Hardware Acceleration of Decision Tree Learning Algorithm

Asim Zoulkarni
Department of Electrical and Computer Engineering
NTUA
Athens, Greece
el13068@mail.ntua.gr

Christoforos Kachris
Institute of Communications and Computer Systems
NTUA
Athens, Greece
kachris@microlab.ntua.gr

Dimitrios Soudris
Department of Electrical and Computer Engineering
NTUA
Athens, Greece
dsoudris@microlab.ntua.gr

***Abstract*—Decision Tree Classification variants are among the most popular machine learning algorithms and have been applied in various fields with success. Their versatility and popularity along with the ease to use make it imperative that solutions be found regarding its performance optimization problem, hence in this paper we tackle this issue by applying methods to optimize a Decision Tree Learning implementation (version C4.5), that will be executed in a heterogeneous computing system involving FPGA along with CPU, making use of the tools offered by the Software-Defined System On Chip (SDSoC) development platform. Initially, a profiling of the code is done, after which the computationally intensive part of the algorithm is determined, that is found to be the decision tree training part and within that part specifically, the Information Gain computations. Then the kernel has been developed as a hardware accelerated function that assumes the latter computations. The presented performance of the kernel was evaluated on a Zed platform integrated with Xilinx Zynq-7000 SoC in a FPGA-based heterogeneous system and it was shown that the accelerator can yield up to 2.48x kernel speedup in a single decision tree's training part, compared to an embedded ARM processor based implementation.**

***Index Terms*—machine learning, data mining, decision tree, C4.5, SDSoC, hardware acceleration**

I. Introduction

Machine learning algorithms and consequently data mining approaches such as Decision tree learning have been increasingly gaining popularity among fields such as medical diagnoses, risk evaluation of credit card application approval, consumers' behaviors, etc. Such a broad spectrum of fields where decision tree learning and classification are applicable along with the growth of available information to a massive extent in the era of big data make it desired to seek ways of optimizing a computationally wise heavy algorithm that can make use of this information and which alone is widely utilized and forms a single fraction of ensemble learning methods such as the random forests, etc. There the benefits of an optimization in the former algorithm would be even more highlighted as the speedup is expected to be boosted by a significant degree.

Hardware engineers, from their perspective, make an effort to seek ways of making hardware acceleration methods accessible for a software designs of the algorithm that are abstracted from the hardware they are going to run on, only to the highest degree it is still possible. This software approach of hardware acceleration involving Field Programmable Gate Arrays (FPGA)-based heterogeneous system is addressed in the presented work.

Such approaches are intended in order to improve the energy efficiency of broadly used machine learning algorithms' implementations. Modern day sophisticated processors that would be alternatively used for that purpose, though fast in high clock frequencies yield high energy consumption, escalating while more and more cores are utilized in parallel implementations. Recently, FPGAs are increasingly getting popular since they offer the option of highly parallelized implementations, where operations and code patterns are executed simultaneously in versatile computational units suiting only the necessary functions utilizing only as many elements such as logic gates required by the specific computations, in contrast to all alternative hardware, that have a fixed architecture requiring the integration of all of its components to operate.

The main contributions of the paper include the presentation of a novel hardware architecture for acceleration of decision tree learning algorithms, secondly, a performance evaluation and comparison with Central Processing Unit (CPU) implementations and finally the possibility for up to approximately $2.5\times$ speedup and lower energy consumption compared to typical CPU implementations.

II. Background

A. Classification Model

Machine learning algorithms such as decision tree learning are applied as models to numerous cases that require classification and prediction. Specifically a tree structure is formed in order to represent the input data in a more compact way of if-then rules, the so called classification rules. Every node in the tree defines a control condition of a given attribute of the instances and every branch that departs from that node corresponds to a different discrete value of the specific attribute. In every node, the case's value for the node's attribute is examined and the branch to follow is hence determined.

