ISSN: 1533 - 9211
DOI: 10.5281/zenodo.6879643

AN FPGA IMPLEMENTATION OF A TWO MEANS DECISION TREE

P. Suneel Kumar
Professor, Department of Electronics and Communication Engineering, Sridevi Women's Engineering College, Hyderabad, India, psunilkumar.ece@gmail.com

T. Vasanthi
U.G Student, Department of Electronics and Communication Engineering, Sridevi Women's Engineering college, Hyderabad, India, vasanthiteki00@gmail.com

R. Akhila
U.G Student, Department of Electronics and Communication Engineering, Sridevi Women's Engineering college, Hyderabad, India, akhilaresoju123@gmail.com

A. Sahithi
U.G Student, Department of Electronics and Communication Engineering, Sridevi Women's Engineering college, Hyderabad, India, arravothusahithi@gmail.com

Abstract: Decision Trees (DTs) are exuberantly used in machine learning (ML) because of their quick implementation and decipherability. As DT training is laborious, on this short, we suggested a hardware training accelerator to fasten the training procedure. The proposed training accelerator is carried out on the field programmable gate array (FPGA) having the most operating frequency of 62 MHz. The proposed structure uses a combination of parallel execution for training time deduction and pipelined execution to reduce resource intake. For a given design, the proposed application is found to be at the least 14 times faster than the C-based software programming. Besides, the proposed structure makes use of the single RESET signal to re-equip new set of data. This active training proves the hardware flexibility for any form of implementation.
Keywords: Decision tree (DT), field-programmable gate array (FPGA), machine learning (ML), training accelerator, two means DT (TMDT).

1. Introduction

Decision Tree's induction algorithms structurally split-up the input space in a up-down method until unique termination conditions are met. This hierarchical dividing is performed via routing data via a decision function hosted at split up nodes of DT. The tree ends at leaf nodes. These leaf nodes correlate to nearly pure areas of input area containing most of the instances with the identical label. The model's complexity and training time of DT depend upon selection features hosted at splitting nodes. The ranging level is from easy in-line rules (C4.5) [10] and implicit splits [1] to complex networks (neural trees) [12]. Most practical methods use a software program to train the DTs. Nevertheless, retraining parameters on those chips come to be hard for adapting to specific applications.

DOI: 10.5281/zenodo.6879643

Apart from that, various methods decide on nearby accessibility of data to embedded processors for portability and other motives [7]. Graphical processing units (GPUs) are frequently employed for machine learning (ML) methodologies regarding dense calculations, causing immense power consumption. Relatively, field- programmable gate arrays (FPGAs) are an appropriate choice to accelerate training in hardware.

Various works had been mentioned on hardware architectures [9][3][4] for accelerating the DT classification. Whilst the FPGA implementation proposed in [9] absorbs less power, it minimizes the throughput. Comparing the FPGA implementations proposed within the aid of Tong et al.[3], one implementation yields balanced classification in a short time, and the other provides an enhanced tree with less area. [4] Saqib et al. proposed a classification hardware that makes use of pipelined architecture to elevate the classification time.

The two means DT (TMDT) algorithm is just like the k-means algorithm. This brief proposes an adaptable FPGA based combined parallel and pipelined hardware architecture to stimulate the training of TMDT. Furthermore, the proposed implementation uses one FPGA, while the architecture in [6] uses four FPGAs. The most important contributions of this brief are indexed as follows.

•This adaptable hardware may be retrained on any binary dataset through using a reset signal, making it adaptable to adjustments in the training data.

•The node's execution is pipelined to maximize the throughput and minimize the resource consumption.

•The memory access and distance calculation is parallelized for all dimensions to scale down latency.

1.Method

2.1 Analysis of a two means decision tree

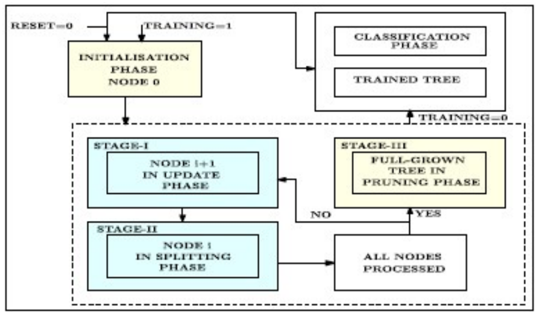

Fig 1 Training process flow of TMDT

The design passes through three states—standby, training, and classification. In the standby

SEYBOLD Report
ISSN: 1533 - 9211

DOI: 10.5281/zenodo.6879643

mode, the power supply to the FPGA board is turned on, and active high signal RESET is asserted 1, which initializes the hardware registers to 0. Once the RESET is set to 0, the TRAINING signal is set to 1, and the training is enabled. The completion of training sets the TRAINING signal to 0, which activates the classification. Whenever the RESET is set to 1, it erases all previous data and makes the hardware register contents 0. Thus, the classification starts once the entire training process is complete for the new data, and the process continues. This hardware can be retrained on the new data by switching the RESET signal between 0 and 1.

