HC-CART: A Parallel System Implementation of Data Mining Classification and Regression Tree (CART) Algorithm on a Multi-FPGA System

GRIGORIOS CHRYSOS, PANAGIOTIS DAGRITZIKOS, IOANNIS PAPAEFSTATHIOU, and APOSTOLOS DOLLAS, Technical University of Crete

Data mining is a new field of computer science with a wide range of applications. Its goal is to extract knowledge from massive datasets in a human-understandable structure, for example, the decision trees. In this article we present an innovative, high-performance, system-level architecture for the Classification And Regression Tree (CART) algorithm, one of the most important and widely used algorithms in the data mining area. Our proposed architecture exploits parallelism at the decision variable level, and was fully implemented and evaluated on a modern high-performance reconfigurable platform, the Convey HC-1 server, that features four FPGAs and a multicore processor. Our FPGA-based implementation was integrated with the widely used "rpart" software library of the R project in order to provide the first fully functional reconfigurable system that can handle real-world large databases. The proposed system, named HC-CART system, achieves a performance speedup of up to two orders of magnitude compared to well-known single-threaded data mining software platforms, such as WEKA and the R platform. It also outperforms similar hardware systems which implement parts of the complete application by an order of magnitude. Finally, we show that the HC-CART system offers higher performance speedup than some other proposed parallel software implementations of decision tree construction algorithms.

Categories and Subject Descriptors: C.3 [**SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS**]; H.2.4 [**DATABASE MANAGEMENT**]: Systems Query Processing; H.2.8 [**Database Applications**]: Data Mining; H.3.4 [**Systems and Software**]: Performance *evaluation (efficiency and effectiveness)*

General Terms: Design, Performance, Algorithms

Additional Key Words and Phrases: Decision tree classification (DTC), classification and regression tree (CART), WEKA platform, R project, rpart library, HW architecture, reconfigurable system, high-performance computing

ACM Reference Format:
Chrysos, G., Dagritzikos, P., Papaefstathiou, I., and Dollas, A. 2013. HC-CHART: A parallel system implementation of data mining classification and regression tree (cart) algorithm on a multi-fpga system. ACM Trans. Architec. Code Optim. 9, 4, Article 47 (January 2013), 25 pages.
DOI = 10.1145/2400682.2400706 http://doi.acm.org/10.1145/2400682.2400706

This work is partly funded by "secure, Mobile visual sensor networks ARchiTechure" research project (SMART, project no. 100032), funded within the ARTEMIS Joint Technology Initiative as a Joint Undertaking project between the European Commission, the member states and ARTEMIS Industrial Association (ARTEMISIA) and partly funded by the FASTCUDA project (project no. 286770) implemented within the Seventh Framework Programme and financed by Community Funds. Also, the authors acknowledge the support of the Convey Computer Corp. for their technical support on this work.
Authors' addresses: G. Chrysos (corresponding author), P. Dagritzikos, I. Papaefstathiou, and A. Dollas, Technical University of Crete, Greece; email: gregory.chrysos@gmail.com.

DOI 10.1145/2400682.2400706 http://doi.acm.org/10.1145/2400682.2400706

1. INTRODUCTION

The wide availability of huge amounts of data and the great need for turning such data into useful information created the new scientific area of data mining, which can be considered as a result of the natural evolution of information technology. The database system industry has evolved in the development of data collection, data management (including data storage, data retrieval, and database transaction processing), and advanced data analysis.

Data mining is one of the main processing steps within the Knowledge Discovery from Data (KDD) approach. Its purpose is to extract or "mine" knowledge, e.g., data patterns, from large amounts of data by applying intelligent methods. It consists of six classes of problems: anomaly detection, association rule learning, clustering, classification, regression, and summarization. Data mining results have been used in many application domains ranging from market analysis, fraud detection, and customer retention, to production control and science exploration, bioinformatics, etc. The last decade's data explosion led to the implementation of software- and hardware-based parallel systems for scaling up data mining processing [Bekkerman et al. 2011]. Some huge commercial organizations, like Google, IBM, and Yahoo, offer comprehensive, parallel and scalable platform-based approaches, such as the Hadoop framework [Apache Hadoop 2012] while IBM markets its SPSS software for tolerating high data mining workloads. There are also numerous other commercial organizations, like Facebook, Linkedin, Twitter, and eBay, which work on the Hadoop parallel system [Apache Hadoop]. This framework is widely used for mapping data-intensive applications (including data mining) on large clusters, grids, or multicore platforms. The Hadoop framework implements MapReduce, a computational paradigm where the application is divided into many small fragments of work, each of which may be executed or reexecuted in a distributed manner on any node in the cluster or grid.

Several surveys have been presented with reviews of data mining tools and data mining algorithms. Mikut et al. present a range of existing state-of-the-art data mining algorithms and related tools [Mikut et al. 2011]. According to the Rexer Analytics survey for 2010, which is considered as the largest survey of data mining professionals in the industry, the decision trees, the regression, and the cluster analysis form the core algorithms in the data mining area [Rexer Analytics].

In particular, the most powerful tool for classification is considered to be the Decision Tree Classification (DTC) [Wu et al. 2010], as it offers the ability to break down complex decisions into a collection of simpler decisions, similar to the decision tree structure shown in Figure 1. This provides a solution that is often easier to interpret [Anyanwu et al. 2009]. The DTC is ideal for nonmultivariable classification problems but its performance reduces dramatically in case of multivariable problems. The main drawback of DTC is the difficulty in designing an optimal decision tree that can lead to NP-complete solutions even for simple concepts [Yang et al. 2006].

The Rexer Analytics survey indicates that about 28% of the generally constructed data mining models use more than 45 variables, motivating the need for high-performance tools and algorithms [Rexer Analytics]. According to Wiley Interdisciplinary Review], there are many available tools but choosing one specific tool becomes increasingly difficult for the end-user. Data mining surveys [Rexer Analytics; Wiley Interdisciplinary Review] reveal that the most widely used platforms for real projects are RapidMiner [Mierswa et al. 2006], the R project [R Project], and the Pentaho/WEKA platform [Hall et al. 2009]. These platforms offer several sequential decision tree implementations, such as the popular (according to [Anyanwu et al. 2009]) C4.5 [Quinlan 1993] and CART [Breiman et al. 1983] algorithms. There are also many parallel implementations of the DTC algorithms [Bekkerman et al. 2011]. In this work we are

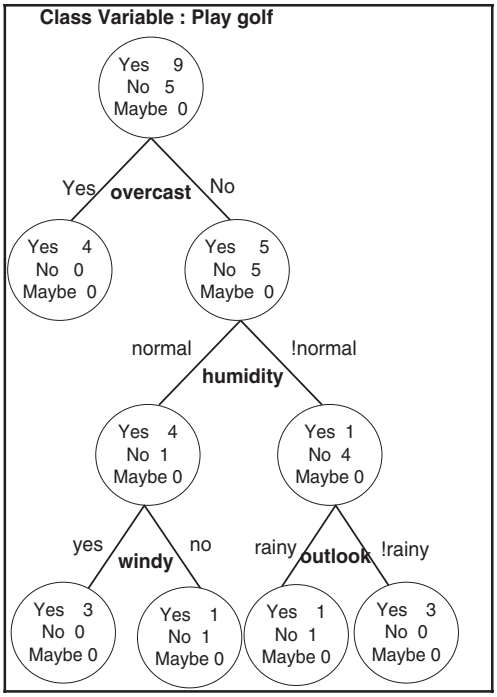

Fig. 1. An example of a decision tree structure.

using the Java platform of WEKA and the C library, *rpart*, of the R project in order to evaluate our novel approach since these are very widely used by both academic and for-profit data miners.

The performance of data mining is of great importance for many applications due to the massive input datasets. The low performance of sequential decision tree classification for real-life problems creates the need for the acceleration of the classification algorithms and the implementation of parallel decision tree algorithms, as Ben-Haim et al. described in Ben-Haim et al. [2010]. There are several ways to parallelize decision trees, described in detail by Amado et al. [2001] and Srivastava et al. [1998]. The highly parallel nature of the algorithm indicates that the decision tree training would be a suitable driving problem for multi-FPGA hardware platforms with high data transmission rate, such as the Convey HC-1 server [Convey Computer Corporation].

To sum up, the decision tree induction is considered to be a very important problem for data mining as well as for other scientific areas due to the need of turning the vast amount of available data into a meaningful format. The decision tree classification, specifically, is a highly parallelizable problem leading huge commercial companies, like Google, IBM, and Yahoo, to the proposal of efficient and scalable software-based parallel frameworks. CART is considered one of the most widely used DTC algorithms [Anyanwu et al. 2009]. In addition, there is high interest in developing more advanced versions of the CART algorithm with high classification and prediction accuracies; for example, Salford Systems implements newer versions of CART, called CART®.

In this article we present the first system implementing the complete CART DTC algorithm on a modern multi-FPGA platform capable of handling real-world databases. Our novel architecture exploits parallelism at the level of a decision variable. In particular, in order to parallelize the algorithm for each node of the tree,

a certain splitting variable and threshold must be decided; in this work we evaluate the system performance when up to 64 variables are processed in parallel, and use a pipelined comparator tree to decide the best variable to be used. If more variables are required our system can easily be scaled in order to handle them.

In the implementation of our CART system (called HC-CART) we take advantage of: (i) the parallelism in the evaluation of different variables, (ii) the high memory bandwidth available in the Convey HC-1 multi-FPGA server, and (iii) the high processing throughput offered by the FPGA technology. Furthermore, in this work we integrate our multi-FPGA HC-CART acceleration system with the rpart C-library of the R project in order to create the first known complete reconfigurable system that can handle large databases and which triggers a significant speedup when compared with the traditional software-based approaches.

The main contributions of our work are as follows.

