International Journal of Research Studies in Science, Engineering and Technology
Volume 6, Issue 11, 2019, PP 74-86
ISSN 2349-476X
DOI: https://doi.org/10.22259/2349-476X.0611007

Gradient Boosting Decision Trees with FPGA Accelerator

Annadasu Saiprathima[1], Guntupalli Sai Sravani[1], Dr. Sunitha Kondepu[2]

[1]*CSE Department, Malineni Lakshmaiah Women"s Engineering College, Guntur, AP*

[2]*Associate Professor, CSE Department, Malineni Lakshmaiah Women"s Engineering College, Guntur, AP*

ABSTRACT

A decision tree is a commonly used machine learning approach. The strong Gradient Boosting ensemble method, which allows for steadily increasing accuracy at the cost of processing a huge number of decision trees, has recently expanded their popularity. We describe an accelerator in this research that optimises the execution of these trees while lowering energy usage. We tested it using a relevant case-study: pixel classification of hyperspectral pictures, which we implemented in an FPGA for embedded systems. In our tests with various images, our accelerator was able to process hyperspectral photos at the same rate as the hyperspectral sensors created them. On average, our design is twice as fast as a high-performance processor running optimised software. Our design is twice as quick and uses 72 times less energy on average. It is 30 times faster and uses 23 times less energy than an embedded processor.

Keywords: *GBDT; FPGA; energy efficiency; decision trees.*

INTRODUCTION

Decision trees are a simple and effective machine learning technique that have been proven to work in a variety of categorization tasks. Because energy efficiency is as crucial as precision in embedded systems, it is necessary to look for efficient algorithms that can be accelerated. As a result, decision trees are an excellent candidate for developing an FPGA accelerator.

For sophisticated tasks, a single decision tree is typically insufficient, however ensemble approaches allow multiple trees to be combined to solve complex issues. Gradient Boosting Decision Trees (GBDT) [1] is an ensemble method for improving accuracy by incrementally adding new trees in each iteration that improve the previous ones' results. Traditional GBDT implementations have poor scaling for large datasets or a high number of features, however new efficient implementations, such as XCGBoost [2], CatBoost [3], or LightGBM [4,] have recently solved this issue. LightGBM, for example, is an open-source GBDT-based framework that provides up to 20 times the speed of traditional GBDT. Due to its efficiency and accuracy, GBDTs are now considered one of the most powerful machine learning models thanks to LightGBM's support. For example, they've lately been used in a number of successful machine learning applications. They can be applied to a wide range of issues. For example, they have been successfully used to produce accurate COVID-19 evolution forecasts and identify factors that influence its transmission rate [6], to detect fraud from customer transactions [7], to estimate major air pollutants risks to human health [8] using satellite-based aerosol optical depth, and to classify GPS signal reception in order to improve its accuracy [9. Furthermore, a recent paper [10] examined various machine learning approaches for image processing in remote systems, with a focus on on-board processing, which necessitates both performance and low power consumption. The scientists discovered that GBDTs offer an intriguing trade-off between the utilisation of computing and hardware resources and the precision gained. Their accuracy scores were comparable to convolutional neural networks, the most accurate approach at the time, while requiring one order of magnitude less CPU processes. Furthermore, the majority of their inference operations are integer comparisons, which can be efficiently calculated even by low-power CPUs and quickly accelerated by FPGAs. As a result, they are a viable solution for embedded systems.

In this research, we describe a Gradient Boosting Decision Trees (GBDT) accelerator that can run GBDTs trained with LightGBM. We've set up a repository including the GBDT models we utilised as well as our source codes [11]. Our accelerator was created for embedded systems, which have limited hardware resources and power budgets. Hence Our accelerator's register-transfer level (RTL) design was written in VHDL, and we used it to demonstrate its potential in a low-cost FPGA evaluation board (ZedBoard) [12], which includes an FPGA for embedded systems, and we used it for the relevant case study analysed in [10]: pixel classification of hyperspectral images with the goal of processing the data at run-time. We analysed the execution time and power consumption of our accelerator and discovered that, even with a modest FPGA, our design can handle complex GBDT models. Our accelerator can process hyperspectral information at the same pace as the hyperspectral sensors create it in our case study, and the dynamic energy consumption is low.

