Scalable Inference of Decision Tree Ensembles: Flexible Design for CPU-FPGA Platforms

Muhsen Owaida, Hantian Zhang, Ce Zhang, Gustavo Alonso
Systems Group, Department of Computer Science, ETH Zurich
{firstname.lastname}@inf.ethz.ch

***Abstract*—Decision tree ensembles are commonly used in a wide range of applications and becoming the de facto algorithm for decision tree based classifiers. Different trees in an ensemble can be processed in parallel during tree inference, making them a suitable use case for FPGAs. Large tree ensembles, however, require careful mapping of trees to on-chip memory and management of memory accesses. As a result, existing FPGA solutions suffer from the inability to scale beyond tens of trees and lack the flexibility to support different tree ensembles. In this paper we present an FPGA tree ensemble classifier together with a software driver to efficiently manage the FPGA's memory resources. The classifier architecture efficiently utilizes the FPGA's resources to fit half a million tree nodes in on-chip memory, delivering up to 20x speedup over a 10-threaded CPU implementation when fully processing the tree ensemble on the FPGA. It can also combine the CPU and FPGA to scale to tree ensembles that do not fit in on-chip memory, achieving up to an order of magnitude speedup compared to a pure CPU implementation. In addition, the classifier architecture can be programmed at runtime to process varying tree ensemble sizes.**

I. Introduction

Decision tree ensembles such as Random Forest and XGBoost [5], [6] deliver higher accuracy than classifiers based on a very deep, single tree. XGBoost is used in half of the winning solutions in the Kaggle 2015 competition [4]. However, the sequential evaluation of tree levels, the increase in cache misses due to random memory accesses, and the limited number of CPU cores processing trees in parallel significantly degrade performance for large decision tree ensembles. As a result, existing CPU based solutions may not be adequate for applications with high data rates, such as particle classification in high energy physics. For example, the LHC collider produces approximately 10^{11} collisions per hour [2]. A decision tree ensemble classifier used to identify the Higgs boson would need to be able to process roughly 27.8×10^6 particles per second. In 2014, a winning solution in the Kaggle's HiggsML challenge used gradient boosting decision trees [19]. For this type of application, an FPGA is a strong candidate to deliver the required throughput because it is better suited to exploit parallelism in large tree ensembles.

Current FPGA implementations of decision trees [8], [3], [18], [13], use either a pipeline of compute elements each processing one tree level or allocate a compute element for every tree node. Both approaches are not scalable to tree ensembles beyond tens of trees as they quickly run out of on-chip memory and logic resources. Thus, many existing solutions demonstrate speedup only for small tree ensembles (of 3 to 16 trees) [8], [13]. Moreover, most current FPGA implementations are designed for fixed size tree ensembles. Decision tree ensembles differ between data sets and they often change as data sizes grow and new features arise. Hence, FPGA architectures implementing a single tree ensemble are hard to use in practice.

While FPGAs are strong candidates to parallelize the processing of independent trees and data samples, scarce on-chip memory requires proper consideration when mapping trees to memory blocks on the FPGA. Hybrid processing of tree ensembles using both the CPU and FPGA is a potential approach to scale to very large ensembles and deep trees. It also provides flexibility to adapt to changing structure of the ensembles.

Based on this, in this paper we introduce a hybrid classification engine for XGBoost on a CPU+FPGA shared memory platform (Intel's Xeon+FPGA platform [14]). The classification engine implements the inference of tree ensembles generated by XGBoost, using aggregation to produce the final classification result for each data point. The engine can be extended with a different voting technique, if necessary, as it decouples trees evaluation and voting.

In developing the hybrid classification engine we have three objectives: 1) scalability to large tree ensembles with deep trees through hybrid execution on CPU and FPGA; 2) Programmability at run time with tree ensembles of different sizes; 3) Efficient FPGA resources management to achieve maximum performance.

In addition to the FPGA architecture, the classification engine is accompanied by a software driver abstracting the classification process as a function call. The driver uses the tree ensemble's structure (i.e., number of trees and tree depth) to determine the proper mapping of tree nodes to the FPGA's memory resources. If the tree ensemble does not fit in the FPGA's on-chip memory, it resorts to hybrid CPU+FPGA processing.

Experimental evaluation demonstrates that the designed classifier delivers up to 20x speedup over 10-threaded CPU only implementation when processing the complete tree ensemble on the FPGA. For tree ensembles that do not fit in the FPGA's on-chip memory, the hybrid CPU+FPGA processing delivers an order of magnitude speedup over 10-threaded CPU-only implementation.

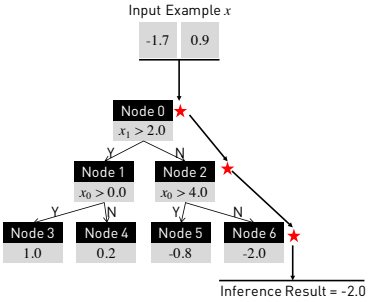

Fig. 1: An example of decision tree. Nodes 0-2 are decision nodes and nodes 3-6 are end nodes.