—The novel HC-CART architecture, which is the first (to the best of our knowledge) generic parallel reconfigurable architecture for the well-known CART algorithm, utilizes a multi-FPGA platform.
—We offer the first combined HW/SW system that integrates a parallel FPGA-based architecture with a well-known and very widely used data mining software platform (the rpart library of the R project), offering high speedups in execution time compared to traditional software implementations.
—We give a performance comparison of our implemented system against previous hardware DTC algorithm implementations.
—We provide a comparison between highly parallel software approaches and our multi-FPGA implementation in terms of performance and energy consumption based on measurements conducted on real-world data mining workloads.

The rest of this article is organized as follows: Section 2 presents previous software- and hardware-based research works on the general data mining topic. Section 3 describes two of the best-known data mining platforms, WEKA and R, and describes in detail the CART algorithm. Section 4 offers a brief description of the multi-FPGA HC-1 server on top of which we implemented our novel system, and describes our proposed architecture for the efficient execution of the CART algorithm; the same section presents HC-CART, i.e., the integrated system which we implemented. Section 5 evaluates the performance of our system when handling real-world data mining workloads and it compares the HC-CART system's performance against the performance of previous proposed parallel implementations. Section 6 describes some future improvements to the HC-CART system that could offer higher performance and Section 7 draws the conclusions of this work.

2. RELATED WORK

The most widely known algorithms for tackling the decision tree induction problem are the C4.5 [Quinlan 1993] and the CART [Breiman et al. 1983] ones. The first attempts towards parallelizing the decision tree induction led to the proposal of two efficient software-based solutions: (a) the SPRINT [Shafer et al. 1996] and (b) the SLIQ [Mehta et al. 1996] algorithms; both of which handle massive datasets by either changing the CART algorithm's nature (SPRINT) or trying to change the way that the data are stored in the memory (SLIQ). The SPRINT algorithm implemented the same split selection method as the one utilized in the CART algorithm, and it is considered to be the successor of the SLIQ algorithm. The SPRINT and the SLIQ algorithms achieved an almost linear speedup with respect to the number of CPUs and the sample size. RainForest [Gehrke et al. 1998] is a framework for building fast decision trees based on massive datasets. It used a different way of processing the input datasets, so that

they could be stored in the cache memory, and in this way it improved the performance by over a factor of three when compared to SPRINT, the fastest algorithm for tree induction at that time.

As the technology moved on to multiprocessor, cluster, and grid systems, new parallel and scalable software algorithms were proposed for the data mining tree induction. The ScalParC [Joshi et al. 1998] (Scalable Parallel Classifier) algorithm is a parallel formulation of a decision-tree-based classifier that uses the basic data structure of the SPRINT algorithm. This algorithm was implemented on a 128-CPU Cray T3D platform offering a speedup of up to 2x when compared to the SPRINT algorithm. The speedup is very small due to the high communication overheads of the underlying basic algorithm.

Additional modern software-based solutions for parallel decision tree induction are recently emerging by utilizing the MapReduce framework and its open-source implementation, Hadoop. Google Inc., which works on the classification and the regression tree learning on massive datasets, presented the PLANET project, a scalable distributed framework to create tree models over large datasets [Panda et al. 2009]. PLANET uses a MapReduce cluster in order to train the classification and regression trees or ensembles of such learners; its main advantage is that it has comparable accuracy to a traditional in-memory algorithm, while it is capable of handling significantly more training data. They used up to 400 machines and achieved a linear reduction in the training execution time, up to a point where the communication overhead was the bottleneck and no higher performance could be achieved. OpenPlanet [Yin et al. 2012] is a scalable regression tree learning implementation built on top of the Hadoop framework and it is based on Google's PLANET project. They compared their system's performance with the WEKA machine learning library and a proprietary MATLAB implementation of DTC algorithm, and showed that the WEKA library achieves the best performance, but on the other hand OpenPlanet is much more memory efficient. Quin He et al. in He et al. [2010] presented parallel implementations of various classification algorithms based on Hadoop's MapReduce process. Their implementations offer linear speedup as the size of the input dataset increases. Tyree et al. present a parallel implementation of the gradient boosted regression tree algorithm using again the MapReduce framework [Tyree et al. 2011]. Their results show that their system's performance speedup can be up to almost 40x when their algorithm is executed on a multicore with shared memory and up to 25x when executed on a cluster; their speedup is reported against the time needed when the algorithm is executed on a single-threaded CPU.

A number of researchers have proposed the use of specialized hardware platforms, such as FPGAs and GPUs, on top of which they have implemented certain data mining algorithms.

The association rule learning was the first data mining area that has been implemented on special-purpose hardware. Baker et al. proposed a highly parallel custom architecture implemented on a reconfigurable computing system for the Apriori algorithm, which is a fundamental correlation-based data mining method [Baker et al. 2005, 2006]. They used bitmapped Content Addressable Memories (CAMs) offering a minimum of 24x execution speedup versus the serial SW implementation. Sun et al. presented a reconfigurable systolic tree architecture for the frequent pattern mining problem implementing the FP-growth algorithm [Sun et al. 2008; Sun and Zambreno 2008]. Their architecture offers a maximum of three orders of magnitude speedup versus a Java software implementation. Fang et al. presented two efficient Apriori implementations of the frequent itemset mining scheme on GPUs which outperform the CPU-based Apriori algorithm from one and up to two orders of magnitude depending on the input data's nature [Fang et al. 2009].

Clustering is another important data mining area, where many hardware implementations have been proposed. Saegusa et al. describe a hardware implementation of the k-means algorithm and show that real-time (30 fps) k-means clustering can be realized for large-size color images and large number of clusters, by generating the kd-trees dynamically on the FPGA [Saegusa et al. 2007]. Wang et al. presented the first FPGA-based architecture that implements the whole k-means computation algorithm in an FPGA, offering an 11x performance speedup versus the software approach; this is mainly achieved by exploiting the numerous fast floating-point units residing on today's FPGAs [Wang et al. 2007]. Estlick et al. examined certain algorithmic-level transforms of the k-means algorithm that dramatically increased the achievable parallelism [Estlick et al. 2001]. Their work achieved 200x performance speedup for image clustering compared to a general-purpose CPU executing the conventional algorithm. Finally, Ma et al. proposed a processing structure especially for GPUs that can be efficiently utilized in a wide range of data mining algorithms, like k-means clustering or EM clustering [Ma et al. 2009].

The classification problem is another data mining research area where certain hardware implementations have been proposed. Luo et al. presented an FPGA-based architecture implementing a fast Decision Tree Searching for IP Traffic Classification that is based on a novel decision classification tree with lower number of memory accesses when compared with the other known algorithms [Luo et al. 2008]. Papadonikolakis et al. proposed an FPGA-based architecture for Support Vector Machine (SVM) classification [Papadonikolakis et al. 2008]. Also, they implemented the SVM algorithm on a GPU and compared the performance between the SW implementation, the FPGA, and the GPU systems [Papadonikolakis et al. 2009]. According to their results, their FPGA-based implementation achieved a speedup of 2–3 orders of magnitude, compared to the corresponding CPU implementation, while outperforming other proposed FPGA and GPU approaches by more than 7 times. Narayanan et al. proposed an FPGA-based architecture that implements the most compute-intensive kernel of most DTC algorithms [Narayanan et al. 2007]. Their implementation offers a 5.5x performance speedup versus the software implementation, but without taking into account the I/O overhead. Chrysos et al. presented an FPGA-based system that implements the BF-tree classification method of the WEKA platform [Chrysos et al. 2011]. Their implementation offers 25x performance speedup on average versus a high-end CPU executing the same algorithm, however, the I/O overhead is taken into account.

Finally, there are certain papers that combine software frameworks, like Mapreduce, and hardware platforms, like FPGAs and GPUs, for the acceleration of data mining algorithms. Yi Shan et al. presented a MapReduce framework tailored to an FPGA, which provides programming abstraction, hardware architecture, and basic building blocks to developers [Shan et al. 2010]. They implemented on this framework a case study implementation of the Rankboost algorithm, which is an effective ranking algorithm and it is widely used in applications which involve data mining tasks. Their implementation achieved approximately an estimated 30x performance speedup when compared to the performance of a CPU-based implementation.

Concluding, there are several hardware- and software-parallel implementations of numerous data mining algorithms. Many researchers have very effectively implemented certain parts of the data mining algorithms in reconfigurable technology, indicating that the FPGA technology offers an excellent area for scaling up data mining workloads. When comparing our novel system to the existing devices implementing certain parts of the DTC algorithms [Narayanan et al. 2007; Chrysos et al. 2011], the contributions of this article are mainly the following: (1) We exploit on hardware, for the first time, the parallelism between different variable evaluations; this approach heavily increases the overall level of parallelism and thus we efficiently exploit the

underlying reconfigurable resources. (2) We integrate the FPGA-based implementation with a widely used data mining software platform; in contrast to the previous works that present standalone systems implementing a certain part of the application, we have a fully working system handling any amount of data. (3) We propose a generic and scalable FPGA-based architecture that can be used by any DT classification scheme. (4) Our performance results are based on experiments executed on top of real-world databases while the already presented performance numbers are either estimated (based on the assumption that the intercommunication overhead between the CPU and the reconfigurable devices is negligible) or based on very small datasets. (5) This is the first system, to the best of our knowledge, that implements the complete DTC algorithm on a platform consisting of a multicore CPU and several FPGAs.

3. DECISION TREE CLASSIFICATION

This section describes the data mining classification method. Also, it presents two well-known platforms, the Pentaho/WEKA platform and the R project, which were used as the actual references in this work. Finally, there is a description of the DTC method and a very detailed description of the CART algorithm that is the most widely used DTC method.

3.1. Decision Tree Classification

The classification in data mining is the process of examining a given collection of records, called a *training set*. Each record—called an *instance*—has a set of variables that are either numeric or nominal. One of these variables is the class variable, whose value is considered the output result according to the values of the other variables. The classification goal is the construction of the best model that can predict correctly the value of the class variable of an unseen instance.