The procedure during which the tree is build is called decision tree training. According to the C4.5 decision tree learning algorithm and similarly with the Iterative Dichotomiser 3 (ID3)

algorithm, a tree is built using the metric of information entropy. It works in a top-down manner recursively building the tree by finding the feature of the input data that will be used as splitting criteria at each point. An extension provided by the C4.5 algorithm is the capability to deal with continuous features, discretizing them in two categories below or equal and greater than a threshold.

B. Training Data Discretization

Depending on the training data set and whether it contains continuous features, a prepossessing for the binary discretization of the latter is applied; subsequently, the feature is treated as categorical with each instance evaluated as exceeding or subceeding an appropriately elected value. The threshold that divides the continuous feature (assuming it to be a real number) into two classes of the lower and higher values is calculated as follows. Let $S = \{a_1, a_2, ..., a_n\}$ be the training values of a continuous attribute in a sorted order and m the number of classes $\{C_1, C_2, ..., C_m\}$ of the data set. Out of the $n-1$ possible splits evaluated, the one that minimizes the information metric is selected. Using the sorted order of the values, the maximum information conveyed is found in a single pass. The information metric is calculated for each split a_i as:

$$G(S) = -p_h \cdot E(S_h) - p_l \cdot E(S_l)$$

where

$$p_{h,l} = \frac{|S_{>a_i, \leqslant a_i}|}{|S|}$$

are the fractions of the values greater and lower than or equal to the split, respectively and

$$E(S) = -\sum_{i=1}^{m} p_i \cdot log_2(p_i)$$

is the information entropy of a set, where

$$p_i = \frac{freq(C_i, S)}{|S|}$$

is the fraction of the values within the set that correspond to the class i.

C. Decision Tree Training

Having dealt with with continuous features, the decision tree building part treats attributes in a universal way for both continuous and categorical values. For a given set of training data there are the three following cases evaluated by the algorithm [7].

- If the instances provided by the data set belong to a single class, the tree is a leaf corresponding to that class.
- If the data set contains no instances, the tree is a leaf corresponding to a default class.
- If the data set contains instances belonging to various classes, then the feature A with attribute values $\{a_1, a_2, ..., a_n\}$ that maximizes the information gain calculated for each feature as

$$G(S, A) = E(S) - \sum_{i=1}^{n} p_i \cdot E(S_i) \quad (1)$$

where $E(S)$ is the entropy of the examined feature and

$$p_i = \frac{|S_i|}{|S|} \quad (2)$$

where S_i is the new data set that would be yielded if the split were to occur in the i-th attribute value. The algorithm proceeds to create every tree node recursively using subsets of the data.

III. Software Acceleration

The FPGA offers the capability of in-parallel executing functions that could be run by the CPU, resulting in the benefits obtained by the parallel implementation as a circuit. The SDSoC development environment provides the opportunity to design code in higher level languages such as C/C++ and automatically produce the vhdl description that implements the indicated function. In this project, we focus on identifying a function that is highly computationally intensive, separating it from the rest of the code and let it be compiled as a circuit. This code pattern yields a number of restrictions for both the code selected to be hardware accelerated as well as the rest of the code sharing structures and other variables with the target function. The profiling of the algorithm execution is needed in advance to help us identify that function.

A. Algorithm Profiling

A custom implementation of the C4.5 decision tree algorithm in C++ is designed, developed and then profiled as to what percentage of the time is consumed by each subprocedure in order to help us identify the most highly computationally intensive one. The results are presented in the diagram below.

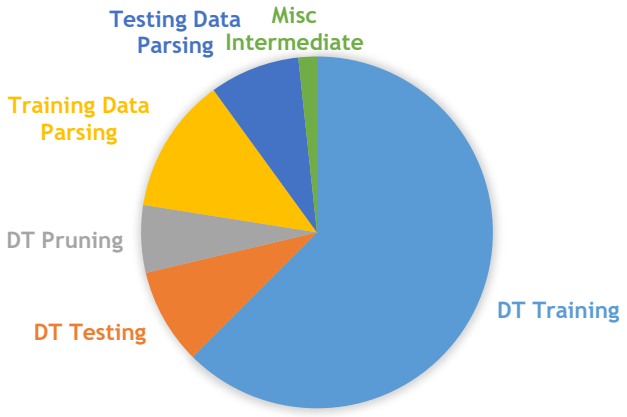

Fig. 1. Profiling of Decision Tree (DT) Classifier for Adult [10] data set.