II. Background

A. Machine Learning: A Simplified Data Model

We present a simplified data model for machine learning to set up the terminology for the rest of the paper. We focus on *inference*, a process taking as input an *example* and outputting a *label*. An example x is a d-dimensional vector: $x \in \mathbb{R}^d$ and each dimension x_i is called a *feature*. A machine learning *model* M specifies a function $f_M : \mathbb{R}^d \mapsto \mathbb{R}$, and we call $f_M(x)$ the *inference result* of the example x.

An ensemble method is a family of algorithms that combines *multiple* machine learning models. We focus on gradient boosting which has a simple inference procedure: let M_1, ..., M_k be k machine learning models, the inference result of the ensembled model is $f(x) = \sum_i f_{M_i}(x)$. That is, we first run inference for each model M_i and then sum up all the results.

B. Decision Tree

Decision trees [7] are one of the most popular machine learning models and have been widely used for a range of machine learning tasks such as classification [7] and regression [9]. Figure 1 illustrates a binary decision tree model. Each non-leaf node is called a *decision node* and each leaf node is called an *end node*. Each decision node contains criteria for choosing either the left or right node in the next level, and each end node contains the classification or regression result (i.e., label). During inference, an example traverses from the root to an end node according to the criteria of decision nodes. In this paper, we focus on *non-oblique* trees, in which each decision node only compares a single feature.

C. Decision Tree Ensembles and XGBoost

A Gradient Boosting Decision Tree (GBDT) is an ensemble algorithm based on gradient boosting and uses decision trees as its base classifier. XGBoost [6] is a popular and efficient implementation of GBDT. A model in XGBoost contains K decision trees M_1, ... M_K. To run inference, it first runs inference over all K decision trees, and it then sums up the inference results.

III. Classifier Engine Overview

We have developed the classifier engine on HARP v1, the Intel's Xeon+FPGA platform[1]. Figure 2 shows the classifier's FPGA architecture.

The classifier's FPGA architecture is accompanied with a software driver on the CPU side. The driver exposes the inference of a decision tree ensemble as a function call abstracting low level CPU-FPGA communication away from the application level. In a simplified syntax, it exposes the following function for the application developer:

```
Result = classify(Model, TestData)
```

The user passes a pointer to the test data, and a data structure describing the tree ensemble model including parameters such as the number of trees, the tree depth, and all the decision and leaf nodes. The driver then uses the model parameters to make a few decisions regarding the inference of the tree ensemble such as whether the model fits in the FPGA's memory, how to perform the CPU-FPGA hybrid processing, how to map the trees to the FPGA's memory, and how to schedule the processing of different data examples on the parallel processing elements of the FPGA architecture. In section V we discuss in more details the driver's operation.

Once all the above decisions are made, the driver composes a classification request and writes it to a designated shared memory location monitored by the classifier *I/O Unit* on the FPGA. This request includes the above parameters passed by the user, in addition to a few parameters with the decisions made by the driver. The classifier *I/O Unit* then retrieves this request, loads the tree model, starts reading test data, and writing back inference results.

The classifier FPGA architecture consists of 8 *Compute Units*, each includes 8 decision tree processing elements (*DT-PE*) as Figure 2 shows. A *DT-PE* unit is programmed dynamically to evaluate one or more decision trees per data example. A DT-PE unit outputs a leaf value for each tree evaluated. The *Reducer* unit is a tree of floating point adders. It sums up the leaf values from all 8 DT-PE units in the same *Compute Unit*. A single *Compute Unit* processes the whole tree ensemble if it fits in it's local memory. Multiple *Compute Units* are combined to process larger tree ensembles. The actual number is determined by the software driver using the tree ensemble's size.

The *Scheduler* executes the software driver's decisions on how to distribute the ensemble trees to the *Compute Units* and how to parallelize the processing of different data examples. The *Combiner* consists of a single floating point adder and iteratively accumulates partial results generated by the *Compute Units* to produce the final result per data example. When initially triggered, the *I/O Unit* loads the tree ensemble and stores it in the *Compute Units*' local memory. Then, it reads the data examples and writes back the classification results. Since

[1]Results in this publication were generated using pre-production hardware and software donated to us by Intel, and may not reflect the performance of production or future systems.

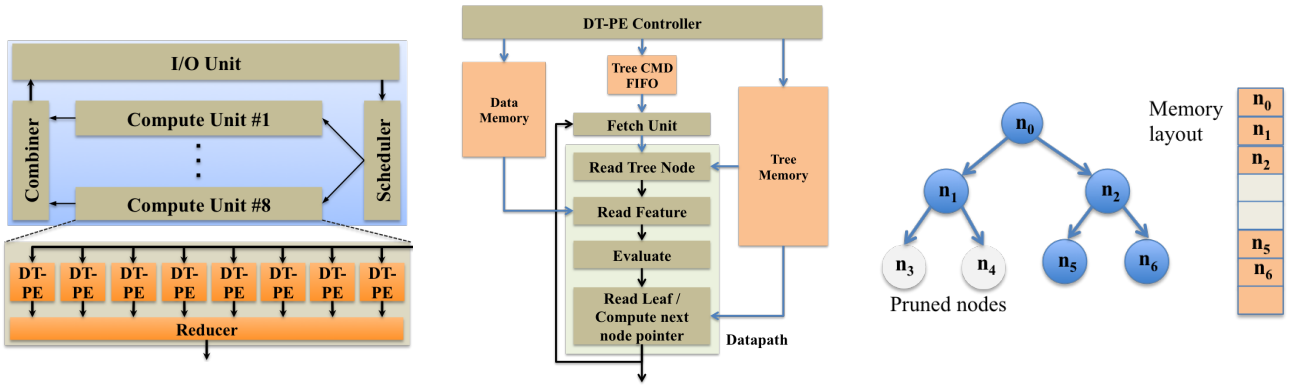

Fig. 2: (Left) Classifier FPGA Architecture. (Middle) Decision Tree Processing Element, DT-PE. (Right) Tree memory layout.

the FPGA has coherent memory access to the same address space as the CPU, no CPU effort is required in moving data in or out of the FPGA.