RELATED WORK

FPGA acceleration of Decision Trees has been the subject of several earlier studies. The training processes are the centre of reference [13]. In our scenario, we'll suppose that training is done offline and that we'll concentrate on inference, which will be done online. A bespoke pipeline architecture was provided in reference [14], demonstrating the possibilities of an accelerator for decision trees. They do not support GBDT, however, and only use their methodology in basic case studies. To develop an FPGA accelerator, reference [15] offers using a high-level synthesis technique. They concentrate on Random Forest, an ensemble technique that calculates the average value of multiple trees trained with various input data to produce a more accurate and robust final result. We've made the decision to

Another distinction is that we created a custom register-transfer level (RTL) architecture rather than relying on high-level synthesis to produce the RTL design from C-code. High-level synthesis is appealing for portability and shortening the design cycle, but we can completely design the final architecture and investigate various sophisticated optimization options using our RTL design. Another study that examines the advantages of implementing Random Forest on FPGAs is reference [16]. They compare the performance of random forest classifiers on FPGAs, GP-GPUs, and multi-core CPUs. They conclude that FPGAs provide the best performance, but that due to the size of the forest, they do not scale. As previously stated, GBDT models use fewer trees to achieve the same results. FPGAs could be utilised to speed up the execution of decision trees in the Microsoft Kinect vision pipeline, which recognises human body parts and movements. They employ a high-performance FPGA and achieve excellent results for decision trees organised as a random forest. They do note, however, that their architecture is not suitable for low-power FPGAs due to its memory requirements. Reference [18] is a recent paper that describes an approach for producing compact and almost identical representations of the original input decision trees. by means of threshold compaction The fundamental idea is to group comparable thresholds together to reduce the number of individual thresholds required, and then save those values as hard-wired logic. The size of the trees can be lowered using this method. This technique is diametrically opposed to our approach and may be advantageous to our design because it reduces the size of the trees, making storage in embedded devices easier. Another recent paper [19] examines the advantages of FPGA acceleration for Gradient-boosted decision trees. They are evaluating the Amazon cloud services in this situation, which include access to high-performance FPGAs that can be exploited via high-level interfaces. As a result, this study complements ours because it focuses on high-performance cloud servers, whereas we focus on data centres. In conclusion, earlier research has shown that FPGAs may be used to execute decision trees. The majority of these projects focus on Random Forest, and they either construct small systems or require high-performance FPGAs. As a result, in order to apply these technologies in embedded systems, their scalability must be improved. We offer a GBDT-based accelerator capable of solving very complex models even in a small FPGA in this study. GBDTs have shown outstanding results in a variety of machine learning challenges, and their properties are particularly appealing to embedded systems. As a result, we developed a hardware accelerator for FPGAs with the goal of executing sophisticated models based on GBDT in low-cost, low-energy devices. In order to demonstrate.

Decision Trees with Gradient Boosting

A Decision Tree is a decision method that generates its result using a tree-like paradigm. It can be viewed as a means of displaying an algorithm with only conditional control statements. In a binary tree structure, each tree's decision is based on a sequence of comparisons connected between them. Each leaf node includes the outcome of the prediction [20], whereas each internal node indicates a comparison needed to determine the next node. When decision trees are used to solve classification problems, each tree leaf is labelled with the anticipated class, a probability for that class, or a probability distribution across all classes. Figure 1 depicts the use of a Decision Tree to solve a sequence of problems. First, this tree compares the value of feature 4 of the input to 20; when the input value decreases, it moves to the left child and repeats the process until it reaches the leaf with the output value of 0.3.

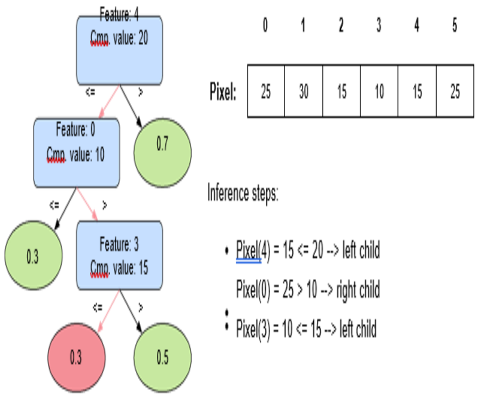


Figure 1. Decision Tree example.