Hardware Modules

The hardware structure of the TMDT algorithm obtained by mapping the flow diagram is shown.

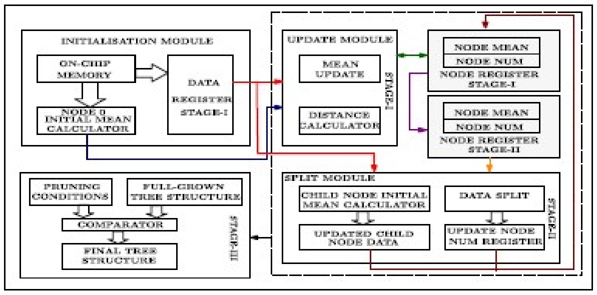

Fig 2 Hardware architecture for the TMDT

Initialization Module: In the initialization module, the data are first loaded into the data register from the on-chip memory to reduce the time wasted in accessing the memory. The initial mean of the root node is calculated in parallel, while the same dataset is loaded into the data register array from on-chip dynamic random access memory.

Distance Calculator: The architecture of the distance calculator module runs in parallel for all dimensions. For m-dimensional data, this architecture uses m subtractors and m multipliers to compute (μ[l]−x[l])2, where l = 0, . . ., (m−1), for m dimensions in parallel.

Update Module: The update module consists of mean update and distance calculator as the left and right means are updated based on the shortest distance from the instances of the data. After every iteration, the node mean register is accessed to store the updated mean calculated in pipeline stage-I.

Split Module: In the split module, the node data are split into two parts depending on the distance from left and right mean. In the split module, whenever a data vector is assigned to a particular child node, the node number of that data vector is updated in the Node Num register, and the vector is added to the initial mean of that child node. In this stage, all the split nodes of the full-grown tree obtained from the previous stage are tested for the split condition.

DOI: 10.5281/zenodo.6879643

Pipeline Execution

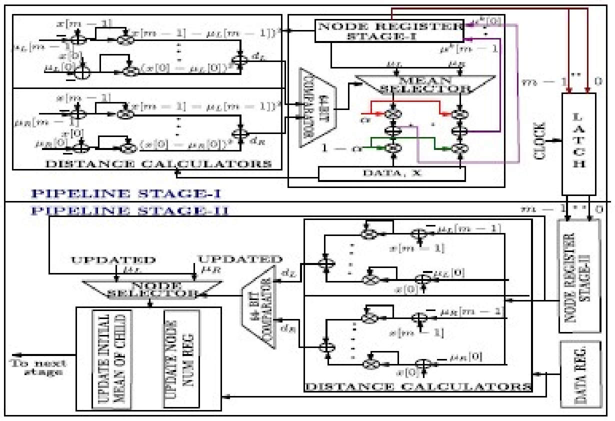

Fig 3 Pipeline stage-I and stage-II

The detailed architecture for stage-I and stage-II is demonstrated in above figure. This architecture uses two distance calculator units running in parallel to calculate the distance between the data and left and right means in stage-I. The two distances are then compared, and the mean with a smaller distance from data is selected by the mean selector. The selected mean is then updated and stored in the node register. In this way, all the data belonging to that node are processed in stage-I. Then, the final updated mean and node numbers from the node register of stage-I are sent to the node register of stage-II through a latch, and stage-I starts processing the next node. The clock period of the latch is set a little higher than the latency of stage-II so that the data are not overwritten. In stage-II, again, two distance calculators are used to calculate the distance between µL, µR, and xk stored in the updated node. The data are designated with the node number of child nodes whose mean is closest to the data. This node number is stored in the Node Num register. In this way, all data instances in the node are split into two parts and sent to one of the two child nodes. At the same time, the data going to the child node are summed up to calculate the initial mean of that child node that is stored in the update initial mean register. Once all the nodes have been processed in stage-I and stage-II, stage-III is executed.