The classifier architecture operates in two processing modes: full FPGA processing when the whole tree ensemble fits in the *Compute Units'* combined local memories; and hybrid CPU-FPGA processing, otherwise. Section V-B discusses in more detail how hybrid processing is performed.

IV. Decision Tree Processing Element (DT-PE)

The architecture of the DT-PE unit is depicted in Figure 2. The DT-PE unit consists of two types of components: local memories store the tree ensemble and the data example's features, and a datapath evaluates a tree node for input data examples.

A. DT-PE Memory Layout

There are two types of local memories in the DT-PE unit: *tree memory*, and *data memory*. The tree memory either stores one big tree up to 8192 nodes (decision and leaf nodes), or multiple trees sharing equally overall memory capacity. The tree nodes are stored as a one dimensional array (Figure 2). The storage scheme assumes a full binary tree with no missing nodes and every node stored at a dedicated location. Each tree consumes a memory footprint equaling $2^{MAX\_TREE\_DEPTH}$. If a tree node is pruned, its dedicated memory location stays empty and is not used by another tree node. Such a storage layout allows the calculation of the child node pointer using the parent pointer as follows:

$$child\_pointer = (parent\_pointer << 1)+1+GO\_RIGHT$$

where GO_RIGHT either equals 1 or 0, based on the comparison result of the parent node threshold and the corresponding feature values. An alternative memory layout is to avoid holes in the data structure and only allocate as much memory space as the tree has nodes. Because of the random nature of tree pruning, it is difficult to calculate the child pointer. Hence, every tree node has to store a pointer for its left child, thereby increasing the memory space requirement per tree. If the trees are highly sparse, the latter memory layout will be more efficient than the former one. Moreover, the latter memory layout might end up with an unbalanced mapping of trees to DT-PE units as the tree sizes will be different. Our memory layout captures the worst case scenario with full binary trees for large tree ensembles. We leave the design for ensembles with highly sparse trees to future work.

The *data memory* stores incoming data examples and has a capacity of 4096 features (floating point). The data memory has one write and one read port, allowing prefetching the next data examples while available data examples are being processed.

The data memory is designed to saturate the QPI bandwidth (6 GB/s). Operating at 200 MHz, the data memory has to deliver 32 Bytes per cycle to saturate the QPI bandwidth. Hence the data memory has a data line width of 256-bits (i.e. 32 bytes) which requires stitching together 7 Block RAMs (BRAMs). Since each BRAM has 512 entries and the size of a feature is 4 bytes, we compute the maximum capacity of the data memory to be equal to 4096 features.

The size of the tree memory is selected as a trade-off between allocating as many DT-PE units as possible to parallelize the evaluation of trees, and the tree depth that can be fitted in the DT-PE tree memory. We selected 13 levels as the maximum tree depth (or $2^{13} = 8192$ nodes), which we considered relatively deep enough for XGBoost generated trees. Given that we need 6 Bytes to store a tree node, a total of 48 KB is needed for the tree memory, or 19 BRAMs. Given these BRAM resources requirements for a single DT-PE, we allocated 64 DT-PE units consuming nearly 70% of the FPGA memory resources. These numbers can be easily adjusted for other FPGA platforms.

B. DT-PE Datapath

The DT-PE's datapath pipeline consists of four operations: reading a tree node from the tree memory; reading the corresponding data example feature from the data memory; comparing the tree node threshold to the feature value; and either computing the next decision node pointer or reading the leaf node. These are the operations required to evaluate one tree level for the input data example. To evaluate all the tree

levels, the next decision node pointer is fed back to the first operation to continue processing subsequent levels. These four operations are iterated until a leaf node is reached, or the last tree level stored in the tree memory is reached in the hybrid processing mode. The datapath pipeline is 8 clock cycles deep. Hence, it requires $8 * N$ cycles to process a tree of N levels.

Once all the features of a data example are stored in the data memory, the DT-PE *Controller* pushes a command into the *Tree CMD FIFO* for each tree stored in the DT-PE unit (Figure 2). Since the operation of the Controller is independent from the datapath pipeline, the Controller will push a new set of commands in the Tree CMD FIFO for a subsequent data example once it arrives to the data memory. This overlaps and pipelines the processing of two or more data examples. A command specifies to the datapath which tree to evaluate on which data example. The *Tree CMD FIFO* allows us to order the processing of different data examples such that all trees stored in the DT-PE unit are evaluated for the same data example before proceeding to the next data example.

