Random Forest Architectures on FPGA for Multiple Applications

Xiang Lin
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
xianglin@andrew.cmu.edu

R. D. (Shawn) Blanton
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
rblanton@andrew.cmu.edu

Donald E. Thomas
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
dthomas@andrew.cmu.edu

ABSTRACT

A random forest is a widely used machine learning classifier. An FPGA is a good platform for performance acceleration of random forests due to their inherent concurrent memory accesses and computational parallelism. Amortizing the cost of an FPGA implementation among different applications is desirable, but the context switch time between applications significantly influences the forest architecture. Several architectures for random forests implemented within an FPGA are described, and the area-reconfiguration tradeoffs that determine the suitability of each for multiple applications are characterized. We show that each architecture can maximize the area utilization efficiency of an FPGA given a constraint on the context switch time.

CCS Concepts

•Computing methodologies → Classification and regression trees; •Hardware → Reconfigurable logic and FPGAs;

Keywords

FPGA; Machine Learning; Reconfigurable Computing

1. INTRODUCTION

A random forest is a well-known, widely used machine learning classifier [1]. It is an ensemble method [2] that uses decision trees, each trained independently on a common data set. To improve their performance, researchers have explored hardware implementations of a random forest accelerator, investigating both training and evaluation phases [3,4]. In this work, we focus on accelerating the evaluation phase of a random forest classifier.

Each internal node in a decision tree of a forest is responsible for a comparison test, consisting of a comparison attribute and a comparison value. Each leaf node has an output class label. The comparison attributes, comparison values of each internal node and the class labels of each leaf node are known collectively as the parameters of a given decision tree, all of which constitute what is learned from the training phase. A large degree of parallelism can be exploited in the forest algorithm, between levels of a tree and between trees. The demand for concurrent accesses to the parameters in a forest is well met by the memory resources of an FPGA, which tend to be the limiting factor in an FPGA implementation of a random forest [3].

Because machine learning is being increasingly used to solve a wide range of problems, the capability to context switch between different applications is an attractive approach to amortize the cost of a random forest (or any machine-learning capability implemented in hardware). Fault detection and tolerance systems such as the work presented in [5] demand moderate switch times between tasks on the order of milliseconds. A system that only uses learning for a single, unchanging task (e.g., a system with a machine learning JTAG security scheme [6]) can be implemented minimally since there is no need for context switching. On the other hand, a system that employs multiple machine learning applications may require fast switch times. Various design scenarios have different constraints on how quickly switches need to be performed between or within applications, which therefore demands a spectrum of architectures that aim for efficient implementation while meeting constraints on context switch time.

This work explores three different architectures: memory-centric, comparator-centric, and synthesis-centric. The optimal choice of architecture depends on the needs of the design scenario which include: forest prediction accuracy, size of the reconfigurable fabric available, and the maximum switch time allowed. To the best of our knowledge, this is the first work to examine the tradeoff between machine learning context switch time and design performance.

The rest of this paper is organized as follows: Section 2 describes related work performed in the field of hardware implementations of decision tree/random forest classifiers, and section 3 describes in greater detail the architectures developed in this work. Section 4 describes example applications for various design scenarios, and section 5 compares the architectures. We conclude with a short summary in section 6.

2. RELATED WORK

In [3], a comparison of various forest classifier implementations on multi-core CPU, GP-GPU, and FPGA platforms

GLSVLSI '17, May 10-12, 2017, Banff, AB, Canada

DOI: http://dx.doi.org/10.1145/3060403.3060416

is performed. For the FPGA, a memory-centric architecture is used, balancing the use of BRAMs and distributed flip-flops to store decision tree parameters. The forest is trained using the Compact Random Forest algorithm [3], resulting in a compact forest determined by a maximum depth parameter. The focus in [3] is on tradeoffs between hardware platforms, rather than different architectures on an FPGA. The work of [7] creates a decision tree classifier on an FPGA and examines different architectures that are either pipelined or parallel in nature. These implementations vary classification latency, throughput, and area, but all parameters are stored in memory (distributed and block RAMs). The work in [8] demonstrates an accelerated decision tree classifier by combining architectural techniques and a lightweight addressing scheme for tree nodes. The work of [4, 9] explores FPGA implementations that accelerate the training processes of a decision tree and a random forest respectively, using efficient parallel structures to accelerate the key computational step of determining the optimal comparison value for each internal node of a decision tree.