Parallel Execution

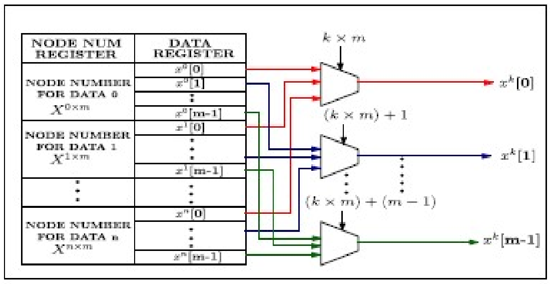

Fig 4 Hardware architecture for parallel execution

ISSN: 1533 - 9211
DOI: 10.5281/zenodo.6879643

The data register access is also parallelized to further speedup the execution, as shown in above figure. The register, named the Node Num register, stores the node number of all the data in order to identify the node to which the data belong. If the node number of kth data does not match with the current node number, then the node number of the (k +1)th data is checked; otherwise, the kth data are selected. To access all the m dimensions of data in parallel, m numbers of multiplexers are used. The current data location $k \times m$ is used as the select line to identify 0th dimension of kth data and vice versa. This allows to access m dimensional data in a single clock cycle, thus saving $(m - 1)$ clock cycles. Each m-dimensional data uses m memory locations of the data register array. Hence, for $n \times m$ information, $n \times m$ areas of the Node Num register and the information register were utilized to store the node number and data.

2.2 Analysis of Recent Research

Recent studies claims various methods were taken in consideration to develop techniques for machine learning allowing the user to feed the algorithm in a computer with immense data, helping it to analyse, recommend and decide based on given input data.

The fast architecture for filtering sensor data by using DT has been proposed in [9]. The serial architecture proposed in this brief implemented the DT and random forest classifier both on FPGA and Von-Neumann CPU and compared their relative performance. Although the FPGA implementation proposed in [9] consumes low power, it reduces the throughput.

Comparing the FPGA implementations proposed within the aid of Tong et al. [3], one implementation yields balanced classification in a less time, and the other provides an enhanced tree with less area.

The k-means architecture proposed in [5] implements k-means whose search area is pruned by the usage of a kd-tree data structure, leading to computation reduction up to 5 times from conventional implementation, but the latency in step with iteration is huge.

The classification and regression tree (CART) architecture reported in [6] implements training for the CART algorithm on the multi-FPGA system in a high-capacity (HC) server.

2.3 System analysis

Field-Programmable Gate Arrays (FPGA)

A field-programmable gate array (FPGA) is a block of programmable logic that can implement multi-level logic functions. FPGAs are most commonly used as separate commodity chips that can be programmed to implement large functions.

However, small blocks of FPGA logic can be useful components on-chip to allow the user of the chip to customize part of the chip's logical function. An FPGA block must implement both combinational logic functions and interconnect to be able to construct multi-level logic functions. There are several different technologies for programming FPGAs, but most logic processes are unlikely to implement anti-fuses or similar hard programming technologies, so we will concentrate on SRAM-programmed FPGAs.

XILINX ISE

This instrument can be utilized to make, execute, re-enact, and integrate Verilog outlines for usage on FPGA chips.

ISE: Integrated Software Environment

1. Environment for the improvement and trial of computerized systems configuration focused to FPGA or CPLD

2. Integrated gathering of apparatuses available through a GUI. Based on an intelligent combination motor (XST: Xilinx Synthesis Technology)

3. XST underpins diverse dialects:

   ❖ Verilog

   ❖ VHDL

4. Translate, guide, place and course

5. Bit stream era

The version used here is **XILINX ISE 13.2** for simulation and Synthesis. The programs are written in Verilog language.

3. Discussion of Results

The algorithm got implemented at the Intel Core i5 Processor for software contrast, which runs at 3.2 GHz. The Virtex Ultrascale+ XCVU9P-FSGD2104-three-E-ES1 FPGA board was used for hardware implementation of the proposed accelerator. The board makes use of 16-nm technology, and the design became implemented with a speed grade of −5 for optimum usage. The maximum operating frequency is restricted to 62 MHz due to the massive critical path of the order of 16 ns. This path is specifically because of block RAM (BRAM) access operation. The design's implementation was on Vivado 2017, and no high-level synthesis device was in practice.

Entity diagram for training accelerator two means decision tree is shown in the below figure.

DOI: 10.5281/zenodo.6879643

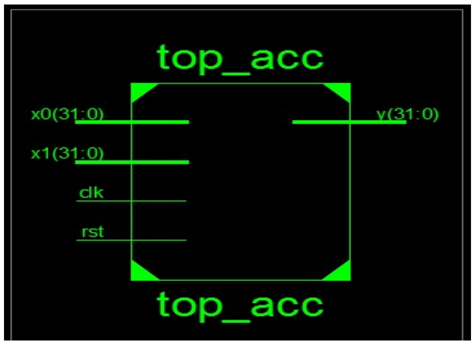

Fig 5 Entity diagram

RTL schematic for training accelerator two means decision tree is shown in below figure.