In the testing part, a single path along the decision tree is traversed for each of the testing instances. The training part of the algorithm is by far the most time consuming compared to with the testing part and although the classification part

has the ability to be parallelizable by any desired factor regarding the independent testing of the input instances, in this work we focus on accelerating the most time consuming part which is the training one. The building of the decision tree involves a series of highly time consuming procedures some of them being mainly memory intensive requiring high memory usage and are the hardware acceleration of such functions is deemed inefficient. On the other hand, the part regarding the information gain computation is time consuming enough to be worth being accelerated separately, requiring memory utilization capable of being provided by the FPGA resources, unlike the rest of the training part that requires processing the data set and its subsets simultaneously and in a recursive manner.

Specifically, from the profiling carried out for a C++ implementation of the C4.5 algorithm, 62.5% of the total time consumed for the whole procedure, including the time required to load and process the raw data from the input files, and the whole training and testing parts of the algorithm, was required by the training part. The testing part consumed only the 8.3% of the total time while the time required to parse these data consumed 8.8% of the total time and when the ratio of the instances for the two parts of the algorithm (training and testing) respectively was 2:1. Since the scaling of the testing data yields commensurate time increase to the respective procedure, due to the $O(1)$ complexity of a single instance processing for a constant number of features, the new percentages of the time consumed for each part, if the ratio training to testing data were to become 1:1, by increasing the amount of the testing data to match that of the training data, would be

$$p_{train} = 62.5\% \times \frac{100}{117.1} = 53.4\%$$

and

$$p_{test} = 2 \times 8.3\% \times \frac{100}{117.1} = 14.2\%$$

and these percentages justify our decision to attempt an optimization on the decision tree training part, since the amount of time spent on building a decision tree with a specific amount of instances takes up to 3.76 times more than it would for the same amount of instances of the testing part, which, including an inner sight of the implementations of these parts as mentioned earlier, is deemed computationally not intensive. Having said that, we might now want to focus on what nested functions of the training part would be time-wise worth of being implemented as hardware functions that would run on FPGA. Of the 62.5% of the time that is consumed for the building of the tree, its two portions of 37.5% and 25%, are required by the processes of a) forming the reduced data set that provides the subset of instances that the next step on the recursive building of the tree will use, and b) the evaluation of the splitting feature at each step of the building, respectively.

B. Hardware Function Selection

The first sub-part of the training is highly memory intensive, and though parallelizable, this characteristic lies upon accessing random-access memory (RAM) in parallel manner, of data that very easily exceed the amount of dynamic-RAM (DRAM) provided by an FPGA-based heterogeneous system, especially when these instances of data-subsets are required to co-exist in the RAM while the tree is built recursively, in a top-down manner. On the other hand, the part regarding the evaluation of the splitting feature at each step of the recursion, requires the processing of the whole data included in each subset of training instances. During this part, the information gain is calculated for each feature by means of evaluating the frequencies of the attributes and applying of the metric each time, and the feature maximizing the information gain is its output.

The sub-part of the splitting feature evaluation is implemented as hardware function through special re-design of the code with adjustments regarding the structures used to describe the data processed by the FPGA as well as the rest of the algorithm that refers to the very same data.

IV. Accelerator Design

The next step in our work is to implement the selected function in hardware. For this purpose, we make use of the SDSoC development environment that allows the designer to control the C/C++ synthesis through optimization directives (`#pragma`) [8].