The aforementioned prior work focuses on the training phase [4,9], or examines the evaluation phase but relies only on memory elements to store parameters [3, 7, 8]. The work here explores alternative options for storing parameters and the trade-offs incurred in area and context switch time.

3. FOREST ARCHITECTURES

Three different architectures are explored in this work. A conventional memory-centric architecture uses distributed and block RAMs to store the parameters of the random forest. A novel but straightforward synthesis-centric architecture allows a larger forest to be implemented on a given FPGA than the memory-centric approach, at the cost of much higher switch time. Additionally, a novel comparator-centric architecture resides in the middle of the spectrum, balancing switch time and area efficiency.

The implementations of random forests in this work use the same majority vote unit, with the emphasis of variation in the implementations of the individual decision trees within the forest.

3.1 Memory-centric

Figure 1 illustrates a memory-centric random forest architecture. Each tree predicts an output class label, and the predictions of all trees are processed by a majority vote unit

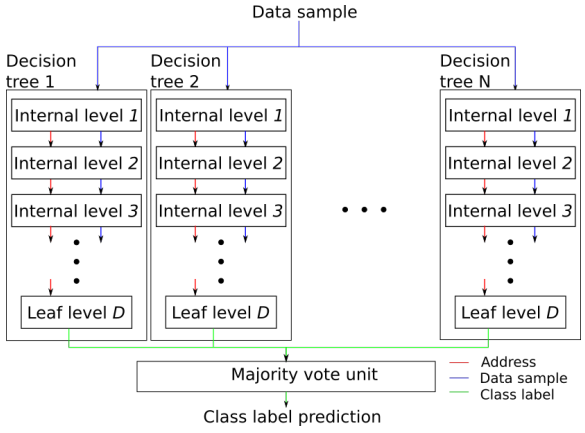

Figure 1: The memory-centric system architecture consists of $D-1$ internal levels and a leaf level D per decision tree.

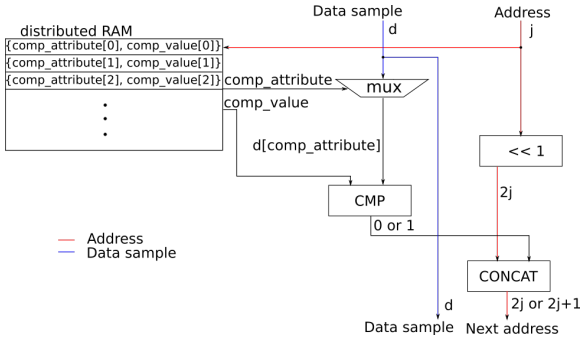

Figure 2: An internal level of a tree in the memory-centric architecture uses the current address to read a comparison attribute and comparison value, which are used to perform the comparison test of the corresponding node.

to produce a final result. Each binary D-level tree has $D-1$ levels of internal nodes and a terminal leaf node. For most applications, very few trees are full binary trees. As a result, each decision tree obtained from training is expanded as follows: each leaf node residing at a depth $i < D$ is converted to an internal node with the comparison bypassed, and has two identical child nodes. Each child node has the class label of the original parent node. This allows the architecture to support any decision tree of up to depth D.

For each internal level, the comparison attributes and comparison values for every node in a given decision tree are stored in memory. A natural choice for storing this information is within BRAM or distributed RAM. Because each data sample being classified can only traverse one node per level in a given decision tree, only a single memory read, and hence memory element (distributed RAM or BRAM), per level is required. The internal levels use distributed RAM, and the leaf levels use BRAM, which results in an approximately equal proportion of utilization of FPGA resources.