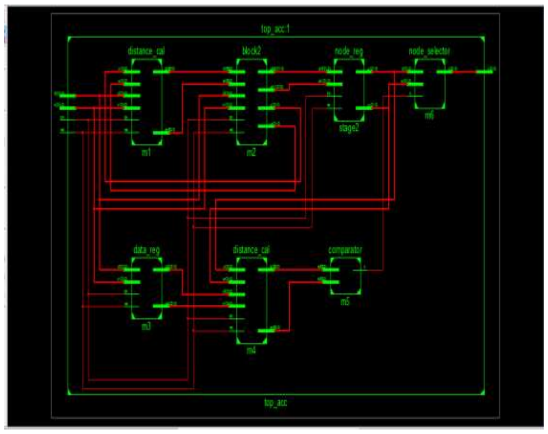

Fig 6 RTL schematic

Simulation results are obtained as follows

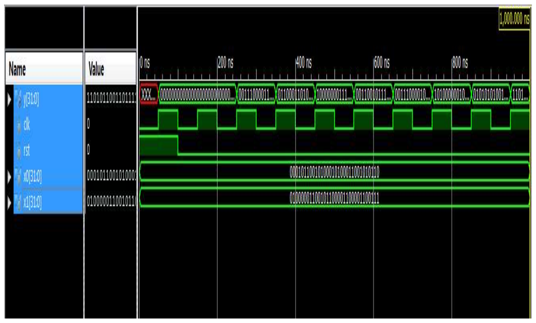

Fig 7 Simulation

4. Conclusion

In conclusion, we carried out a training accelerator for the TMDT algorithm. The training hardware was put to test using both 32-bit fixed factor and integer data. The training accelerator is observed to speedup the training process by at least 14 times faster. The training

DOI: 10.5281/zenodo.6879643

in FPGA was completed in 16.54 × 10−3, while software program training required at the least 224 × 10−3. The data were copied into this on-chip memory of the FPGA board from the PC prior to the start of the training process. This enabled independent training by connecting the FPGA board directly to the power supply once the design is implemented onboard. The design supports 32-bit fixed-point implementation up to two places of decimal.

In future works, this hardware can be reconfigured to enforce multiclass classification within the batch mode to enhance memory usage. The hardware proposed on this short implements training for 32-bit data. This board has 500 MB of on-chip memory, allowing a considerably large amount of 32-bit training datasets of almost 125 million data points. This large on chip-memory allows efficient latency data access.

References

[1] S. Ruggieri, “Efficient C4.5 [classification algorithm],” *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 2, pp. 438–444, Apr. 2002.

[2] A. Godbole, S. Bhat, and P. Guha, “Progressively balanced multi-class neural trees,” in *Proc. 24th Nat. Conf. Commun. (NCC)*, Feb. 2018, pp. 1–6.

[3] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, “High-performance FPGA-based CNN accelerator with block-floating-point arithmetic,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1874–1885, Aug. 2019.

[4] J. Wang, J. Lin, and Z. Wang, “Efficient hardware architectures for deep convolutional neural network,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 6, pp. 1941–1953, Jun. 2018.

[5] S. Buschjager and K. Morik, “Decision tree and random forest implementations for fast filtering of sensor data,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 1, pp. 209–222, Jan. 2018.

[6] D. Tong, Y. R. Qu, and V. K. Prasanna, “Accelerating decision tree based traffic classification on FPGA and multicore platforms,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3046–3059, Nov. 2017.

[7] F. Saqib, A. Dutta, J. Plusquellic, P. Ortiz, and M. S. Pattichis, “Pipelined decision tree classification accelerator implementation in FPGA (DT-CAIF),” *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 280–285, Jan. 2015.

[8] T. Saegusa and T. Maruyama, “An FPGA implementation of real-time K-means clustering for color images,” *J. Real-Time Image Process.*, vol. 2, no. 4, pp. 309–318, Dec. 2007.

DOI: 10.5281/zenodo.6879643

[9] G. Chrysos, P. Dagritzikos, I. Papaefstathiou, and A. Dollas, “HCCART: A parallel system implementation of data mining classification and regression tree (CART) algorithm on a multi-FPGA system,” *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, pp. 1–25, Jan. 2013.

[10] F. Winterstein, S. Bayliss, and G. A. Constantinides, “FPGA-based K-means clustering using tree-based data structures,” in *Proc. 23rd Int. Conf. Field Program. Log. Appl.*, Sep. 2013, pp. 1–6.

[11] S. Behnke and N. B. Karayiannis, “Competitive neural trees for pattern classification,” *IEEE Trans. Neural Network.*, vol. 9, no. 6, pp. 1352–1369, Nov. 1998.

[12] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, “High-performance FPGA-based CNN accelerator with block-floating-point arithmetic,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1874–1885, Aug. 2019.