The *Fetch Unit* reads tree commands from the *Tree CMD FIFO* until the datapath pipeline is full, then it feeds the datapath with commands to evaluate subsequent levels for the trees currently processed in the pipeline. When there is no subsequent levels to evaluate for a tree in the pipeline, the *Fetch Unit* resumes reading commands from the *Tree CMD FIFO*. Since the datapath pipeline is 8 cycles deep, it pipelines the processing of up to 8 trees which can be either different trees evaluated for the same data example or one or more trees evaluated for different data examples.

V. Classifier Software Driver

A. Mapping Trees on FPGA Memory

The classifier software driver is designed to maximize the processing throughput in data examples per second. The driver considers the following architectural features: the number of DT-PE units (64), the pipeline depth of the DT-PE datapath (8 cycles), and the Combiner processing rate. Since the Combiner does not parallelize the aggregation of partial results, the driver avoids spreading the tree ensemble across all Compute Unites and exploits pipeline parallelism in the datapath. We try to pack and fit the whole tree ensemble in 1, 2, 4, or 8 Compute Units. We select these numbers so we can have multiple clones of the tree ensemble, each occupying the same number of Compute Units. Multiple clones of the tree ensemble can then be used to parallelize the processing of different data examples.

Given a tree ensemble, the driver maps it to the DT-PEs as follows: initially, it spreads the tree ensemble over all Compute Units. Then, it packs trees in half the number of the Compute Units if the DT-PE tree memory can accommodate the additional trees. It repeats the packing step until the DT-PE memory is full or if at least 8 trees are packed in the DT-PE tree memory, since 8 trees saturates the DT-PE datapath pipeline. The outcome of the mapping process is two numbers: the number of trees mapped to the DT-PE's tree memory (N) and the number of Compute Units (N_C) required to store the whole tree ensemble.

The *Scheduler* uses N_C to determine how to distribute trees and data examples to *Compute Units*. For example, if N_C equals 1, this means the tree ensemble will be replicated to all Compute Units. Then each Compute Unit processes a subset of the input data examples in parallel. The *Scheduler* iterates over the *Compute Units* and their DT-PE units in a round robin fashion, storing one tree at a time until all the ensemble trees are stored in the DT-PE units. The use of a round-robin distribution scheme balances the distribution of the tree ensemble across the DT-PE units. The *Combiner* uses N_C to determine from how many *Compute Units* it should aggregate partial results.

B. Hybrid CPU-FPGA Processing

We resort to hybrid CPU-FPGA processing in two cases: the tree ensemble size is more than half a million nodes (i.e., the maximum capacity of the FPGA architecture) but trees are not deeper than 13 levels, and when the trees in the ensemble are deeper than 13 levels (i.e., a full binary tree has more than 8192 nodes, the maximum capacity of the DT-PE tree memory).

For the first case, we partition the ensemble into smaller sub-ensembles (i.e., a smaller number of trees) that fit in the FPGA's memory. Then each partition of the ensemble is processed independently on the FPGA and the partial results from each partition are accumulated at the end by the CPU.

For the second case, the first 13 levels of a single tree consume all the DT-PE's tree memory capacity. Hence, 64 trees can be processed and stored in the FPGA architecture at the same time. If the tree ensemble is larger than 64 trees, we partition it into ensembles of size 64 trees. Then each partition is processed on the FPGA independently producing pointers to decision nodes in the subsequent level (i.e., level 14). The *Combiner* unit then collects all these pointers and writes them back to the shared memory. Then, the resulting pointers from all partitions are processed on the CPU, evaluating levels from depth 14 onward and aggregating leaf values.

VI. Experimental Evaluation

A. Experimental Setup

We implement our decision tree ensemble classifier on Intel's HARP v1. It is a two-socket machine with a 10-core Intel Xeon E5-2680 v2 CPU (clocked at 2.8 GHz) in one socket and an Altera Stratix V 5SGXEA in the other. The two sockets are connected through QPI. On the CPU socket, 96 GB of main memory are installed, accessible to the FPGA through the QPI link. No DDR memory is attached to the FPGA socket. Memory access from the FPGA is bound by the bandwidth of the QPI link which we measured to peak at 6 GB/s. In contrast, the CPU sees up to 25 GB/s memory bandwidth. The FPGA design is configured with 64 DT-PE units and clocked at 200 MHz. The FPGA is programmed once and the same design is used for all the experiments with different tree ensemble structures. Loading a new tree ensemble into

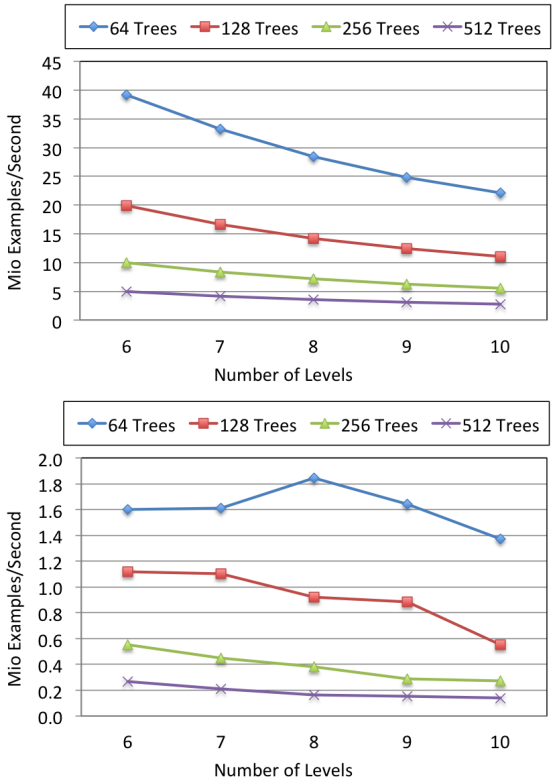

Fig. 3: Performance scalability with increasing tree depth for data with 32 features. FPGA (above), and CPU (below).