Figure 2 illustrates each internal level of a tree. Data samples to be classified are passed through each decision tree from the first level, along with an initial address of 0. The current address is used to read the appropriate node parameters from the memory element, which are then used to test the data sample, determining the address for the next level. The leaf level contains a BRAM, where a look up using the final address received results in the class-label prediction.

Context switching from one forest to another is performed by loading new parameters into the distributed and block RAMs. However, because of the heavy reliance on memory elements, this architecture also demands the largest amount of FPGA resources for a D-level forest.

3.2 Comparator-centric

The comparator-centric architecture reduces the reliance on memory elements, as shown in Figure 3. Note that the parameters for a given forest are static. "Custom comparators" can use LUTs to implement parameter storage and perform the comparison for each internal node, eliminating the ability to change parameters quickly but with the benefit of lower area cost. Conceptually, a custom comparator is a direct representation of an internal node, which is possible only if it assumes the comparison attribute and comparison value are static constants. For example, a 'CC-A2-5' comparator checks whether the second attribute of its input sample is less than 5. Thus, each custom comparator re-

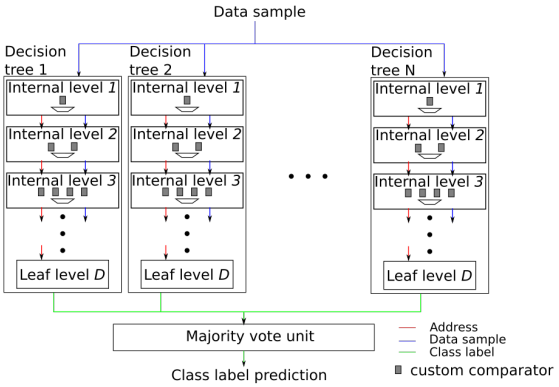

Figure 3: The comparator-centric architecture implements a custom comparator, represented as a gray box, for each internal node in a decision tree.

places the need for a memory element if the parameters are static.

Each internal level $i, (1 \leq i \leq D-1)$ contains 2^{i-1} custom comparators, and a multiplexer to select the output from the custom comparator being used at that level. The leaf levels remain unchanged: a single BRAM contains the class labels of all the leaf nodes per tree. The data flow is similar to the memory-centric architecture: each internal level i processes data sample d in the same way as in the memory-centric architecture, but with addresses of custom comparators instead of memory elements.

To context switch between forests, partial reconfiguration is used to modify each individual custom comparator as needed. An exhaustive library of all possible custom comparators for a given number of attributes and attribute value size can be pre-designed and stored off chip, or designed on-the-fly by a host processor. When context switching, once the parameters of the new application are known, the architecture can switch to the new application by using partial reconfiguration to redefine the custom comparators. While not as fast at switching as in the memory-centric architecture, the comparator-centric architecture makes more efficient use of LUTs to store parameters.

3.3 Synthesis-centric

The synthesis-centric architecture takes the concept of substituting memory elements with combinational logic one step further. Note that a decision tree is essentially a stateless boolean function which takes a data sample as input and returns a class label as output. This boolean function can be optimized by a synthesis tool, eliminating the need for a tree structure and the need for memory storage elements (Figure 4). The resulting boolean function is then implemented using LUTs as logic elements, resulting in a decision tree that has the smallest area footprint.

Data samples to be classified are provided as inputs to the boolean function, with a class label as the output. This design uses almost no memory elements, using only the minimal required for the majority vote unit. However, this architecture has a significant drawback in context switch time. To context switch to a different forest, synthesis of the new forest has to be performed, which can take substantial time depending on the size of the new forest. If the parameters of the new forest are known a priori, the alternative configuration can be synthesized offline and loaded onto the FPGA.