Decision Tree Classification (DTC) is a simple and widely used classification method. DTC methods consist of two tasks. First, the classification learning algorithms take as input a set of classified examples and use them to build the classification tree models. Second, the decision tree model is then applied to the query records to predict their class affiliation. The leaves of the derived tree structures represent classifications, whereas the branches represent conjunctions of variables that lead to these classifications. The tree's branches are created according to homogeneity splitting criteria by choosing each time the best splitting variable. The splitting parameter differs depending on the algorithm's characteristics. Decision-tree-based classifiers are attractive because they provide high accuracy, they are simple to understand and interpret, and they can be combined with other decision techniques, giving even more accurate results.

3.2. DTC Algorithms

Standard decision tree classification algorithms that are commonly used for data mining are the C4.5 and Classification And Regression Trees (CART) algorithms. These algorithms expand the tree nodes in a fixed order based on various criteria (e.g., information gain or Gini index for C4.5 and CART, respectively). The Gini gain score is an impurity measure which shows how often a randomly chosen instance from the set would be incorrectly labeled. It is computed by summing the probability of each item being chosen multiplied with the probability of incorrect categorizing. The CART algorithm usually constructs binary tree models. The splitting of the training set at each node is based on the variable's choice that maximizes the reduction of the impurity.

3.3. CART Algorithm

The pseudocode of the CART algorithm is presented in Figure 2. First, the algorithm reads the training data that will be used for the construction of the classification

Function : CART tree costruction
Input : training instances, number ofinput attributes
Output : Tree model
✓ Read the training data.
✓ Consider ***all*** possible values of ***all*** variables.
✓ Calculate for **each** possible value of ***all*** the variables the Gini score.
✓ Select the variable/value **($X=t_1$)** that produces the maximum Gini score. ($X=t_1$ is called a "split")
✓ If$X< t$ then send the data to the "left node"; otherwise, send data point to the "right node".
✓ Now recursively repeat same process on these two "nodes"
✓ Return the tree model.

Fig. 2. Pseudocode for the CART algorithm.

```
@relation weather.symbolic

@attribute outlook {sunny, overcast, rainy}
@attribute temperature {hot, mild, cool}
@attribute humidity {high, normal}
@attribute windy {TRUE, FALSE}
@attribute play {yes, no, maybe}

@data
sunny,hot,high,FALSE,no
sunny,hot,high,TRUE,no
overcast,hot,high,FALSE,yes
rainy,mild,high,FALSE,yes
rainy,cool,normal,FALSE,yes
rainy,cool,normal,TRUE,no
```

Fig. 3. Example of input dataset.

model. Figure 3 shows an input dataset example, with 4 independent variables: *outlook, temperature, humidity, windy*, and one class variable: *play*. Then, the algorithm counts the item frequencies of all the possible value combinations between the independent variables and the class variables. The item frequencies of the class variable's values are also counted. These frequencies are used to find the best splitting criterion for each one of the non-leaf nodes. The splitting criterion for the CART algorithms is based on the impurity score (called Gini gain score) of the nodes, which are created after the splitting of the parent node into its two successors. Next, the algorithm selects the value of the variable with the best score and splits the dataset into two separate subdatasets each for the two children nodes. Last, the process is repeated recursively until the data of the children nodes reach down to a specified threshold. According to Narayanan et al., the critical part in the construction of decision tree nodes is the choice of the splitting attribute and its corresponding value, consuming approximately 95% of the overall execution time [Narayanan et al. 2007].

3.4. Parallel Formulations of DTC

There are several possible approaches to create decision trees in parallel. Srivastava et al. [1998] and Amado et al. [2001] describe three different software-based parallel

formulations of DTC. The first one, *synchronous tree construction*, proposes that each worker works on the same expanded node but on a different part of the dataset. The partitioning of the dataset can either be horizontal or vertical. For the horizontal partitioning each worker processes only a part of the input dataset; however, this method needs great communication overhead to combine the results. For the vertical partitioning each worker processes one or more independent variables by finding the best splitting gain. Finally, the best splitting criterion is found by comparing the calculated values from the all the workers. The advantage of the synchronous tree construction formulation is that no movement of the training dataset is required; however, this formulation suffers high communication cost in cases of systems without shared memory.

The second parallel formulation, *partitioned tree construction*, proposes that the worker works on different nodes of the tree, therefore on different datasets. The advantage of this approach is that each worker is responsible only for a specific subtree. On the other hand, this approach is demanding in synchronization and communication for the expansion of the upper part of the classification tree.

The third parallel formulation of DTC algorithm is the hybrid of the previous two approaches. It starts using synchronous tree construction for as long as the communication cost is not too high and switches to partitioned tree construction in the lower tree levels. In this way it combines the benefits from both techniques and it is shown to provide the best performance on a platform where the processors are connected in a hypercube topology.

3.5. Data Mining Platforms: Frameworks

3.5.1. Pentaho/WEKA Platform. The Waikato Environment for Knowledge Analysis (WEKA) [Hall et al. 2009] is a Java open-source machine learning suite that implements many state-of-the-art data mining algorithms. The WEKA tool offers the opportunity of parametrical runs of classification, regression, clustering, and association rule algorithms, while being able to visualize and further analyze the mining results. The WEKA tool has limited support for parallel execution of scripts and methods through WEKA-Parallel [Celis et al. 2002]. However, they do not offer a parallel version of the CART algorithm, which is itself trained atomically on a single machine and whose training time is limited by the internal memory. According to Annual Rexer Analytics Data Miner Surveys for 2009 [Rexer Analytics] and the survey in Mikut et al. [2011], the WEKA tool is one of the most popular tools for data miners.

3.5.2. R Project. The R project [R Project] is an open-source software environment for statistical computing, graphics, and data analysis. It provides a wide variety of statistical and graphical techniques, classification, clustering, and other data analysis techniques. The R software environment is written primarily in C, FORTRAN and R programming languages, which can be linked and called at runtime for computationally intensive tasks. The R tool is easily extensible through user-submitted packages that implement specific functions and extensions. Therneau and Atkinson in Therneau and Atkinson [1997] describe Rpart, an R package that implements many of the ideas found in the CART algorithm. The Rpart library builds classification or regression models of a very general structure using a classic two-stage procedure. According to Rexer's Annual Data Miner Survey in 2010 [Rexer Analytics], R is the data mining tool that is used by most data miners (about 43%).

4. SYSTEM ARCHITECTURE

In this section, we first describe the Convey HC-1 platform on top of which we implemented our novel proposed architecture. We then describe in detail the proposed FPGA-based architecture for the CART algorithm, and its mapping to the multi-FPGA

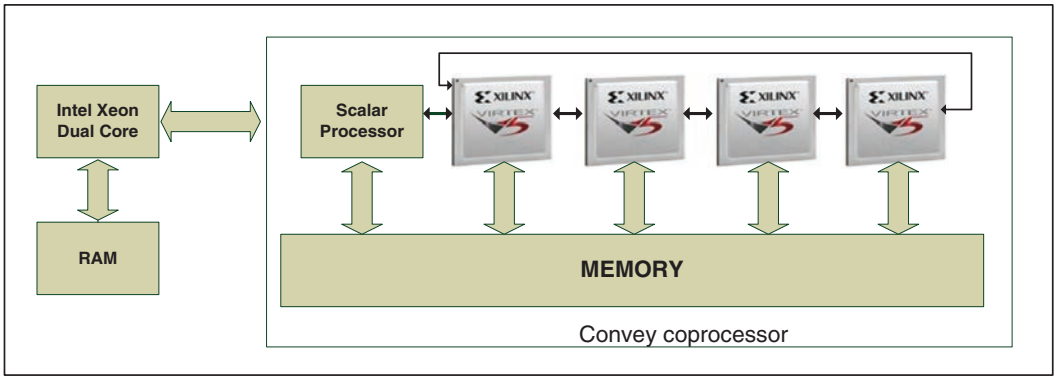

Fig. 4. Convey HC-1 system.

Convey server. Finally, we describe the integration of the Convey HC-CART system with the R project's Rpart software library so as to create a fully functional complete system that can handle real-world databases.

4.1. Convey HC-1Server

HC-1 server is a hybrid-core computer offering an important platform to the high-performance computing (HPC) field. Convey's infrastructure (shown in Figure 4), combines a 4-core Intel Xeon processor and a Convey-designed coprocessor based on Xilinx Field Programmable Gate Arrays, with its own high-bandwidth memory addressed and cache-coherent memory subsystem. There are two main advantages in HC-1 platform: first, it offers the ability to build new FPGA-based architectures, which are implemented on the Convey reconfigurable coprocessor; second, it offers a bandwidth of up to 80Gigabytes/sec, allowing great potential in cases of data-intensive problems.

The Convey HC-1 coprocessor consists of three main components: the Application Engine (AE), the Memory Controllers (MCs), and the Application Engine Hub (AEH). There are four user-programmable Application Engines, each of which corresponds to a Virtex-5 XC5VLX330 FPGA, and that are used to map the user's FPGA-based design. The Application Engine Hub (AEH) consists of a scalar processor, which is responsible for the interface between the host Intel Xeon processor and the AEs. Finally, there are 8 memory channels (MCs) that are used to transfer data from the memory to the application engines and vice versa. Each MC is connected to two DDR2 DIMMs. The coprocessor supports multiple instruction sets, called *personalities*. The personalities compose the mappings of the FPGAs and they are preloaded to them. The high transmission rate is a key factor for implementations of data mining algorithms in general and in particular for our proposed architecture of the CART algorithm.

