement this operation, we aim to significantly reduce the running time of the Gini calculation process, and in turn, the decision tree induction process.

4 Hardware Architecture

Our goal is to design an architecture that will compute the Gini score using minimal hardware resources, while achieving significant speedups. The bottleneck in Gini calculation is the repetition of the process for each of the attributes. Therefore, it is clear that an architecture for DTC should allow for the handling of multiple attributes simultaneously. Our architecture consists of several computation modules, referred to as 'Gini units', that perform Gini calculation for a single attribute. In our generic architecture we assume that we have nG Gini units. If $nG > |A|$, then the entire Gini computation can be completed in one phase. Otherwise $\lceil \frac{|A|}{nG} \rceil$ runs are required to compute the minimum Gini index for a set of records. The number of Gini units nG that can be accommodated depends on the hardware platform.

The high-level DTC architecture is presented in Figure 1. There is a DTC controller component that interfaces with the software and supplies the appropriate data and signals to the Gini units. The architecture functions as follows: when the software requests a Gini calculation, it supplies the appropriate initialization data to the DTC controller. The DTC controller then initializes the Gini units. The software then transmits the class ID information required to compute the Gini score in a streaming manner to the DTC controller. The format of this data can be tweaked to optimize performance, which will be discussed in the following sections. The DTC controller then distributes the data to the Gini units, which perform the Gini score computation for that level. At the end of each cycle, the Gini score calculated at that split is compared to the scores obtained at the other attributes using a tree-like structure of hardware comparators. The minimum

Gini value among all attributes at that cycle is then compared to the global minimum Gini score. If the current Gini score is less than the global minimum, the global minimum is updated to reflect the current split point and split attribute. This process is carried out until all the records have been streamed through the Gini units. The global minimum value at that stage is then transmitted to the DTC controller, and subsequently to the software. If $nG < |A|$, several runs of the above process are required to obtain the split value and the split attribute.

4.1 Bitmap Generation

There is ample scope for optimization of the Gini computation architecture. Commonly, the class ID assumes only 2 values, '0' and '1'. This allows us to optimize the data transfer process to the Gini units. In software, the class IDs are stored in an integer data type. Transmitting the class IDs to the Gini units in the raw form would be very inefficient, as in each input cycle, a total of $|A| \cdot S$ bytes of data would have to be transmitted, where S represents the size of the data type used by the software implementation to store the class IDs. In hardware, a single bit is sufficient to represent the class ID. Thus only $|A|$ bits are required to represent a set of class ID inputs for a single cycle. A bitmap representation is ideally suited to represent the data in this format. Therefore we modify the software to generate bitmaps of class ID information. It should be noted that our architecture can be easily extended to support a wider range of class ID values.

Apart from the initial cycle, this procedure is carried out each time the records are distributed among the child nodes, after the split position and attribute have been decided. This step has to be performed irrespective of the data representation format used, and hence the additional overhead caused due to bitmap generation is minimal. The size of the bitmaps generated can be adjusted to equal the number of physical Gini units available. When the DTC controller receives the bitmap containing the class IDs, it distributes them among the Gini units. Each Gini unit then uses the class information to compute the Gini score at that stage.

4.2 Optimizing the Gini Unit

From a hardware perspective, we would like to minimize the number of computations and their complexity while calculating the Gini score. An implementationion of the hardware in which the Gini score calculation is unaltered will be very complex and inefficient. A key observation is that the absolute value of the Gini score computed is irrelevant to the algorithm. It is only the split value and split attribute that are required. Therefore, we attempt to simplify the Gini computation to require minimal hardware resources, while generating the same value of split position and split attribute generated as earlier. Considering our assumption of only two distinct values for the class ID, the Gini score computation can be simplified. First, we rewrite equation 1 for two class IDs as follows:

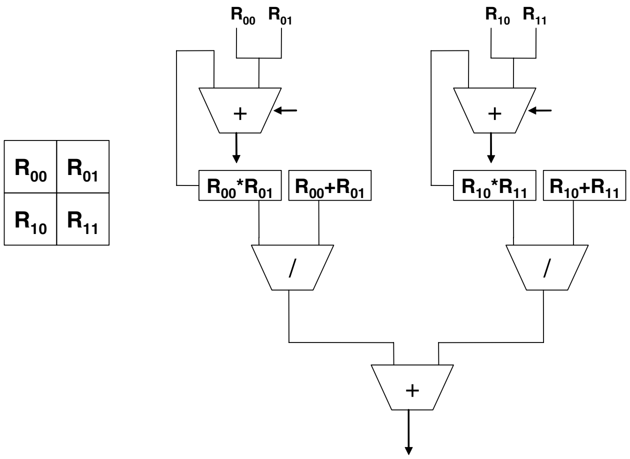

Figure 2. Count matrix / Gini unit architecture

$$gini_0 = 1 - \frac{R_{00}}{R_0}^2 - \frac{R_{01}}{R_0}^2 \tag{2}$$

$$gini_1 = 1 - \frac{R_{10}}{R_1}^2 - \frac{R_{11}}{R_1}^2 \tag{3}$$

$$gini_{total} = \frac{R_0}{R_0 + R_1} \cdot gini_0 + \frac{R_1}{R_0 + R_1} \cdot gini_1 \tag{4}$$

If the Gini unit were required to compute the above expression, it would require 6 multipliers, 6 dividers and 7 adders, severely limiting our ability to accommodate multiple Gini units on the hardware platform. It can be seen that

$$gini_0 = \frac{{R_0}^2 - {R_{00}}^2 - {R_{01}}^2}{{R_0}^2} \tag{5}$$

$$gini_1 = \frac{{R_1}^2 - {R_{10}}^2 - {R_{11}}^2}{{R_1}^2} \tag{6}$$

By definition,

$$R_0 = R_{00} + R_{01}$$

and

$$R_1 = R_{10} + R_{11} \tag{7}$$

Therefore, the equations 5 and 6 can be rewritten as

$$gini_0 = \frac{2 \cdot R_{00} \cdot R_{01}}{{R_0}^2} \tag{8}$$

$$gini_1 = \frac{2 \cdot R_{10} \cdot R_{11}}{{R_1}^2} \tag{9}$$

$$gini_{total} = \frac{2 \cdot R_{00} \cdot R_{01}}{R_0 \cdot (R_0 + R_1)} + \frac{2 \cdot R_{10} \cdot R_{11}}{R_1 \cdot (R_0 + R_1)} \tag{10}$$

Case "0"	Case "1"
$R_{00} = R_{00} + 1;$	$R_{01} = R_{01} + 1;$
$R_{10} = R_{10} - 1;$	$R_{11} = R_{11} - 1;$
$[R_{00}+R_{01}] = [R_{00}+R_{01}] + 1$	$[R_{00}+R_{01}]=[R_{00}+R_{01}] + 1$
$[R_{10}+R_{11}] = [R_{10}+R_{11}] - 1$	$[R_{10}+R_{11}] = [R_{10}+R_{11}] - 1$
$[R_{00}*R_{01}] = [R_{00}*R_{01}] + R_{01}$	$[R_{00}*R_{01}] = [R_{00}*R_{01}] + R_{00}$
$[R_{10}*R_{11}] = [R_{10}*R_{11}] - R_{11}$	$[R_{10}*R_{11}] = [R_{10}*R_{11}] - R_{10}$

Figure 3. Gini unit operations

We know that R_0+R_1 represents the total number of records and is a constant for all split positions and split attributes. Hence a simplified computation that is equivalent can be formulated as

$$gini'_{total} = \frac{R_{00} \cdot R_{01}}{R_{00} + R_{01}} + \frac{R_{10} \cdot R_{11}}{R_{10} + R_{11}} \qquad (11)$$