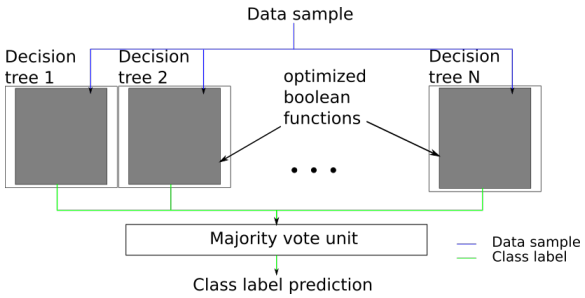

Figure 4: The synthesis-centric architecture implements each decision tree as an optimized boolean function (shown as gray boxes).

However, if this is not the case, then the lengthy process of synthesis will dominate the context switch time.

4. ARCHITECTURE APPLICATIONS

To explore the applicability of forest architectures described in Section 3, this section examines various design scenarios, each of which is characterized by a different switch time constraint. These scenarios include the JTAG security application, the on-chip diagnosis application, and a combination of both. Each architecture is implemented and verified on a Xilinx ZYNQ-7 ZC706 Evaluation Board, which has 218,600 LUTs and 545 36Kb BRAMs.

4.1 JTAG Security

JTAG is an ubiquitous IEEE standard for an on-chip test access port. While primarily targeted for improving testability and post silicon debugging, JTAG also establishes an easily-accessible channel for security attacks [6]. To prevent these attacks, a classifier integrated with existing JTAG hardware is trained to accept JTAG instructions as inputs to determine if the test port is under attack [6].

The nature of JTAG attacks do not change frequently [6]. This implies that changes to the port monitoring hardware are neither frequent nor time-critical, requiring infrequent updates, if any. Therefore, the synthesis-centric architecture is a good match for this design scenario because (i) context switches are infrequent due to the relatively static nature of the test port hacks, and (ii) the performance maximization and area minimization means JTAG operation can be classified quickly at little cost.

Each instance in the dataset represents a series of JTAG instruction sequences, and the goal of classification is to distinguish legitimate and illegitimate use of the JTAG test port. A random forest of ten trees with a maximum depth of 15 levels is trained from this dataset, and ten bits are used as the data width for the hardware implementation.

4.2 On-Chip Diagnosis

In this scenario, on-chip diagnosis [10] uses a classifier to predict the location of a faulty sub-circuit from a group of locations when self-test of a sub-system fails. Different classifiers are needed for each sub-system. It is too costly to simultaneously implement all the required classifiers to cover all of the sub-circuits, especially if only one sub-circuit fails at a time. It is therefore better to have a configurable classifier that can switch based on which sub-system requires diagnosis. Furthermore, while a fast context switch is desirable, it is not extremely critical. These characteristics of on-chip diagnosis suggest that the comparator-centric architecture is a good match.

	Area utilization					
	Total LUTs	LUTs as logic	LUTs as memory	Flip-flops	BRAM	Context switch time (s)
Security (synthesis-centric)	5700 (2.61%)	5700	0	39 (<0.01%)	0 (0.0%)	1.615
Diagnosis (comparator-centric)	88594 (40.53%)	88594	0	88 (0.02%)	10 (1.85%)	0.295
Security and diagnosis (memory-centric)	97720 (44.70%)	10000	87720	83 (0.02%)	320 (58.71%)	0.00164

Table 1: The memory-centric architecture has the lowest context switch time but uses the most FPGA resources, the comparator-centric architecture achieves a middle ground among the architectures, and the synthesis-centric architecture uses the least resources.

Diagnosis data for the L2B cache controller circuit [11] is used in the implementation of the comparator-centric architecture. The goal of this classification problem is to identify which sub-circuit in the failed circuit (the L2B in this case) contains the failure. A random forest of ten trees with a maximum depth of 15 is trained from this dataset, and 12 bits are used for the data width for the hardware implementation.

4.3 Security and Diagnosis

Now consider a design scenario that requires JTAG monitoring and implements on-chip test and diagnosis. For this case, a fast context switch time is necessary. For example, consider the situation where a failure has been detected and on-chip diagnosis is being performed. If JTAG operation begins while diagnosis is being performed, then it will be necessary for the system to quickly context switch to protect the JTAG from potentially being attacked. Such a situation necessitates a fast switch time, which makes the memory-centric architecture a natural choice.