First, the initially two-dimensional matrix containing the byte-encoded attributes, is re-designed as a serialized array that is stored in a physically contiguous memory to achieve the optimal kernel-processor streaming strategy. Additional characteristics inherent to the splitting feature evaluation function and the data structures involved in it are taken into account as the hardware design using high-level directives is implemented.

Second, the fact that the data are not accessed in a sequential manner with regards to the serialized array, leads us design a different way with which the data will be streamed through the Advanced Extensible Interface (AXI) interconnect core. Every attribute column is processed once in order to have the attributes' frequencies counted and stored in block-RAM (BRAM) for the proceeding calculations, while at the same time the class column of the data which corresponds to the class that each instance targets is traversed as many times as the number of features is. For this purpose an extra reference on the last column of the training data streamed along with each column of attributes. This way sequential access of each column is achieved with the kernel accessing different features' values as different references that are streamed simultaneously. This optimizes the throughput of the communication required each time the hardware function is utilized and the data selected by the host are streamed for further processing.

Furthermore, a number of pragma directives were added to specify an implementation of the maximum information gain evaluation as well as the metric's values calculations. To proceed in the further explanation of the accelerator architecture regarding the selected function, we shall explain in more detail the work carried out by that function.

For each feature column the frequencies of the present attributes with respect to the corresponding classes that the training instance targets are calculated and stored in a two-dimensional frequency matrix as demonstrated below.

TABLE I
INSTANCE OF ATTRIBUTE'S VALUES FREQUENCIES WITH RESPECT TO CORRESPONDING CLASSES

working class attributes[a]	**Income Classes**	
	$\leq 50K$	$> 50K$
private	12947	3569
state-gov	683	256
federal-gov	449	276
local-gov	1108	445
⋮	⋮	⋮
self-emp-inc	363	454

[a]Statistics obtained from Adult [10] data set.

Such a frequency matrix is formed by a single column of attributes corresponding to just one feature of the training data and one is formed for every external loop of the maximum information gain evaluation. A single iteration of the respective feature column whose data are streamed through the AXI interface and are processed sequentially is enough for its items' frequencies to be stored in the dual-port BRAM in the kernel. The dual port BRAM is selected for it offers the opportunity to evaluate the attribute's frequencies with respect to the corresponding classes of two features at a time (hence the two attribute arguments in Fig.2; the offset is the address difference of the two references) making use of the independent nature of the processing on the FPGA. This is accomplished by specifying the unroll pragma directive with a factor of two to the loop regarding the iterations over the number of features.

Additionally, an attempt to optimize the loop regarding the iteration over the input data that measures the aforementioned frequencies is made. This is achieved by the use of the pipeline pragma directive, the effect of which essentially allows concurrent operation executions of the loop's body by means of deploying loop iterations in overlapping time intervals, yielding a respective drop to the parameter of the loop's initiation interval to the minimum possible for the specific loop and which is determined by SDSoC during the synthesis.

The next step of the processing carried out at the kernel regards the processing of the recently measured frequencies and the application of the information gain formula with the calculation of the respective cumulative sums of entropy portions. This part of the algorithm is implemented as a series of double nested loops of which the innermost have the pipeline applied to their iterations over the number of classes of the data set and the outermost have it applied to the iterations over the number of attribute values per examined feature. The latter is furthermore unrolled by a factor that would statistically allow the throughput of the frequencies matrix reads to be maximized as the accesses of its elements is done by the independent unrolled loop parts that can refer to the different blocks of the array simultaneously, for which the respective block partition pragma is specified for the linearized matrix. At the same time, the last column of the training data that corresponds to the target class of the training instances has been streamed and stored in dual-port BRAM in the FPGA, that allows the unrolled by a factor of two loops that iterate over equal amounts of the features' attributes (split in half) and enables the access of the classes values at the same time while guaranteeing the independence of the execution of these two unrolled loop parts.