the FPGA local memory takes nearly 400 μs for large tree ensembles consuming the whole tree memory capacity. In the experiments we use the throughput of processed data examples (Million data examples per second) as the performance metric. We compute this metric, by dividing the total number of processed data examples over the end-to-end time from calling the *Classify()* function until the FPGA writes the last result. This measurement includes, time taken by the driver to make its decisions and communicate the classification request to the FPGA, loading trees to the FPGA memory and processing the data examples.

B. Scalability Evaluation

In this section we study the scalability of the FPGA architecture with varying properties of the tree ensemble such as the number of trees, the tree depth, and the number of features per data example. We run the experiments on 1 million data examples in all configurations. We used the Scikit-learn [16] machine learning package in Python to generate data sets for our experiments. We generated a training data set of 100K examples, and a testing data set of 1 M examples for different number of features (16, 32, 64, 128, 256, and 512 features). As a baseline, we used XGBoost to train the models and perform inference. In the training we did not enable tree pruning to evaluate the performance requirements for full binary trees. For CPU performance, we run XGBoost with 10 threads on the HARP's 10 core Xeon CPU.

Scaling throughput with tree depth. Figure 3 shows the FPGA and CPU throughputs for varying tree depth and different number of trees. We run this experiment for 32-feature data examples. All the tree ensemble configurations shown in the figure fit in the FPGA's memory and are processed fully on the FPGA.

From the results we can draw a few conclusions. The FPGA throughput drops linearly with increasing tree depth and number of trees. Similarly, increasing tree depth and the number of trees also degrade the CPU performance. Increasing either tree depth or number of trees increases the number of compute units required to store the whole tree ensemble. Meaning, evaluating the whole tree ensemble per data example reduces the available parallelism in the FPGA as the ensemble size increases, which reduces the overall throughput of the FPGA. In the CPU implementation, the data examples are partitioned equally between the 10 threads. However, because of the random access to the tree ensemble and its dependency on the data, the thread which suffers from the most cache misses will be the slowest and the CPU execution time equals the slowest thread execution time. As a result, the trend in CPU performance is not consistent for small tree ensembles (e.g., 64 trees) as they are more sensitive to cache misses. For larger tree ensembles, all the CPU threads suffer from cache misses and read more often from the L3 cache. From Figure 3, the FPGA reaches a 15 to 25 speedup over a 10-threaded CPU.

Scaling throughput with data features. We have also studied the behavior of the FPGA performance for varying number of data features. Figure 4 shows the results for different data example sizes at tree depth of 10. In the FPGA throughput plots we can distinguish two trends. First, the FPGA performance stays the same and represents the maximum compute capacity of the FPGA for the specific number of trees. Second, the FPGA performance starts decreasing when it becomes bound by the QPI bandwidth. Figure 5 shows the maximum achievable FPGA throughput considering the QPI bandwidth on the HARP v1 machine (future version of the HARP machine will have more memory bandwidth). The CPU sees a higher bandwidth to main memory. As a result, the CPU performance is not affected by the number of features. For small tree ensembles, the L1/L2 cache misses lead to a small inconsistency in the CPU performance. For large number of features, the DT-PE unit starves for data, waiting for the complete data example to be loaded into the data memory before processing it. Although the DT-PE overlaps prefetching and processing of the data examples, prefetching a large data example takes much longer than processing. Figure 6 shows how the FPGA loses its advantage as the number of features increases. For ensembles with 512 trees, the FPGA speed up does not drop even for 512 features, because the FPGA benefits from high parallelism and bandwidth to its local memory compared to the 10-threaded CPU.

The conclusions we draw from the scalability analysis are the following. Our scheduling methods efficiently exploit the parallelism in the FPGA architecture. For large tree ensembles, we parallelize the processing of up to 512 different trees per

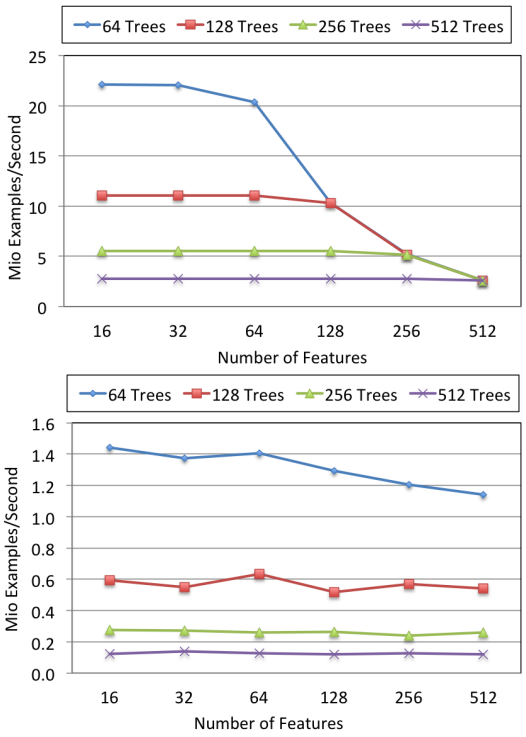

Fig. 4: Performance scalability with the number of features per data example for 10-levels trees. FPGA (above), and CPU (below).