A memory-centric random forest that can switch between on-chip diagnosis and security must support the larger of the parameters of either application. Hence, the implemented architecture can support forests of up to ten trees with a maximum depth of 15, and a data width of 12 bits.

5. ARCHITECTURE COMPARISONS

Table 1 summarizes the implementation results for each architecture, reporting area utilization and context switch time. While area utilization comparisons are straightforward, comparing context switch times is less so. For the synthesis-centric architecture, synthesis of the forest requires significant compute time (∼300 seconds). Assuming however that synthesis is accomplished off-line for the classifier or any update to the classifier, the time needed for switching to an updated JTAG classifier is the time to completely re-program the FPGA, which is 1.62 seconds. The context switch time for the comparator-centric architecture depends on the speed of partial reconfiguration. The worst case assumes the use of an external host processor as the partial reconfiguration controller and a JTAG port operating at 66 Mbps to program the FPGA. Partial reconfiguration for all nodes requires 59 seconds if only a single node is reconfigured per reconfiguration cycle. However, the smallest partial reconfiguration frame is 50x1 CLBs, which implies that 200 LUTs can be reconfigured in a single partial reconfiguration cycle. Thus, the ideal reconfiguration latency for this architecture is 0.295 seconds. Context switch time for the memory-centric architecture is measured by the time required to load new values into each of the memory elements. Assuming the worst case, loading one memory value per clock cycle for each of the 10×2^{14} nodes in the random forest would require 1.64 ms at the given clock frequency.

6. SUMMARY

Different machine learning applications have solutions that result in various design scenarios with different constraints on context switch time and area utilization. We explore three random forest architectures that cater to different design scenarios, and suggest examples of applications that represent these design scenarios. Furthermore, we demonstrate the tradeoffs between area utilization and context switch time between architectures, and show how each architecture maps well to a different design scenario.

7. ACKNOWLEDGMENTS

This work was funded in part by the National Science Foundation under contract no. 1314876.

8. REFERENCES

[1] L. Breiman, "Random Forests," in *Machine learning*, vol. 45, pp. 5–32, 2001.

[2] T. G. Dietterich, "Ensemble Methods in Machine Learning," in *Multiple classifier systems*, pp. 1–15, Springer, 2000.

[3] B. Van Essen, *et al.*, "Accelerating a Random Forest Classifier: Multi-core, GP-GPU, or FPGA?," in *Field-Programmable Custom Computing Machines*, pp. 232–239, 2012.

[4] C. Cheng and C. Bouganis, "Accelerating Random Forest Training Process Using FPGA," in *Field Programmable Logic and Applications*, pp. 1–7, 2013.

[5] S. Shreejith, *et al.*, "Accelerated Artificial Neural Networks on FPGA for Fault Detection in Automotive Systems," in *Design, Automation & Test in Europe*, pp. 37–42, 2016.

[6] X. Ren, *et al.*, "Detection of Illegitimate Access to JTAG via Statistical Learning in Chip," in *Design, Automation & Test in Europe*, pp. 109–114, 2015.

[7] J. Struharik, "Implementing Decision Trees in Hardware," in *International Symposium on Intelligent Systems and Informatics*, pp. 41–46, 2011.

[8] F. Saqib, *et al.*, "Pipelined Decision Tree Classification Accelerator Implementation in FPGA (DT-CAIF)," in *IEEE Transactions on Computers*, vol. 64, pp. 280–285, 2015.

[9] R. Narayanan, *et al.*, "An FPGA Implementation of Decision Tree Classification," in *Design, Automation & Test in Europe*, 2007.

[10] X. Ren, *et al.*, "Improving Accuracy of On-chip Diagnosis via Incremental Learning," in *VLSI Test Symposium*, 2015.

[11] "OpenSPARC T2." http://www.oracle.com/technetwork/systems/opensparc/opensparc-t2-page-1446157.html, Oracle.