4.2. FPGA-Based HC-CART Architecture

4.2.1. FPGA-Based Parallel Architecture. This section describes how we parallelize the CART algorithm so as to efficiently map it to the HC-1 server. As described in Section 3.4, there are three main parallel formulations of the DTC algorithms: the *synchronous tree construction*, the *partitioned tree construction*, and the *hybrid method*, which is the combination of the other two methods. In this work we claim that the *synchronous tree construction* is the "best" method for mapping DTC algorithms on a multi-FPGA platform with shared memory, such as the Convey HC-1 server. According to Srivastave et al. [1998] and Amado et al. [2001], the main disadvantage of this method, especially

in cases of vertical partitioning, is the high communication cost. This cost is eliminated by the Convey HC-1 server's architecture, which offers high data transmission between the shared memory and the coprocessor. Also, this approach takes advantage of the fine-grained parallelization that the FPGA technology offers. On the other hand, the *partitioned tree construction* needs coarse-grained parallelization which is severely restricted by the limited FPGA resources. The partitioned tree construction uses multiple "computing machines", each implementing the entire process of (sub)tree construction; such a machine cannot fit in the available FPGA resources. The *hybrid tree construction* method, which combines the previous two methods, is difficult to map on the FPGA technology due to its "dynamic" processing depending on the part of the tree that is processed. Also, the R project, which was utilized in our combined HW/SW system, uses the depth-first method for the tree construction, thus, the software implementation eliminates the level of the parallel processing. Finally, in our future plans, as described in Section 6, we plan to implement a *semihybrid formulation* where we could exploit the parallelism of the implemented CART modules on nodes that are expanded only on the same level of the tree and only in the cases in which the number of the input variables is not greater than the implemented "workers".

In our proposed architecture we use the *synchronous tree construction*, where multiple parallel computing engines work on a specific portion of the input dataset. In the proposed implementation, each engine calculates the best splitting score for a certain independent variable. Thus, the overall proposed architecture, which is presented in the next section, consists of parallel computing components, each one of which examines individually and independently the input variables by exploiting the available parallelism at the variable level.

$$Gini\ Gain = \frac{\sum_{i=0}^{i<num\_value} R_i^2}{R_{total}} + \frac{\sum_{i=0}^{i<num\_value} L_i^2}{L_{total}} \quad (1)$$

4.2.2. CART Module's Architecture. Next, we describe the basic module's architecture of the proposed implementation, that we call the CART module. The CART module computes the highest splitting score for one independent variable of the input dataset for all the possible splitting ways, as shown in Figure 5. The basic splitting criterion for the CART algorithm, as mentioned earlier, is the impurity score of the node's data that is named Gini Gain score. The CART module takes the input dataset and finds the value of the processing variable that gives the maximum Gini Gain score, according to Eq. (1), as described by Breiman and Olshen [1983] in the official formulation of the CART algorithm. The coefficients R_i, L_i, R_{total}, and L_{total} are the number of instances that belong to the right and left child nodes, respectively, which are produced by the dataset splitting process. An abstract presentation of the CART module is shown in Figure 6.

4.2.3. Frequency Counting Module Architecture. The Frequency Counting module is responsible for producing the coefficients needed for the calculation of the Gini Gain score, as described in Eq. (1). The Frequency Counting module consists of three basic memories: Parent Memory, Left node memory, and Right node memory, and some additional logic, as shown in Figure 7.

As a first step, the CART module processes the input dataset. The Parent Memory keeps the number of all possible value combinations between the examining variable and the class variable. The values of all variables are enumerated, so the combination of the independent variable's value and the class variable's value makes a unique address of the Parent memory, where the number of occurrences of this combination is stored. We assume that there are up to 64 distinct values for each input variable and up to

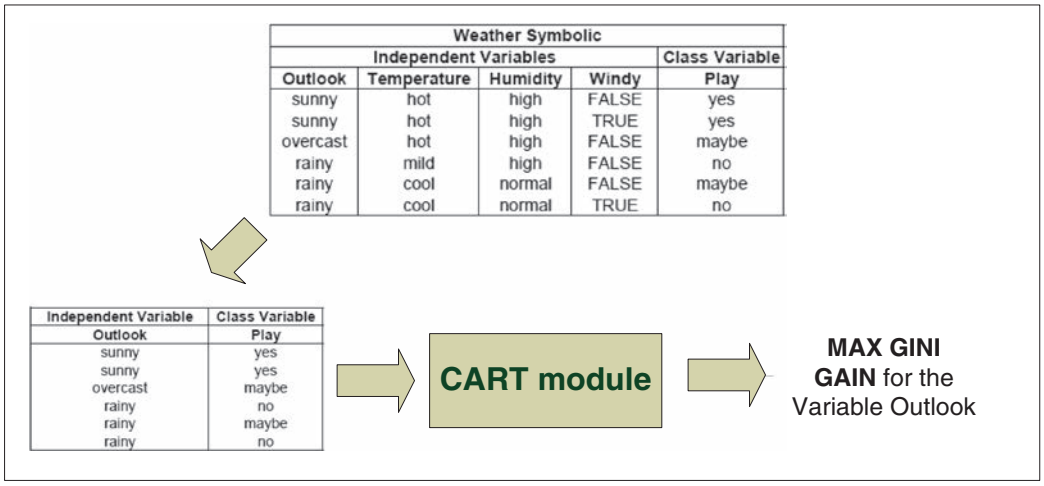

Fig. 5. The node's dataset is split into pairs of independent variables and the class variable, which are the input to the basic CART modules.

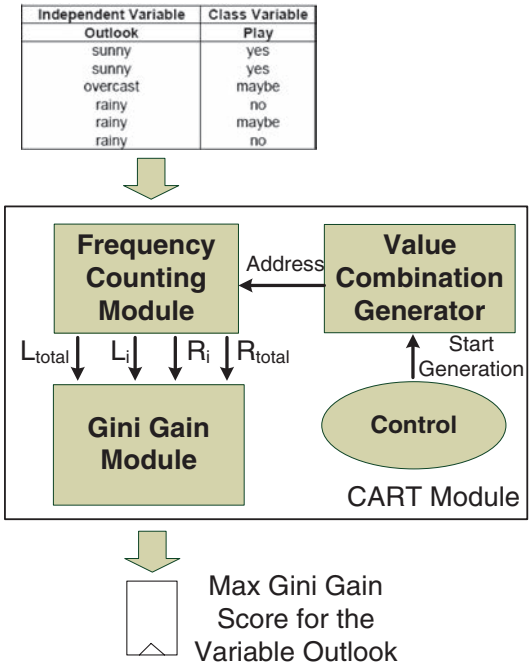

Fig. 6. The CART module's architecture.

64 distinct values for the class variable. Thus, the Parent memory has 4096 entries and we implement it using an internal FPGA BRAM module (BRAM, or Block RAM is a static memory embedded in the FPGA integrated circuit; an FPGA may have many such memories that can accessed independently or together).

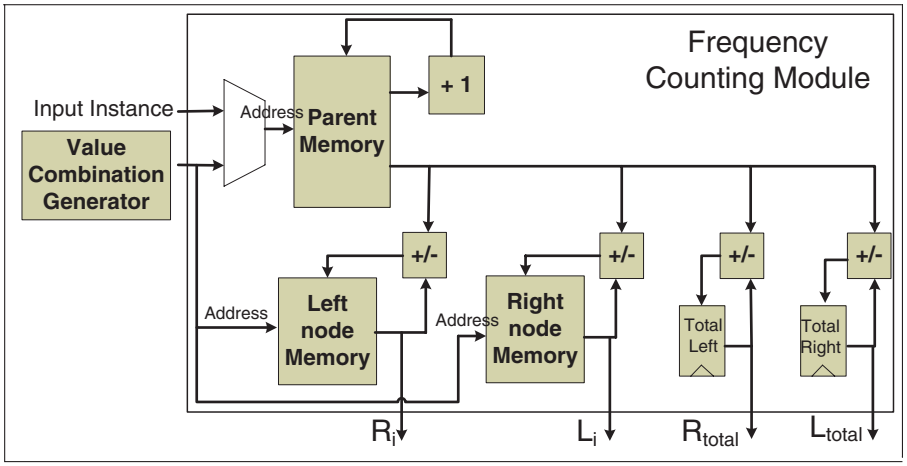

Fig. 7. The Frequency Counting module's architecture.

Then the splitting process begins. The control triggers the Value Combination Generator (refer to Figure 6) to produce the address used to read the memory modules. The Right and Left node memories are used for keeping the number of instances that will be split into the right and left tree node. The splitting process takes place according to the each instance's examining variable's value, which results, according to our previous assumption, in the fact that these two memories have only 64 entries. First, the Parent Memory is read and all the instances are stored in the Right Node memory, as there is no splitting criterion. Then, the Value Combination Generator in cooperation with the system's control unit produces all possible splitting criteria and therefore all the respective memory addresses. This process uses adders to produce in parallel all the needed coefficients for the Gini Gain score calculation.

The Frequency Counting module uses 32-bit wide fixed-point arithmetic, as our sensitivity analysis on various size and nature input datasets showed that we had absolutely no loss of accuracy in the final results. Also, the fixed point arithmetic was used to save FPGAs resources and thus to increase the parallelism exploited by our system.

4.2.4. Gini Gain Module Architecture. As mentioned above, the Gini Gain score for the CART algorithm is calculated according to Eq. (1). The R_i and the L_i values describe the numbers of instances that belong to the right and the left node child, respectively, when a splitting criterion is used. These values are stored in the Left and Right node memories that were presented earlier. The R_{total} and the L_{total} values are the total number of the instances that belong to the left and right node, respectively. The Gini Gain module's architecture is fully pipelined and it calculates in parallel the Gini Gain score of the two node children, as shown in Figure 8.

The implemented module used both single floating-point arithmetic in order to produce highly accurate final results, and fixed point arithmetic for the processing before the division in order to save up more FPGA resources. The results of the final system showed that the proposed architecture supports the application requested accuracy for all the examined input datasets. The Gini Gain's final value is the maximum Gini Gain score achieved by all the possible combinations of splitting the input dataset's independent variable.