The above equation represents a value, which when minimized, will give the same split index and split attribute as that of the original Gini computation. This design can be improved upon by observing that in each cycle, depending on whether the incoming class ID is '0' or '1', only one of R_{00} or R_{01} is incremented by one. Similarly, only one of R_{10} or R_{11} decreases by a value of 1 in each cycle. Furthermore, the values of $R_{00} \cdot R_{01}$ can be computed easily without using a multiplier. This stems from the fact that the product will increase by only a value of either R_{00} or R_{01} in each cycle, depending on the incoming class ID. Thus, both the products $R_{00} \cdot R_{01}$ and $R_{10} \cdot R_{11}$ can be computed using a register and an adder/subtractor, instead of using a multiplier. It should be noted that the initial values of R_{10}, R_{11}, $R_{10} + R_{11}$ and $R_{10} \cdot R_{11}$ are computed using software, and the DTC unit loads these values into the gini units before the start of every new iteration.

The final architecture of each Gini unit, after the application of the above modifications, can be seen in Figure 2. Also the operations to be carried out when the incoming class ID is either '0' or '1' are detailed in the Figure 3. It can be seen that the complex Gini computation has been simplified to a great extent, and can be performed using minimal hardware resources.

5 Implementation and Results

The DTC architecture was implemented on a Xilinx ML310 board which is a Virtex-II Pro-based embedded development platform. It includes an Xilinx XC2VP30 FPGA with two embedded PowerPC processors, 256 MB DDR DIMM, 512 MB compact flash card, PCI slots, ethernet and standard I/O on an ATX board. The XC2VP30 FPGA contains 13696 slices and 136 Block RAM modules. We used Xilinx XPS 8.1i and ISE 8.1i softwares to implement our architecture on the board.

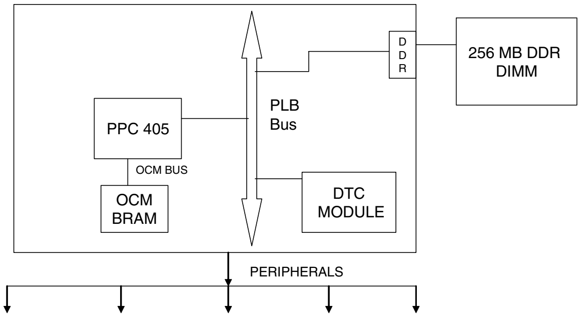

Figure 4. Experimental setup

Figure 4 shows the experimental setup for the DTC architecture. The figure does not show the entire peripheral components supported by the XC2VP30 FPGA, only those relevant to the design. The DTC unit is implemented as a custom peripheral which is fed by the PowerPC. The PowerPC reads in input data stored in DDR DIMM, initializes the DTC component, and supplies class ID data at regular intervals. The OCM BRAM block stores the instructions for the PowerPC operation.

While implementing the design, several tradeoffs were considered. The use of floating point computations complicate the design and increase the area overhead, hence we decided to perform the division operations using only fixed-point integer computations. To verify the correctness of our assumptions, we implemented a version of ScalParC that uses only fixed point values. It was found that the decision trees generated by both the fixed-point and floating-point versions were identical, thus validating our choice of a divider performing fixed point computations. The divider output was configured to produce 32 integer bits and 16 fractional bits, a choice made keeping in mind the size of the dataset and precision required to produce accurate results. The divider was also pipelined in order to handle multiple input class IDs at the same time.

We used the above-mentioned tools to measure the area occupied and clock frequency of our design. Due to the inherent parallelism in the DTC module, it can take a new set of class ID input every cycle. However, we were limited by both the bus width of the PowerPC platform and the maximum number of Gini units that could fit upon the FPGA device (limit of 16 for the XC2VP30). The DTC module is designed to take as input a maximum of 32 bits per cycle.

Table 1 shows the variation in area utilization and performance with varying number of Gini computation units. As expected, the required area increases as the number of Gini units in the design is increased. We have also observed that a major portion of the slices are occupied by the divider units. The area occupied by the dividers may be decreased

nG	N_{slices} (%)	f_{max} (MHz)	Throughput (Gbps)
2	4254 (31%)	102.533	3.28
4	6081 (44%)	102.341	3.27
8	8137 (59%)	100.746	3.22
16	13697 (99%)	100.361	3.21

Table 1. Variation of resource utilization with number of Gini units

by cutting down on the pipeline length, but this will have a detrimental effect on performance. The maximum clock frequency and throughput of the design are, however, stable, thus indicating the scalability of our design when implemented on real hardware. This is expected since all the computations of the Gini units are performed in parallel.