As far as the cumulative sum calculation is concerned, the application of the formula (1) that calculates the information gain of the column/feature is utilized and for which a number of adjustments and directives are applied yielding the respective resource allocation where the additions, multiplications and divisions of floating point numbers are going to take place. The additions for the purposes of the cumulation of the sum of the entropy are implemented in floating-point adders that use only DSP48s [9] primitives, whereas the multiplications of the fractions (2) with their respective logarithm portions are calculated in floating-point multiplier using similar primitives, while the divisions that are required for the fractions (2) calculations are carried out at floating-point divider.

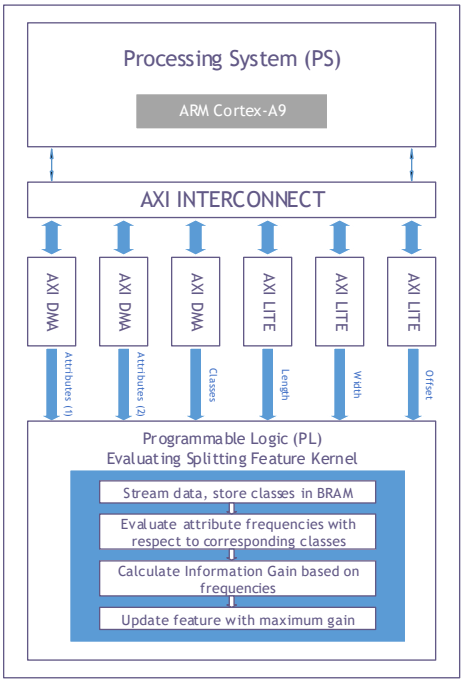

Fig. 2. Above is the simplified block design of the heterogeneous architecture involving FPGA and CPU, featuring therebetween communication.

A simplified block design is shown in Fig. 1 that illustrates the system architecture design. It includes a processing system that carries out the rest of the calculations assigned to the FPGA implemented kernel, which performs the task of the splitting feature evaluation, through the calculation of the information gain metric. Data stored in arrays are transferred through AXI direct memory access (DMA) while the rest

integer type variables are transferred through the AXI4-Lite interface.

V. Performance Evaluation

The case studies considered for the performance evaluation of the training and testing parts of the decision tree classifier were the Adult and Census-Income, multivariate data sets provided by the UCI machine learning repository [10]; features of both data sets included both categorical and integer type attributes with the latter including floating-point type attributes as well. A quantitative profiling of the data sets shows that the former includes 48842 total instances of 14 features, 8 of which are of categorical attributes and 6 are of continuous values upon which the discretization prepossessing was applied, as was done for the 7 continuous features of a total of 40 features, in the Census-Income database of 199523 training instances and 99762 testing ones, with the rest 33 being categorical value features.

One of the major factors that could hinder the speedup and the overall gain of a hardware function implementation on FPGA is the communication overhead caused by the need to transfer the data between the Processing System (PS) and Programmable Logic (PL). Such being the case, in this work we had to compare and deduce the most appropriate means of achieving the PS-PL communication. As mentioned earlier, the classes' attributes are accessed in a periodic manner which violates the sequential access pattern characterizing the rest of the attributes. That led us to evaluate the use of zero-copy for the data streamed, either partially for the classes, or for the whole data set streamed, an overall implementation that was observed to yield higher communication overhead compared to the implementation that resulted in the final kernel speedup. Moreover, it should be noticed that the hybrid implementation imposals of more pre-allocated structures preference over the classic dynamic ones that are typically used, further boost the system speedup of the training part as a whole, even through functions running out of FPGA. The measurements presented were observed for the Adult and Census-Income data sets at data motion and operating clock frequencies both being at 142.86 MHz, the resource utilization for which by the hardware function is shown on table II (DSP: digital signal processors, BRAM: block random-access memory, LUT: look-up tables, FF: flip-flops). In order to achieve the performance of the implementation presented a number of parameters were subjects to fine tuning, such as those of the array partition factors, loop-unrolling factors and resources where operations such as additions, multiplications and divisions were carried out.