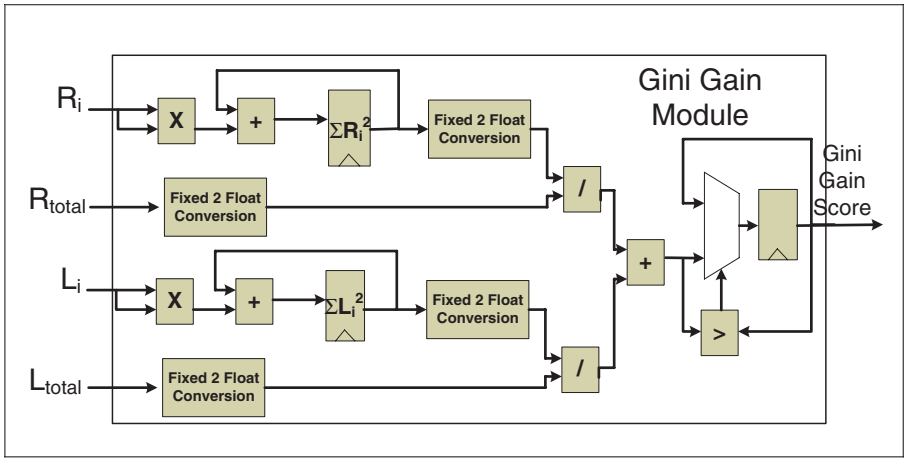

Fig. 8. The Gini Gain module's architecture.

Fig. 9. CART module mapped on a single AE (four AEs are instantiated in the Convey HC-1 server).

4.3. HC-CART Architecture

This section describes the full system implementation of the FPGA-based CART architecture on the Convey HC-1 server. As described above, the final HC-CART system exploits the variable parallelization of the input dataset. Each one of the HC-1's Application Engines (AEs) maps 16 parallel fully pipelined CART modules, which take advantage of the 16 independent busses between the MCs and the shared memory. A memory crossbar was used as each CART module can access any memory location. The critical resource of the FPGA utilization is the number of slices, as the HC-CART system uses 61% of the available Xilinx Virtex-5 LX330 FPGA's logic. Figure 9 shows the architecture implemented in each one AE of the HC-1 server.

The Max Gini Gain module, shown in Figure 10, uses a fully pipelined tree structure of comparators. This module takes as input the 16 best Gini Gain scores for the 16 input variables, which are processed by a single AE, and outputs their maximum Gini

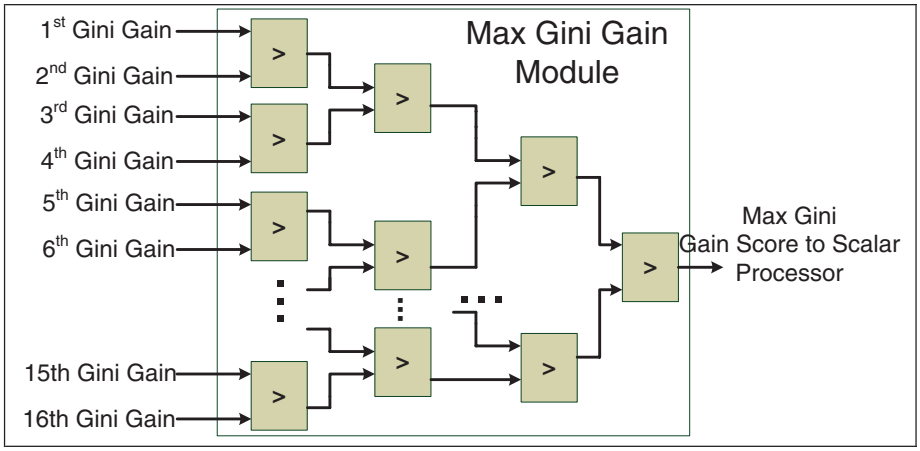

Fig. 10. Max Gini Gain module's architecture.

Gain score. This structure was proposed previously by Naranayan in Naranayan et al. [2007] to solve the same problem, giving impressive results. The maximum Gini score for each one of the four HC-1 application engines is sent back to the Scalar processor, which finally outputs the maximum Gini score of all 64 processing variables to the host processor.

4.4. HC-CART System Integration

Next, we describe the final system integration of the proposed HC-CART system, which was mapped on the Convey HC1 server, with the software R platform. The R's rpart library is the most accurate software implementation of the official CART algorithm. The proposed system finds the maximum Gini gain score, the splitting variable, and its value for a single node splitting. In the integrated system, the R platform runs on the host Intel Xeon processor of the HC1 server. When the Rpart library is loaded and executed, a function call in the Rpart library triggers the execution of the hardware-based HC-CART custom instruction. This instruction is executed for each node splitting of the tree and returns to the host processor the best Gini gain and the variable needed for the splitting process. The software takes all these values back and it continues its execution on the host processor. The entire process is repeated for each node splitting until the decision tree construction finishes. The flowchart for the final integrated HC-CART system is shown in Figure 11.

5. PERFORMANCE EVALUATION

This section evaluates the performance of the implemented HC-CART system for various input datasets. Also, we compare the performance of the HC-CART system against previous FPGA-based works on decision tree classification qualitatively in terms of architecture and quantitatively in terms of performance. Finally, we compare the performance of our proposed architecture to the performance of parallel software-based DTC solutions on high-end modern multi-CPU platforms.

5.1. Datasets

The input files of the system are referred as Attribute-Relation File Format (ARFF). An ARFF file is an ASCII text file that describes a list of instances sharing a set of variables. The ARFF is used by many data mining platforms, and a typical example of ARFF was presented in Figure 3.

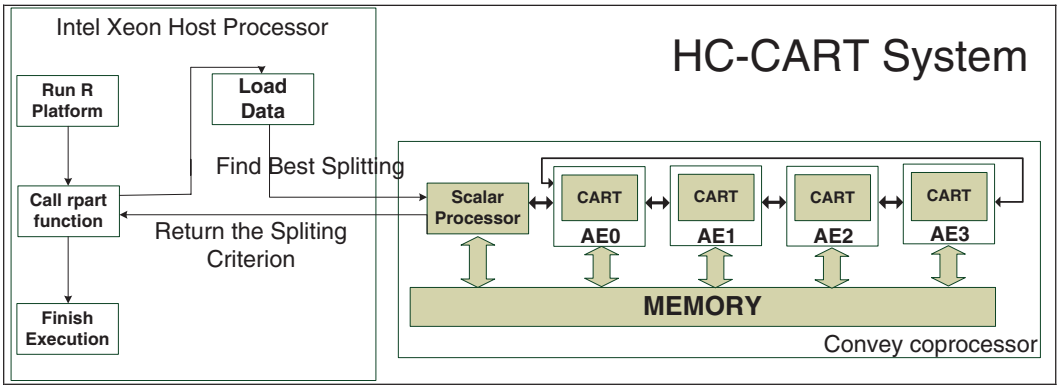

Fig. 11. Integrated HC-CART system flowchart.

Several tests took place on the implemented system so as to validate its correct and accurate operation. The datasets contained both nominal and numeric variables. The nominal variables are much more process demanding than the numeric ones due to the algorithm's nature. According to the CART algorithm, the numeric variables are initially sorted and, next, the median value for each variable is calculated as the splitting criterion. On the other hand, the splitting criterion for each nominal input variable is calculated by the "examination" of all the possible combinations of its possible values. Thus, the search space for each nominal variable increases exponentially as the possible values of the input variable increases. In other words the algorithm accesses only two times the complete database, in the case of numerical variables, whereas it should read numerous times the complete database in the case of nominal variables since those variables cannot be sorted and all the possible value combinations need to be computed.

The datasets were either randomly generated datasets from the WEKA software or datasets that have been used in related papers. We used datasets that ranged from 8 to 128 variables with 4 to 20 possible values per variable. Also, the size of the input datasets ranged from some KBs up to a few GBs. The output results from the system were compared and were found to be identical to those of the official WEKA Java software, version 3.6.2 and the R's Rpart library.

5.2. Implementation and Performance Results

The HC-CART system has no restrictions as far as the nature of the input dataset's variables (nominal or numerical), the number of input variables, and the size of the dataset are concerned. The numerical variables, which are much less process demanding than the nominal ones as described above, are handled by the software in parallel with the coprocessor call that handles the nominal variables of the input dataset.

The HC-CART system supports up to 64 possible distinct values per each nominal variable, and it can cope with multiclass data that are widely used in data mining. Our experience with the real-life data mining datasets from widely used data mining repositories, like UCI, shows that usually the value range for the nominal variables can reach up to a couple of decades; such variables can easily be handled by the HC-CART system. Also, the software tools, like WEKA, use a different algorithm when processing data with a large number of possible values; those algorithms provide significantly less accurate results than CART. Finally, the HC-CART architecture can cope with variables of greater value ranges by slightly changing only the "fixed point" part of the

Table I. Tests with Fixed Numbers of Variables and Possible Values/Variable (#variables: 32, #values per variable: 16)

#text	#instances	WEKA's SimpleCART (Heuristic) (sec)	Rpart Library (sec)	HC-CART System (sec)	Speedup HC-CART vs. SimpleCART	Speedup HC-CART vs. Rpart library
1	500	1.86	5.82	0.26	7.15x	22.38x
2	1000	2.63	8.36	0.27	9.74x	30.96x
3	10000	16.07	11.45	0.36	44.64x	31.81x
4	100000	230.71	12.83	1.79	131.55x	7.17x
5	1000000	2708.47	52.71	20.61	131.42x	2.56x

architecture; in particular such a change involves the use of wider registers, memories, and arithmetic units, like multipliers and adders.

The Convey HC-1 server includes a 4-core Intel Xeon CPU at 2.13 GHz with 32GB RAM. We used this host processor for both the experimental runs of the SimpleCART classifier of the WEKA java suite (i.e., WEKA's platform CART approach) and the R platform C code Rpart library. The WEKA's SimpleCART classifier offers two different execution modes: heuristic and exhaustive search. The exhaustive search mode is the standard official implementation of the CART algorithm but is very slow due to the huge search space in case of nominal variables. On the other hand, the heuristic mode, which was used in our experiments, uses probabilistic methods in order to reduce the search space offering less accurate models.