cycle per data example. As a result, the FPGA sustains a fixed throughput until the data examples are large enough (more than 512 features), such that the throughput is bound by the QPI bandwidth. For small tree ensembles, the FPGA architecture parallelism is used to parallelize the processing of different data examples. This is why the FPGA throughput becomes more sensitive to the QPI bandwidth. The FPGA advantage comes from the high parallel processing power, up to 1.28 billion 10-levels deep trees per second. In addition, the DT-PE units reach nearly 119 GB/s aggregate throughput from local memory.

C. Hybrid Mode Evaluation

In this section we study the scalability of the hybrid CPU-FPGA processing mode. We experimented with an ensemble

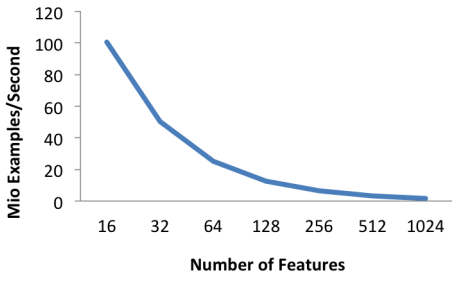

Fig. 5: QPI bound on maximum achievable FPGA throughput.

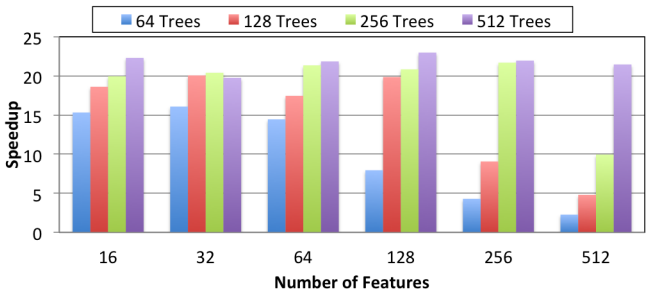

Fig. 6: FPGA speedup over 10-threaded CPU.

of 512 trees for 256 features per data example. We chose this number of features because it is hard to grow trees beyond 11 levels for less features.

Figure 7 shows the CPU and FPGA throughput for a wide spectrum of tree depths. From the figure we notice two phases: up to tree depth 10, the ensemble is fully processed on the FPGA. For deeper trees, we use CPU-FPGA processing to evaluate the whole ensemble as described in section VI-C. The CPU part is multi-threaded using 10 threads.

We notice that the FPGA throughput declines rapidly with increasing tree depth in the CPU-FPGA processing mode (Figure 7). When the tree ensemble does not fit in the FPGA's memory, the driver partitions the tree ensemble into smaller sub-ensembles that fit in the FPGA's memory. For the 512 and 11-levels deep trees, the driver creates two partitions each of size 256 trees. When the tree depth increases by 1, the number of partitions doubles, and the number of trees per partition is halved. Smaller number of trees per partition means more data examples can be pipelined and processed in parallel in the DT-PE's datapath. However, because of limited memory bandwidth, data examples of size 256 features do not arrive fast enough to exploit the pipeline parallelism in the DT-PE's datapath. Yet, doubling the number of partitions, and repeated traversal of the data, almost doubles the FPGA processing time. We expect this limited main memory bandwidth of

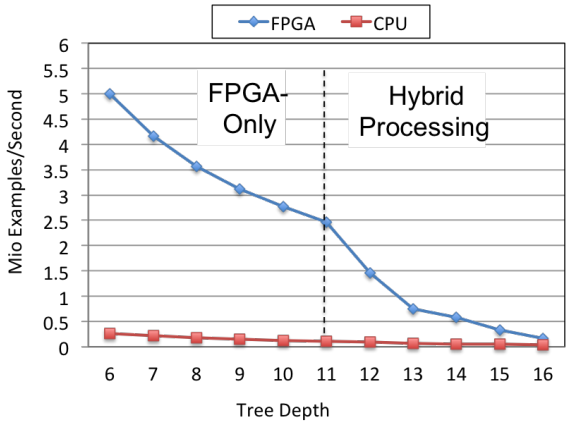

Fig. 7: Performance scalability for deep trees using hybrid CPU-FPGA processing.

TABLE I: Real data sets used in experiments.

Dataset	#Examples	#Features	#Trees	Tree Depth
HiggsML [19]	1,000,000	28	512	7
Physics [12]	855,819	74	200	10
Schizophrenia [10]	119,748	410	200	4

HARP v1 will disappear in the future version of the HARP machine (HARP v2) and the FPGA will have a memory bandwidth similar to the CPU.

From 14-levels depth on, a single tree is larger than the capacity of the DT-PE's tree memory, which accommodates up to 13-levels deep trees. From this depth onward, the driver stops partitioning the tree ensemble further, and it processes the extra levels on the CPU. Now, in addition to accumulating leaf values, the CPU implementation also evaluates internal tree levels. The hybrid processing performance continues to decline as more tree levels are moved to CPU processing.