The C/C++ code of the algorithm along with the software-defined system (SDS) and high-level synthesis (HLS) directives that result in the presented architecture, was compiled into an executable file that along with the bit-stream generated by the SDSoC development platform required for the FPGA configuration, was able to run on the ZedBoard (Zynq-7000 All Programmable SoCs) system yielding the following results

TABLE II
Resource Utilization for Splitting Feature Evaluation

Resource Name	**Hardware Function Resources**[a]		
	Used	***Total***	**% *Utilization***
DSP	48	220	21.82
BRAM	75	140	53.57
LUT	35807	53200	67.31
FF	47608	106400	44.74

[a]Resources regard ZedBoard (Zynq-7000 SoC).

in time and accuracy, for both mentioned sets respectively, as shown in table III:

TABLE III
Time and Accuracy Comparison

Data set Name	**Implementation**			
	Software	***Hybrid***	***Speed-up***	***Accuracy***
Adult	2.33s	0.94s	2.48	82.54%
Census-Income	34.27s	15.5s	2.21	94.26%

VI. Conclusion

In this work, an implementation that featured hardware acceleration of a decision tree classifier was presented, where functions that posed bottlenecks for the performance optimization were let to run on a common processor (Dual-core ARM Cortex-A9) and functions that their hardware implementation was deemed beneficial, had their architecture-involving FPGA-designed in a novel way.

While attempting to accelerate the classification part of the algorithm might as well have given impressive results, two main reasons focused our interest on the training part. The intrinsic simplicity of an implementation regarding the single decision tree classifying part along with the bulk of research carried out already regarding classification using decision tree ensembles, contributed to our decision to attempt an optimization on the learning procedure instead.

Acceleration using state-of-the-art hardware is increasingly getting popular and is sought as a resort for performance optimization for various machine learning algorithms among which decision trees and their ensembles have great potential.

VII. Related Work

A lot of research has been carried out on how either decision tree classifiers or their ensembles such as random forests, other ensemble methods such as the so-called Bagging (Bootstrap Aggregation) or Boosting are to be implemented in some sort of computing system involving FPGAs. The fact that a decision tree constitutes a unit of the structures built by any of these methods indicates the importance of studying the optimization of a decision tree classifier. Any possible benefit obtained is transferable to an implementation involving multiple trees possibly over a cluster of FPGA computing nodes, where scaling the amount of nodes utilized causes increased performance gain; such an approach is made by [1], where software driven FPGA implementation of a decision

tree ensemble classifier was proposed and the scalability over tree structures requiring more than the FPGA provided memory was examined. The classification part's performance was optimized in an implementation that offered the capability to deal with any size or number of decision trees determining whether FPGA alone execution or in combination with CPU is required to handle the size of the structure.

In [2] a different approach on performance optimization was presented, where decision tree ensembles of a determined amount of nodes were stored in an appropriate amount of FPGAs required for them to fit in the on-chip memory and communication protocols regrading their optimal inference were examined. Ensembles of shallow decision trees up to depth of 9 fit into a single FPGA, but trees of increased depth would require doubling of FPGA nodes utilized per depth increase of 1. Such are the trade-offs when attempting to optimize the decision tree ensemble classifier that requires the storage of the classifying structure into FPGA. In our work instead, FPGA is used to perform the computationally intensive training part without the need for the whole data set or decision tree to even fit into the usually restricted on-chip memory.

Another work as shown in [3], proposed an acceleration method of a decision tree classifier by means of exploiting the comparative ease with which FPGA performs calculations with integers over floating-point values. In their proposal the feature label should require the minimum amount of bits to enumerate all labels and the threshold regarding the continuous values shall be stored in 32 bit floating-point variable as any less bits would compromise predictive accuracy. In our work, on the other hand, the optimal way to treat floating-point features would be to evaluate the threshold of every continuous attribute as a C/C++ floating point type at parse time by the CPU and store the testing data as byte-encoded values. This way, having the FPGA to processes any such floating-point types is circumvented, saving data streaming time along with floating-point comparison time, as a common processor might do this very job faster; byte is deemed the best data type that is the most flexible for both CPU and FPGA to deal with at the same time, avoiding redundant type conversions.