The software performance depends on three basic factors: the number of instances, the number of variables, and the number of possible values of each variable of the input dataset. Therefore three different types of real-world tests were conducted, in order to show the performance achieved by the HC-CART system for different input datasets. Each test differentiates one of the three above parameters and uses fixed "typical" values, according to real-life datasets, for the rest of them. Also, it is important that the implemented system was tested with real-world input datasets, like the datasets that are stored in very large repositories, e.g., UCI.

Table I shows the scaling of the HC-CART system compared to software-based solutions when the number of instances of the input dataset increases while the other two parameters retain stable. As the size of the real-life datasets continuously increases, it is important that the HC-CART system performance speedup vs. the software implementations varies from one to two orders of magnitude. As shown in Table I, the communication overhead for huge number of instances is a critical factor to the system's performance, however, the HC-CART still offers at least one order of magnitude performance speedup vs. single-threaded software solutions.

Table II shows the performance of the HC-CART system when only the number of variables of the input dataset increases. Usually, the real-life datasets consist of several dozens and up to thousands of independent variables. In Table II we can see that the performance speedup of our HC-CART system can reach up to almost two orders of magnitude vs. the official software implementations. It is important to note that the implemented system can support an unlimited number of variables, as far as the number of the input variables that will be processed in parallel are combined in groups of 64.

Table III shows the performance of the implemented system when the number of possible values per variable increases. As shown in Table III, the speedup in the execution time can reach up to 100 times for a dataset where its software execution takes almost a half hour.

Table II. Tests with a Fixed Number of Instances and Number of Possible Values/Variable (#instances: 100000, #values per variable: 16)

#test	#variables	WEKA's SimpleCART (Heuristic) (sec)	Rpart Library (sec)	HC-CART System	Speedup HC-CART vs. SimpleCART	Speedup HC-CART vs. Rpart library
1	8	40.75	1.89	0.79	51.58x	2.39x
2	16	89.14	5.08	1.19	74.91x	4.27x
3	32	235.47	12.83	1.79	131.55x	7.17x
4	64	438.11	24.19	3.53	124.11x	6.85x
5	128	1292.72	43.542	4.78	270.40x	9.10x

Table III. Tests with a Fixed Number of Instances and Number of Possible Values/Variable (# instances: 100000, #variable: 64)

#test	#of Possible Values/Variable	WEKA's SimpleCART (Heuristic) (sec)	Rpart Library (sec)	HC-CART System (sec)	Speedup HC-CART vs. SimpleCART	Speedup HC-CART vs. Rpart library
1	4	162.64	4.42	3.08	52.81x	1.44x
2	8	270.38	4.22	2.91	92.91x	1.45x
3	16	441.28	24.19	3.53	124.11x	6.85x
4	20	1002.76	250.03	4.72	212.45x	52.97x

To sum up, this section compared the performance of the single-threaded implementations that are provided by the WEKA tool and the R suite against the performance of the HC-CART system for various input datasets. The comparison showed that the HC-CART system's performance speedup vs. single-threaded software solutions varies from one order of magnitude and up to two orders of magnitude for real-time measurements. The results presented in the above tables show that our system can dramatically reduce the execution time of building the decision tree model from hours to seconds, respectively.

Moreover, these results show that the HC-CART system can significantly outperform the software solutions as far as energy consumption is concerned. In more details, the execution of the Rpart library, which is the best single-threaded software implementation reported, consumes at least 525 Watts per hour on a modern high-end server whereas on the other hand the HC-CART system consumes about 1320 Watts per hour. Taking into account that the speedup that the HC-CART system triggers against the purely software execution of the Rpart library is on average 12x our innovative device consumes about 4 times less energy than a standard server executing the Rpart library for the construction of the same decision tree model.

5.3. HC-CART System versus HW-Based DTC Implementations

This section compares our novel approach with previously reported hardware systems implementing decision tree classifiers. There are mainly two papers proposing and evaluating reconfigurable architectures for decision tree construction algorithms.

Narayanan et al. [2007] presented a reconfigurable implementation that calculated only the Gini Gain score of the input dataset involved in the construction of a decision tree model. Their system consisted of 16 parallel computing units mapped on a low-resource FPGA (Virtex II) that were controlled by the FPGA's embedded PowerPC processor. They used fixed-point arithmetic for their calculations in order to increase their system performance, but they did not take into account the FPGA's data I/O overhead. Their implemented system achieved a maximum performance speedup for

the Gini calculation of 5.58x compared to the software implementation. Also, their implemented system achieved an estimated 1.5x overall speedup compared to the ScalParC algorithm, an efficient parallel formulation of the decision tree induction. The ScalParC algorithm is very similar to the CART algorithm as it uses the Gini Gain score as well. We estimate, according to their resource utilization and the new reconfigurable technology of a modern Virtex 5 FPGA (i.e. the FPGA employed on our Convey system), that, the new FPGAs would fit about 5 times more Gini units working with a 20% higher clock; thus, their estimated speedup may grow up to approximately 8x. However, the speedup reported by the authors is estimated based on the fact that the intercommunication between the CPU and the Gini module would take virtually no time. Based on our real-world measurements this intercommunication is very time consuming; if the intercommunication delay is not taken into account our system is on average 25x faster than the software implementation or about 3x faster than their system, implemented on the same FPGA and given that the reference software systems are considered virtually the same. Moreover, our proposed HC-CART system implements the same Gini score calculation using floating-point arithmetic operations and thus it produces results of much higher accuracy. Furthermore, the overall performance speedup of our HC-CART system can exceed 100x compared to widely used software implementations executed on state-of-the-art servers.

In the authors' own previous work, Chrysos et al. proposed an architecture consisting of parallel computing machines that implemented the input item frequency counting, coupled with a Gini Gain computing module [Chrysos et al. 2011]. We implemented the BF-tree algorithm (e.g., a CART-like algorithm) offering a maximum speed of up to 20x when compared to the execution time of the Java-based algorithm on a high-end server. The implemented system consists of 64 parallel computing machines, thus, it can cope with up to 64 possible different values for each variable. The main drawback of the architecture proposed in Chrysos et al. [2011] is that the datasets with high-dimensionality variables and therefore high search space need to be scanned more than once, adding the respective overhead to the performance. On the other hand, the performance of that implementation is excellent for datasets with low dimensionality.

This work can be considered as a successor of this work presented in Chrysos et al. [2011]. The main differences between the implementation in Chrysos et al. [2011] and the HC-CART implementation are three. First, this work implements a different algorithm, the widely known in data miners CART algorithm, that is close but distinct to the BF-tree method. Second, the two architectures exploited different levels of parallelism. The CART module calculates the maximum Gini score for each processing variable, which means that each variable is processed independently and in parallel with other variables. On the other hand, the work in Chrysos et al. [2011] calculated the Gini Gain for the possible values of each variable in parallel. An advantage of our approach when compared with Chrysos et al. [2011] is that we can support unlimited number of variables in the input datasets, without a need to read the input database more than once. Also, our present work is the first to map a DTC algorithm on a multi-FPGA system, and it achieves speedups that can exceed 100x compared to the official heuristic software implementations of the CART algorithm. Finally, when comparing the maximum performance speedup of 20x that is presented in Chrysos et al. [2011] to the maximum speedup of 200x which is triggered by the HC-CART system it is clear that this work significantly outperforms previous work for at least one order of magnitude.

5.4. HC-CART System versus Parallel SW-Based DTC Implementations

This section evaluates the performance of the HC-CART system when compared with the performance of other presented parallel software-based implementations.

Panda et al. [2009] and Yin et al. [2012] described their parallel DTC implementations, which were based on Google's PLANET project. The OpenPlanet project [Yin et al. 2012], which is the most recent work, implemented the regression tree construction algorithm that is very similar to the CART algorithm, on a 64-core Hadoop cluster. They compared their system's performance against the performance that is achieved by the single-threaded WEKA's REPTree method and the single-threaded MATLAB's classregtree function when they are implemented on a high-end server. They used a core dataset that contained 17,544 tuples (instances) with nine variables each, and they replicated the dataset in order to build datasets from 1.7M to 17M tuples. Their results showed that WEKA and MATLAB tools offered better performance than OpenPlanet for small datasets up to 10M tuples. As the size of the dataset increased, and up to more than 10M tuples, their implementation offered better performance versus MATLAB, however, the performance of the WEKA tool was still better. Finally, they summarized that that their 64-CPU system would give better results than WEKA for a dataset with more than 20M tuples.

As these two implementations [Panda et al. 2009; Yin et al. 2012], as well as their input datasets, were not available to us nor published, we evaluated the HC-CART system's performance using the same reference points. In particular, we compared the performance of our novel system vs. the performance of WEKA's REPTree method as well as vs. MATLAB's classregtree function, which were both used as the reference points in Panda et al. [2009] and Yin et al. [2012], when they are implemented on a high-end server as single-threaded applications. Also, we used input datasets with the same nature and size as those described in the papers.

The Breast Cancer Wisconsin (Diagnostic) dataset from the UCI repository [Frank et al. 2010] was used for our tests, as it consists of 9 variables, and we replicated it to make datasets with 1.7M up to 21M tuples (instances). Both software platforms ran on a high-end 8-core CPU Intel Xeon at 2.66 GHZ server with 12GB RAM and with a compute server motherboard configuration. All the tests took place with the highest optimization flags. The execution time of the three systems and the speedup achieved by the HC-CART is shown in Figures 12 and 13, respectively.

According to the results that are presented in Figures 12 and 13, the HC-CART system outperforms both software single-threaded implementations for all different inputs. Also, it is important that the HC-CART system achieves at least 2x performance speedup against MATLAB's classregtree function even though it processes datasets that exploit less than 1/7 of the full available parallelization (only 9 from 64 parallel CART computing machines are utilized in those experiments). Obviously for larger datasets, which will exploit all our parallel machines, the speedup may well be more than an order of magnitude. As described in Panda et al. [2009] and Yin et al. [2012], the WEKA tools and/or the MATLAB platforms in many cases outperformed their 64-CPU parallel implementations, thus, we can come up to the conclusion that the HC-CART system totally outperforms those two published parallel implementations on the decision tree construction for all the tested datasets.