D. Real World Workloads

We experimented with three real data sets from physics and psychology. Table I describes the properties of the data sets and the used tree ensembles. We used 5-fold cross validation on the training set to determine the best tree ensemble's parameters (i.e., number of trees and tree depth) for each data set. The final chosen tree ensemble produced the best results for each data sets.

For all three data sets, the generated tree ensemble fits in the FPGA's on-chip memory. The results in Figure 8 is consistent with what we observed in the scalability study. The result for the Schizophrenia data set is particularly interesting. The shallow trees of the ensemble (maximum depth equals 4) lead to a good CPU performance since the whole tree ensemble fits in the L1 cache. However, the large number of features (410 features) and limited QPI bandwidth affect the FPGA throughput, hence resulting in only a 1.4X speedup.

The tree ensemble generated for the Physics data set is highly pruned, leading to a better CPU performance than expected for the given tree ensemble parameters. On the other hand, the DT-PE is designed to handle a worst case scenario

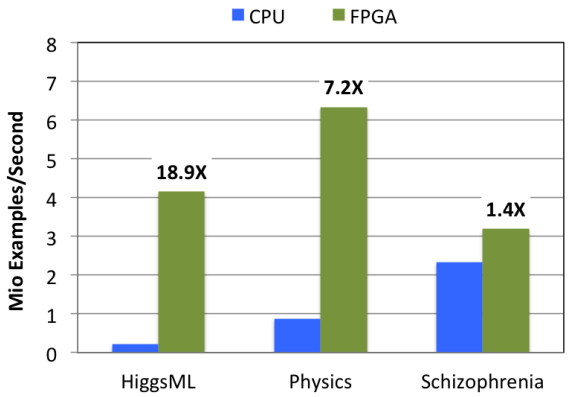

Fig. 8: FPGA vs. CPU performance on 3 real data sets.

TABLE II: Consumed FPGA resources by the classifier.

Module	ALMs		M20k	
Tree Memory	0	0%	19	0.75%
Data Memory	0	0%	7	0.27%
DT-PE	899	0.4%	28	1.1%
Reducer	2,995	1.2%	1	0%
Compute Unit	9,953	4.2%	226	8.8%
Engine (total)	81,472	34.7%	1844	72%

of non pruned trees, and it does not benefit from pruning. Hence, the achieved overall speed up is less than observed in the scalability study, yet it is an order of magnitude higher compared with the CPU result.

The HiggsML data set is a typical case similar to what we observed in Figure 3, as the tree ensemble is not highly pruned.

E. Resources Utilization

Table II lists the amount of resources consumed by the different components (single instance) of the classifier engine in addition to the total FPGA resources consumed. Nearly 66% of the engine memory resources are devoted to storing the tree ensemble aggregating nearly 3 MB of storage capacity. 24% of the engine memory blocks are used to store data examples while being processed. This amount of memory was required to support up to 4096 features per example and to operate at line rate exploiting the maximum QPI bandwidth.

VII. Related Work

The use of FPGAs to accelerate decision tree based classifiers has been extensively studied. Amato et al. [1] proposed an FPGA architecture where every tree node is allocated its own processing unit which generates a local decision, a boolean network of AND operations then consumes the individual decisions to generate the final class for the input data example. This approach is not scalable to large trees. Barbareschi et al. [3] presented an FPGA classifier based on decision tree ensembles using majority voting and the decision tree design from Amato et al. The tree ensembles which they discussed were limited to up to 50 trees, and an upper bound of roughly 3400 nodes. Struharik [18] suggested a hardware architecture for a decision tree that uses less resources than Amato et al. The proposed architecture pipelines the processing of tree levels, and allocates a memory block per level to store the level's nodes and data example features. While this architecture reduces the amount of required logic resources, it is not flexible enough to build large tree forests, and not easily programmable with trees of varying depths. Fareena et al. [20] used a similar pipelined architecture but they instantiated multiple pipelines (8 pipelines) to parallelize the processing of incoming data examples. Kulaga et al. [11] uses Vivado HLS to generate accelerators for decision tree ensembles. They tuned Vivado HLS parameters to pipeline the computations of tree levels and parallelize the processing of multiple trees, achieving competitive throughput to ARM and Intel cores. However, their results where limited to 64 trees of depth 10. The resulting design is fixed for this tree ensemble

size. Different tree ensemble size requires a different tuning of Vivado HLS and generating a new design.

The work of Essen et al. [8] is the closest to our design. They proposed an FPGA architecture to accelerate random forests using pipelined processing of tree levels in a similar architecture to Struharik. The proposed architecture consists of multiple pipelines each can accommodate one or more trees. To mitigate the inefficiency of using BRAMs for early tree levels, they used only BRAMs for levels with 32 nodes or more, and flip flops for levels with less than 32 nodes. However, this design still does not manage on chip memory efficiently. For large forests with hundreds of trees, it will suffer from large logic resource usage and cannot scale to many parallel pipelines. Essen et al. use multiple FPGA devices demonstrating an order of magnitude speedup over multicore CPU solution for a random forest of 234 trees. In our design we pack all tree levels in a single processing element to efficiently use on-chip memory, which allow us to provide highly parallel architecture processing up to 512 trees in parallel.