One of the advantages of Decision Trees over other techniques is that no input preprocessing is required, such as data normalisation, scaling, or centering. They work with the input data in its current state [20]. The reason for this is that features are never blended together. As shown in Figure 1, the trees compare the value of an input feature to another value of the same feature in each comparison. As a result, multiple scales can be applied to various aspects. Other Machine Learning models combine characteristics to produce a single value; as a result, if their values are of various orders of magnitude, some features will initially dominate the outcome. This can be corrected during the training process, but in order to speed up training and increase results, normalisation will be required. Because avoiding normalisationminimises the number of run-time computations required, it's a particularly appealing feature for embedded systems. Furthermore, the magnitude of the input data has no bearing on the model's size or computations. As a result, dimensionality reduction techniques like Principal Component Analysis aren't required to shrink the model. This is also highly useful for embedded systems because it significantly minimises the amount of computing required during inference. During training, the most important attributes are chosen and employed for the tree's comparisons. As a result, features with more information will be employed more frequently in comparisons, whereas characteristics that do not provide helpful information for the classification problem will be omitted. This is an intriguing feature of this algorithm because, based on the same feature selection decisions made during training, we may easily repeat the process. This means that Decision Trees may be used to determine which features provide more useful information, and that this information can then be used to train even smaller models that retain the majority of the information while requiring less memory.

Nonetheless, for complex categorization tasks, a single Decision Tree does not deliver reliable answers. The approach is to employ an ensemble method, which aggregates the findings of multiple trees to enhance accuracy levels. Gradient Boosting is an ensemble method that integrates the outcomes of various predictors in such a way that each tree tries to improve on the preceding one's results. The gradient boosting approach entails training predictors in a sequential manner, with each subsequent iteration attempting to fix the residual error caused in the preceding iteration. In other words, each predictor is taught to correct the residual error of the one before it. The trees can then be utilised for prediction by simply adding the results of all the trees [20]. Designers can also use the GBDT model to trade off accuracy for computation and model size. If a GBDT is trained for 100 iterations, for example, it will produce 100 trees for each class. The designer might then choose whether or not to use all of them or to discard the final ones. Similar trade-offs can be seen in other machine learning methods, such as lowering the number of convolutional layers in a convolutional neural network (CNN). However, in that instance, each conceivable design must be retrained, but with GBDT, only one design must be retrained. Again, this is ideal for embedded systems since we can vary the model size based on memory resources, as well as execution time and power consumption constraints.

In terms of computation, the inference process in most machine learning algorithms necessitates a large number of floating point operations. CNNs and multilayer perceptrons (MLPs), for example, are built using floating-point multiply-accumulate operations. Calculating a tree's output, on the other hand, merely requires a few comparisons. All of these comparisons will only use numbers if the input data is integers, as it is with pixels in an image. This considerably decreases the computational load. In certain circumstances, the sole floating point operation will be the aggregation of each tree's outputs. As a result, each tree will only have one floating point addition. These improvements can be substituted in embedded systems by fixed precision operations, allowing the entire model to be executed even in systems without floating point units.

Design Architecture

For classification tasks, LightGBM employs a one-vs-all technique, which entails training a separate estimator (i.e., a set of trees) for each class, with each one predicting the same thing.The likelihood of belonging to that group. Each class has its own private trees in this technique, and the likelihood of belonging to a certain class is calculated by summing the results of the trees, as shown in Figure 2.

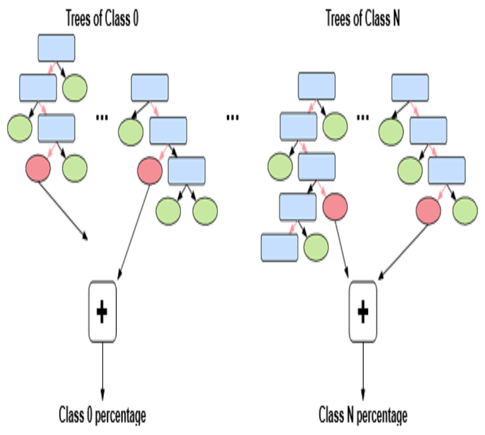


Figure 2. Gradient Boosting Decision Trees (GBDT) results accumulation with one-vs-all approach.

As a result, each class is independent of the others during inference, and the trees of each class can be studied in parallel. By providing one particular module for each class, our accelerator takes use of this parallelism.