In addition to the above, we evaluated the performance of the HC-CART versus the performance of the MATLAB's classregtree function when the number of variables varies. The results, as presented in Figure 14, showed that the HC-CART system outperforms the MATLAB execution on a high-end server for any number of variables; in particular, the HC-CART speedup increases slightly when the number of variables gets higher.

Also, we evaluated the HC-CART system's performance as far as its linearity towards the increasing size of the input dataset. He et al. [2010] presented a parallel implementation of a new decision tree algorithm using the MapReduce framework. They used the car.data dataset from the UCI repository and they evaluated their system's

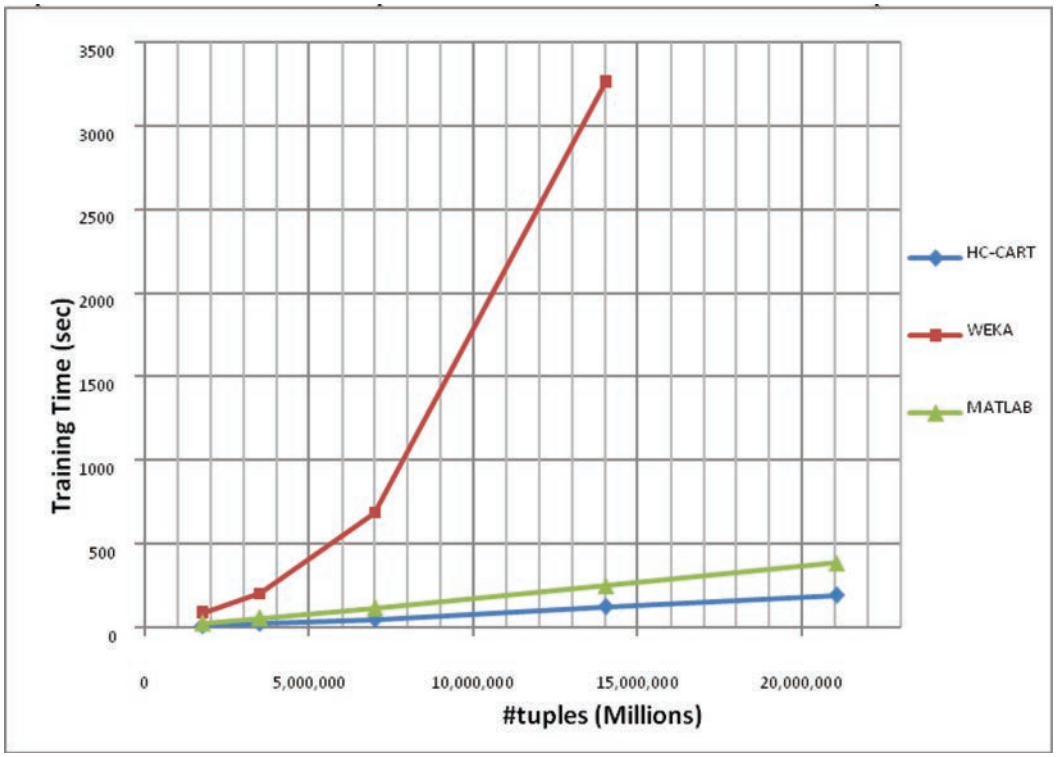

Fig. 12. Execution times for the HC-CART, WEKA platform, and MATLAB's classregtree function.

performance as they increased the size of dataset from 5MB up to 8GB. According to their results, their implementation offered linear growth of the execution time with regards to the sequential training time of the algorithm and they claim that this is the main advantage of their approach. As they did not use the CART algorithm, we cannot make a direct performance comparison of the two implementations. However, Figure 15 shows that the latency of the HC-CART system grows linearly with the size of the input dataset, as well.

Concluding, this section presented an indirect comparison of the HC-CART system's performance against the performance achieved by various parallel software-based DTC approaches. Our evaluation shows that the HC-CART system significantly outperforms other solutions that use cluster or grid systems. Also, the latency of HC-CART grows linearly with the size of the input database.

6. FUTURE WORK

The HC-CART system presented in this article implements a highly parallel system and it is considered to be "optimal" in terms of its mapping to a state-of-the-art FPGA platform in terms of datapath widths, careful usage of fixed-point and floating-point arithmetic without loss of robustness, etc. On the other hand, there are some improvements that can still be done on the task and the data scheduling schemes in order to exploit the parallel use of the computing machines on such a multi-FPGA platform. As described in Section 4, the HC-CART system exploits the available parallelism at the variable level for one node construction per coprocessor call.

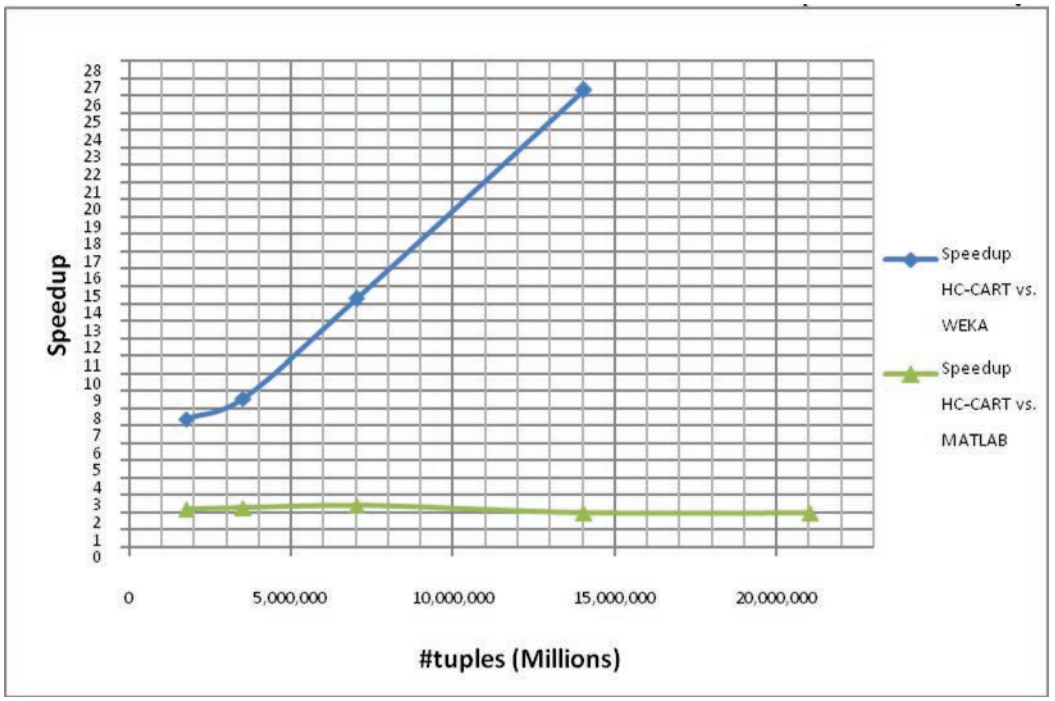

Fig. 13. Speedup of HC-CART training time versus WEKA platform and MATLAB's classregtree function training time (the WEKA platform did not construct decision tree models for datasets with more than 14M tuples due to restricted memory).

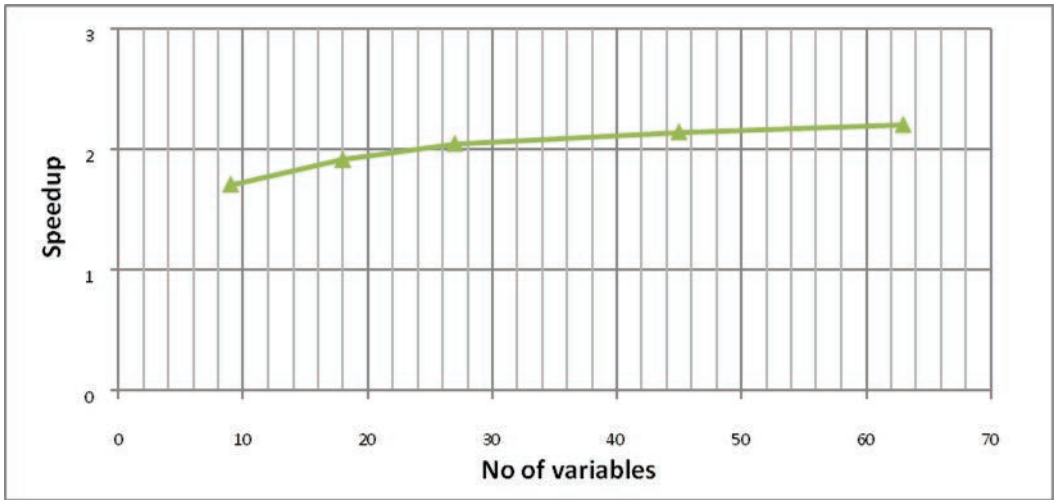

Fig. 14. Speedup of HC-CART training time versus MATLAB's classregtree function training time for different number of variables.