Yun et al. [17] designed a dynamically programmable architecture supporting different decision trees of a similar depth. Tracy et al. [22] explored the use the Automata Processor (AP) from Micron to parallelize the inference of random forests. They map every path in the tree from the root to a leaf node on a chain automaton. In their experiments, they demonstrate that as the tree number grows and trees become deeper, the AP performance matches the CPU based solution similar to our conclusions in section VI-C. However, their AP solution did not scale beyond tree ensembles of 20 trees deeper than 12 levels. Oberg et al. [13] presented an FPGA classification system for Microsoft Kinect depth image pixels. The forest considered in their design consist of 3 20-level deep trees. Since the used trees are very large and cannot fit on the FPGA memory, they stored the forest trees in off-chip DDR and used the on-chip memory to store the relatively small depth image (19000 pixels). To minimize off-chip memory traffic while traversing the trees, they devised a special memory layout for storing trees and used sorting techniques when processing pixels such that pixels using the same tree nodes are processed in parallel. Their design reached up to a 30x speedup compared to a single core Intel's Atom processor. Their approach is interesting for data sets where ensembles of very deep decision trees provide the best results.

The different architectures proposed in previous research are fixed for a number of trees with specific depth. They lack flexibility and programmability at runtime. In our work, our main objective is to provide higher flexibility and a programmable decision-tree based classifier with no restrictions on the number and size of the tree ensemble through runtime management of resources and coordinated CPU-FPGA processing.

VIII. Conclusion

In this paper we have presented a scalable and flexible inference system for decision tree ensembles on CPU-FPGA platforms. The design employs two fundamental features of FPGA devices. The high bandwidth of distributed memory blocks (i.e., BRAMs) and the distributed logic resource, configured into many parallel processing elements around BRAMs. These two features provide an advantage over CPU based solutions for applications with frequent random memory accesses. We plan in the future to integrate the developed inference engine as a machine learning database operator similar to our previous effort in growing a library of database accelerated operators [21], [15].

Acknowledgment

We would like to thank Intel for the generous donation of the hardware platform as part of the Hardware Accelerator Research Program.

References

[1] F. Amato, M. Barbareschi, V. Casola, and A. Mazzeo. An FPGA-based smart classifier for decision support systems. In *IDC'14*.

[2] P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5(4308), July 2014.

[3] M. Barbareschi, S. Del Prete, F. Gargiulo, A. Mazzeo, and C. Sansone. Decision tree-based multiple classifier systems: An FPGA perspective. In *MCS'15*.

[4] R. Bekkerman. The present and the future of the KDD cup competition, 2015. http://www.kdnuggets.com/2015/08/kdd-cup-present-future.html.

[5] L. Breiman. Random forests. *Machine Learning*, 45(1), Oct. 2001.

[6] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *KDD'16*.

[7] T. G. Dietterich. Ensemble methods in machine learning. In *MSC'00*.

[8] B. V. Essen, C. Macaraeg, M. Gokhale, and R. Prenger. Accelerating a random forest classifier: Multi-core, GP-GPU, or FPGA? In *FCCM'12*.

[9] J. H. Friedman and J. J. Meulman. Multiple additive regression trees with application in epidemiology. *Statistics in Medicine*, 22(9), Apr. 2003.

[10] Kaggle. MLSP 2014 schizophrenia classification challenge, 2014. https://www.kaggle.com/c/mlsp-2014-mri.

[11] R. Kulaga and M. Gorgon. FPGA implementation of decision trees and tree ensembles for character recognition in VIVADO HLS. *Image Processing and Communication*, 19(2), Mar. 2015.

[12] LHCb Collaboration. Search for the lepton flavour violating decay $\tau^- \to \mu^-\mu^+\mu^-$. *High Energy Physics*, 2015(121), Feb. 2015.

[13] J. Oberg, K. Eguro, and R. Bittner. Random decision tree body part recognition using FPGAs. In *FPL'12*.

[14] N. Oliver, R. Sharma, S. Chang, et al. A reconfigurable computing system based on a cache-coherent fabric. In *ReConFig'11*.

[15] M. Owaida, D. Sidler, K. Kara, and G. Alonso. Centaur: A framework for hybrid cpu-fpga databases. In *FCCM'17*.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2011.

[17] Y. R. Qu and V. K. Prasanna. Scalable and dynamically updatable lookup engine for decision-trees on FPGA. In *HPEC'14*.

[18] S. R. Decision tree ensemble hardware accelerators for embedded applications. In *SISY'15*.

[19] T. Salimans. HiggsML, 2014. https://github.com/TimSalimans/HiggsML.

[20] F. Saqib, A. Dutta, and J. Plusquellic. Pipelined decision tree classification accelerator implementation in FPGA (DT-CAIF). *IEEE Transactions on Computers*, 64(1), Jan. 2015.

[21] D. Sidler, Z. István, M. Owaida, and G. Alonso. Accelerating pattern matching queries in hybrid cpu-fpga architectures. In *SIGMOD'17*.

[22] T. Tracy, Y. Fu, I. Roy, E. Jonas, and P. Glendenning. Towards machine learning on the automata processor. In *ISC'16*.