Among the several aspects as to how studies have sought to tackle the challenge of optimizing decision tree learning in hardware, comes the work of [4]. In it, the Hoeffding tree algorithm is studied and its hardware implementation is optimized through the fine tuning of parameters such as the number of quantiles required for the attribute learning, which despite having an impact on the resulting accuracy, helps achieve a significant speedup.

An additional proposal could be found at [5], where four different architectures were designed as an answer to the demand for decision tree classifying performance optimization, all of which were shown to out-speed equivalent software-only implementations. Each unit of the ensemble was implemented in different module in two possible ways; pipelined and sequential decision tree evaluation, for which modules speedups varied from 9.56 to 5188.74 times.

The approach that was closer to the current one was the one at [6]. Similarly a hybrid architecture was proposed, where the Gini Score computational weight was let for the FPGA to process, whereas the rest of the decision tree classifying algorithm was running on CPU, yielding again a speedup compared to a software-only implementation. The main differences between it and our work are on a theoretical level the metrics used for the decision tree building where we used the information gain instead and on a more technical level, the implementation and architecture design. The results were presented for an implementation that treated only binary classes, while indicating the extension to support more. Additionally, the binary nature of the classes was exploited to expand the Gini Score formula into a predetermined amount of sums and divisions, whereas in this work a more flexible approach is made. This by allowing the evaluation of the formula to be carried out as a for loop, making use of the high level synthesis capability to transform C-style loops, as well as the rest of the hardware function code, into hardware seamlessly, without loss in accuracy compared to when the implementation runs on CPU only.

ACKNOWLEDGMENT

This project has received funding from the Hellenic Foundation for Research and Innovation (HFRI) and the Genal Secretariat for Research and Technology (GSRT) under grant agreement no 2212: CloudAccel Hardware Acceleration of Machine Learning Applications in the Cloud. This project has also received support from Xilinx by means of providing the hardware and software tools used to develop end evaluate the results of this work.

REFERENCES

[1] Owaida, M., Zhang, H., Zhang, C., Alonso, G. (2017). "Scalable inference of decision tree ensembles: Flexible design for CPU-FPGA platforms". 2017 27th International Conference on Field Programmable Logic and Applications.

[2] Owaida, M., Alonso, G. (2018). "Application Partitioning on FPGA Clusters: Inference over Decision Tree Ensembles". 2018 28th International Conference on Field Programmable Logic and Applications (FPL).

[3] Zhao, S., Sun, Y., Chen, S. (2018). "A Discretization Method for Floating-Point Number in FPGA-based Decision Tree Accelerator", 2018 IEEE 4th International Conference on Computer and Communications (ICCC).

[4] Lin, Z., Sinha, S., Zhang, W. (2019). "Towards Efficient and Scalable Acceleration of Online Decision Tree Learning on FPGA". 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).

[5] Struharik, R. (2015). "Decision tree ensemble hardware accelerators for embedded applications". 2015 IEEE 13th International Symposium on Intelligent Systems and Informatics (SISY).

[6] Narayanan, R., Honbo, D., Memik, G., Choudhary, A., Zambreno, J. (2007). "An FPGA Implementation of Decision Tree Classification". 2007 Design, Automation & Test in Europe Conference & Exhibition.

[7] J. Ross Quinlan, "C4.5. Programs for Machine Learning", Elsevier Inc. (1993), pp. 24-31.

[8] Xilinx Inc., "SDx Pragma Reference Guide UG1253", January 24, 2019, pp. 33-62.

[9] Xilinx Inc., "Vivado Design Suite User Guide High-Level Synthesis UG902", December 20, 2018, pp. 167-171.

[10] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository", 2017. [Online]. Available: http://archive.ics.uci.edu/ml