The HC-CART system consists of up to 64 computing machines which can examine individually and independently the input variables. The input datasets that contain a low number of attributes (much lower than 64 variables) lead to the "deactivation" of many computing machines and as a result to lower performance. In our future plans,

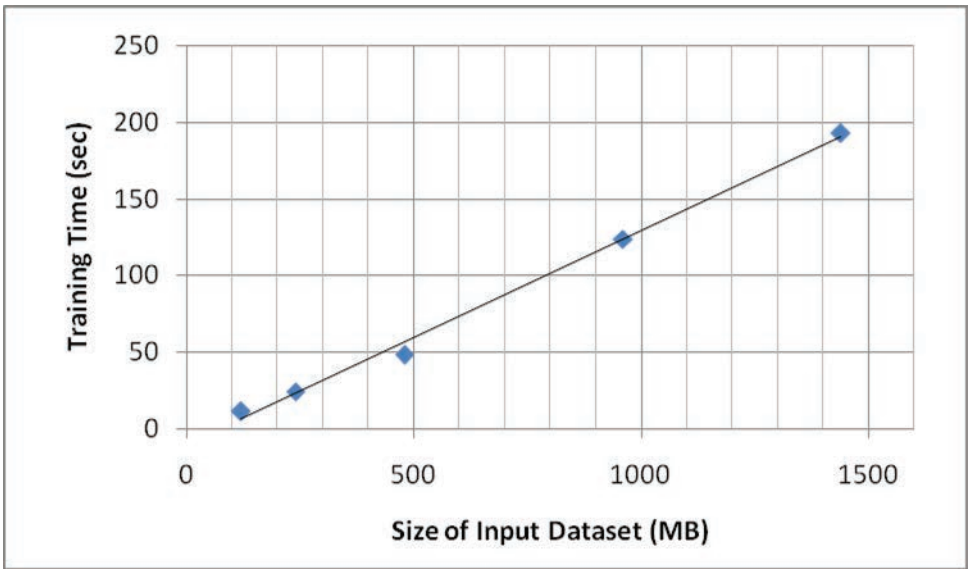

Fig. 15. HC-CART execution time for different size input datasets.

we aim to improve the task and the data scheduling schemes of the HC-CART system so that the final system could exploit the node-level parallelism, especially in cases with low number of variables this could lead to high performance advantages.

7. CONCLUSIONS

In this article we presented a novel architecture for an FPGA-based parallel system for CART, a widely used data mining algorithm. To the best of our knowledge, we are the first to implement a data mining algorithm on a high-end multi-FPGA system, such as the Convey HC-1 server. We are also the first to integrate an FPGA-based accelerator within a widely used software platform like the very widely used R-platform which was used in this work. This integration allows us to compare overall execution times, not just the execution time of the core (or kernel) processing. The implemented system offers impressive performance speedups that can exceed two orders of magnitude when compared to the famous WEKA's code executed on a state-of-the-art CPU when the full I/O latency is fully accounted. Also, our HC-CART system is at least an order of magnitude faster than previous hardware-based solutions that solve the same problem and it outperforms previous multithreaded and/or multiprocessor solutions. The speedup achieved by our novel system, when compared with purely software solutions, grows with the number of variables, thus the higher the dimension of the problem the higher the triggered speedup. Based on the preceding, we believe that our system will be even more efficient in the future, when it will support more complex data mining systems handling even more variables and larger input datasets. Finally, our novel hybrid HW/SW architecture is scalable and generic so it can very efficiently tackle the decision tree classification problem in different application domains since its high performance is not affected in any matter from the input dataset's nature.

REFERENCES

AMADO, N., GAMA, J., AND SILVA, F. 2001. Parallel implementation of decision tree learning algorithms. In *Progress in Artificial Intelligence*, 34–52.

Anyanwu, M. N. and Shiva, S. 2009. Comparative analysis of serial decision tree classification algorithms. *Int. J. Comput. Sci. Secur. 3*, 3, 230–240.

Apache. Apache hadoop. http://hadoop.apache.org/.

Baker, Z. K. and Prasanna, V. K. 2005. Efficient hardware data mining with the apriori algorithm on fpgas. In *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05).* 3–12.

Baker, Z. K. and Prasanna, V. K. 2006. An architecture for efficient hardware data mining using reconfigurable computing systems. In *Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06).* 67–75.

Bekkerman, R., Bilenko, M., and Langford, J. 2011. *Scaling Up Machine Learning: Parallel and Distributed Approaches.* Cambridge University Press.

Ben-Haim, Y. and Yom-Tov, E. 2010. A streaming parallel decision tree algorithm. *J. Mach. Learn. Res. 11*, 849–872.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. 1983. *CART: Classification and Regression Trees.* Wadsworth Press.

Celis, S. and Musicant, D. R. 2002. WEKA-Parallel: Machine learning in parallel. Tech. rep. CS-TR, Carleton College.

Chrysos, G., Dagritzikos, P., Papaefstathiou, I., and Dollas, A. 2011. Novel and highly efficient reconfigurable implementation of data mining classification tree. In *Proceedings of the Conference on Field Programmable Logic and Applications (FPL'11)*. 411–416.

Convey Computer Corporation. http://www.conveycomputer.com/.

Estlick, M., Leeser, M., Theiler, J., and Szymanski, J. J. 2001. Algorithmic transformations in the implementation of k-means clustering on reconfigurable hardware. In *Proceedings of the ACM SIGDA 9th International Symposium on Field Programmable Gate Arrays*. 103–110.

Fang, W., Lu, M., Xiao, X., He, B., and Luo, Q. 2009. Frequent itemset mining on graphics processors. In *Proceedings of the 5th International Workshop on Data Management on New Hardware (DaMoN'09).* ACM Press, New York, 34–42.

Frank, A. and Asuncion, A. 2010. UCI machine learning repository. http://archive.ics.uci.edu/ml.

Gehrke, J., Ramakrishnan, R., and Ganti, V. 1998. RainForest: A framework for fast decision tree.

Hall, M., Eibe, F., Holmes, G., Bernhard, P., Reutemann, P., and Witten, I. H. 2009. The weka data mining software: An update. *SIGKDD Explor. 11,* 1.

He, Q., Tan, Q., Ma, X.-D., and Shi, Z.-Z. 2010. The high-activity parallel implementation of data preprocessing based on mapreduce. In *Proceedings of the International Conference on Rough Set and Knowledge Technology*. Vol. 6401, 646–654.

Joshi, M. V., Karypis, G., and Kumar, V. 1998. ScalParC: A new scalable and efficient parallel classification algorithm for mining large datasets. In *Proceedings of the International Parallel Processing Symposium*. 573–579.

Luo, Y., Xiang, K., and Li, S. 2008. Acceleration of decision tree searching for ip traffic classification. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'08).* 40–49.

Ma, W. and Agrawal, G. 2009. A translation system for enabling data mining applications on gpus. In *Proceedings of the 23rd International Conference on Supercomputing (ICS'09).* ACM Press, New York, 400–409.

Mehta, M., Agrawal, R., and Rissanen, J. 1996. SLIQ: A fast scalable classifier for data mining. In *Advances in Database Technology*. Springer, 18–32.

Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., and Euler, T. 2006. YALE: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06).*

Mikut, R. and Reischl, M. 2011. Data mining and knowledge discovery. *Wiley Interdis. Rev. 11*, 431–445.

Narayanan, R., Honbo, D., Memik, G., Choudhary, A., and Zambreno, J. 2007. An fpga implementation of decision tree classification. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*. 45.

Panda, B., Herbach, J., Basu, S., and Bayardo, R. 2009. Planet: Massively parallel learning of tree ensembles with mapreduce. *Proc. VLDB Endow.* 1426–1437.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Papadonikolakis, M. and Bouganis, C. 2008. A scalable fpga architecture for non-linear svm training. In *Proceedings of the International Conference on Field Programmable Technology.* 337–340.

PAPADONIKOLAKIS, M., BOUGANIS, C., AND CONSTANTINIDES, G. 2009. Performance comparison of gpu and fpga architectures for the svm training problem. In *Proceedings of the International Conference on Field Programmable Technology*. 388–391.

R PROJECT. http://www.r-project.org/.

REXER ANALYTICS. http://www.rexeranalytics.com.

SAEGUSA, T. AND MARUYAMA, T. 2007. An fpga implementation of real-time k-means clustering for color images. *J. Real-Time Image Process. 2,* 4, 309–318.

SALFORD SYSTEMS. http://www.salford-systems.com/

SHAFER, J., AGRAWAL, R., AND MEHTA, M. 1996. SPRINT: A scalable parallel classifier for data mining. In *Proceedings of the 22nd International Conference on Very Large Databases*. 544–555.

SHAN, Y., WANG, B., YAN, J., WANG, Y., XU, N., AND YANG, H. 2010. FPMR: MapReduce framework on fpga. In *Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA'10)*. 93–102.

SRIVASTAVA, A., HAN, E., KUMAR, V., AND SINGH, V. 1998. Parallel formulations of decision-tree classification algorithms. In *Proceedings of the International Conference on Parallel Processing*.

SUN, S. AND ZAMBRENO, J. 2008. Mining association rules with systolic trees. In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 143–148.

SUN, S., STEFFEN, M., AND ZAMBRENO, J. 2008. A reconfigurable platform for frequent pattern mining. In *Proceedings of the International Conference on Reconfigurable Computing and FPGAs*. 55–60.

THERNEAU, T. M. AND GRAMBSCH, P. M. 1997. *An Introduction to Recursive Partitioning Using the RPART Routines*. Mayo Foundation.

TYREE, S., WEINBERGER, K. Q., AGRAWAL, K., AND PAYKIN, J. 2011. Parallel boosted regression trees for web search ranking. In *Proceedings of the 20th International Conference on World Wide Web*. ACM Press, New York, 387–396.

WANG, X. AND LEESER, M. 2007. K-Means clustering for multispectral images using floating-point divide. In *Proceedings of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'07)*. 151–162.

WILEY. Wiley interdisciplinary review. http://wires.wiley.com/WileyCDA/.

WU, X., KUMAR, V., QUINLAN, J. R., GHOSH, J., YANG, Q., MOTODA, H., MCLACHLAN, G. J., NG, A. F. M., LIU, B., YU, P. S., ZHOU, Z. H., STEINBACH, M., HAND, D. J., AND STEINBERG, D. 2010. Top 10 algorithms in data mining. *Knowl. Inf. Syst. 14*, 1, 1–37.

YANG, Q. AND WU, X. 2006. Ten challenging problems in data mining research. *Int. J. Inf. Technol. Decis. Making 5,* 4, 597–604.

YIN, W., SIMMHAN, Y., AND PRASANNA, V. 2012. Scalable regression tree learning on hadoop using openplanet. In *Proceedings of the 3rd International Workshop on MapReduce and Its Applications*.

Received June 2012; revised November 2012; accepted November 2012