The IBM DataQuest Generator was used to generate the data used in our performance measurements. The Gini calculation was also implemented in software (using C) and run on the PowerPC under identical conditions. The speedup provided by hardware was measured in terms of the ratio of number of cycles taken by the hardware-enabled design to those taken by the software implementation. Figure 5 shows the speedups obtained when the DTC module was tested on the FPGA. The results show significant speedups over software implementations. As expected, the speedup increases with the number of Gini units on board, due to the parallelism offered by additional hardware computation units. The experimental hardware imposed a size limitation of 16 Gini units, which achieves a speedup of $5.58\times$. It would be possible to achieve larger speedups using higher-capacity FPGAs.

Given the fraction of execution time that the Gini score calculation takes in ScalParC [7, 10], the overall speedup of this particular implementation of DTC can be estimated to be $1.5\times$. A direct comparison of our implementation with other existing hardware implementations [2, 3] is difficult since the structure and goals of the underlying data mining algorithms are vastly different.

6 Conclusion

In this paper, we have designed a hardware implementation of a commonly used data mining algorithm, Decision Tree Classification. The Gini score calculation is determined to be the critical component of the algorithm. We have developed an efficient reconfigurable architecture to implement Gini score calculation. The arithmetic calculations required to compute the optimal split point were then simplified to reduce the hardware resources required. The design was implemented on a FPGA platform. The results show that our designed architecture yields up to $5.58\times$ speedup with 16 Gini units, while achieving throughput scalability as the number of Gini units on board increases.

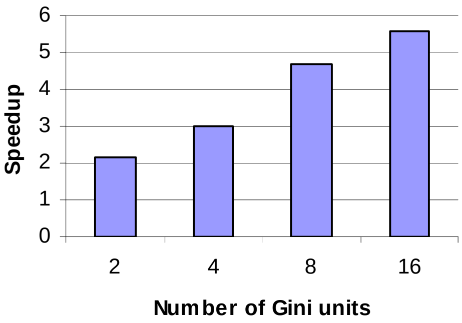

Figure 5. DTC module speedups

References

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo. Fast discovery of association rules. *Advances in Knowledge Discovery and Data Mining*, pages 307–328, 1996.

[2] Z. Baker and V. Prasanna. Efficient hardware data mining with the Apriori algorithm on FPGAs. In *Proc. of the IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, 2005.

[3] Z. Baker and V. Prasanna. An architecture for efficient hardware data mining using reconfigurable computing systems. In *Proc. of the IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, 2006.

[4] J. Catlett. Megainduction: Machine learning on very large databases. Ph.D Thesis, University of Sydney, 1991.

[5] M. Estlick, M. Leeser, J. Szymanski, and J. Theiler. Algorithmic transformations in the implementation of k-means clustering on reconfigurable hardware. In *Proc. of the IEEE Symposium on Field Programmable Custom Computing Machines(FCCM)*, 2001.

[6] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.

[7] M. Joshi, G. Karypis, and V. Kumar. ScalParC: A new scalable and efficient parallel classification algorithm for mining large datasets. In *Proceedings of the 11th International Parallel Processing Symposium (IPPS)*, 1998.

[8] J. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proc. of the Int'l Conference on Very Large Databases (VLDB)*, 1996.

[9] C. Wolinski, M. Gokhale, and K. McCabe. A reconfigurable computing fabric. In *Proc. of the Engineering of Recongurable Systems and Algorithms Conference (ERSA)*, 2004.

[10] J. Zambreno, B. Ozisikyilmaz, J. Pisharath, G. Memik, and A. Choudhary. Performance characterization of data mining applications using MineBench. In *Proc. of the Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, 2006.