It is critical to optimise memory resources when designing an efficient accelerator. To reduce data transfers to the external memory, we want to store the trees in the FPGA's on-chip memory resources. However, because those resources are limited, optimising the format used to store the trees in order to reduce their memory requirements is a major part of the design. All of a class's trees are stored in the trees nodes RAM memory, which is unique to the class module. The format we used to store the tree structure on this memory is shown in Figure 3. Our goal is to fit all of the information about each node into a 32-bit word. This word size can be increased or decreased as needed because our code uses generic parameters. However, in our tests, we found that 32 bits provided a fair balance between tree representation accuracy and storage requirements. These 32 bits are formatted differently depending on whether they are leaf nodes or not. There are four fields in the non-leaf format. The first and second fields record the information needed to perform the comparison, such as which input (8 bit) will be utilised and which value will be compared (16 bit). The addresses of the child nodes must then be saved. It is not possible to save its exact addresses because we only have 8 bits left. In fact, given the size of the memories we're employing, we'd require nearly all of the 32-bits to hold that data. We came up with two strategies to fix this problem. To begin, nodes are stored in the trees nodes memory using the pre-order traversal approach, which means that the left child of a non-leaf node is always allocated. The address of the left child does not need to be kept in this method because it may be acquired by adding one to the current address. As a result, we just need to save the address of the correct child. Second, rather than keeping the right child's absolute address, we store a relative address that specifies its distance from the current address. A 7-bit field is used to store the relative distance. The greatest depth of a tree using this method is 128. This is more than adequate for all of the trees we've looked at, as GBDT doesn't rely on really massive trees, but rather on a vast number of them. Finally, we've included a flag in each node's less significant bit: This In our tests, 14 bits were sufficient for absolute addresses. The LightGBM GBDTs' initial output is a 32-bit floating point number. In our research, however, we get similar accuracy using a 16-bit fixed-point representation. In any case, by utilising relative addresses for the @next tree field instead of absolute addresses, it is possible to use additional bits for the output without increasing the size of the memory word. The last two bits are two flags that indicate if this is the class's final tree and whether or not the node is a leaf.

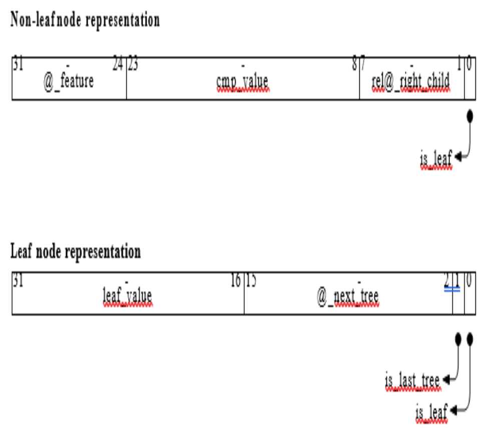
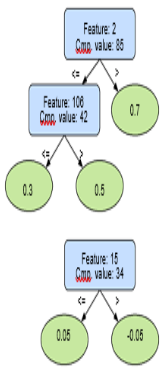


Figure 3. Node representation.

Figure 4 shows a simple example of storing two trees of the same class. As seen in the diagram, the first tree's root node is stored at address 0. Then, following these same criteria recursively, the full tree structure corresponding to its left child is stored, and ultimately, the right child is stored last. The relative leap to its right child is stored in the bits corresponding to the rel@ right child field. The first tree's leaf nodes all indicate that the following tree starts at address 5. Finally, the second tree's leaf nodes signal that there are no more trees to be processed. There are eight nodes in this simple example. We will be successful if we implement it in our architecture.



@	31 - 24	23 - 16	15 - 8	7 - 2	1	0
0	2	85		4		0
1	106	42		2		0
2	0.3		5		0	1
3	0.5		5		0	1
4	0.7		5		0	1
5	15	34		2		0
6	0.05		-		1	1
7	-0.05		-		1	1

Figure 4. Trees representation example.

The internal design of one of the modules that executes a class's trees is shown in Figure 5. A register, as well as the previously mentioned trees nodes RAM memory, are included in the design. last node is used to hold the address of the most recently visited node, as well as the logic that performs comparisons, computes the next node to visit, and accumulates the tree results.

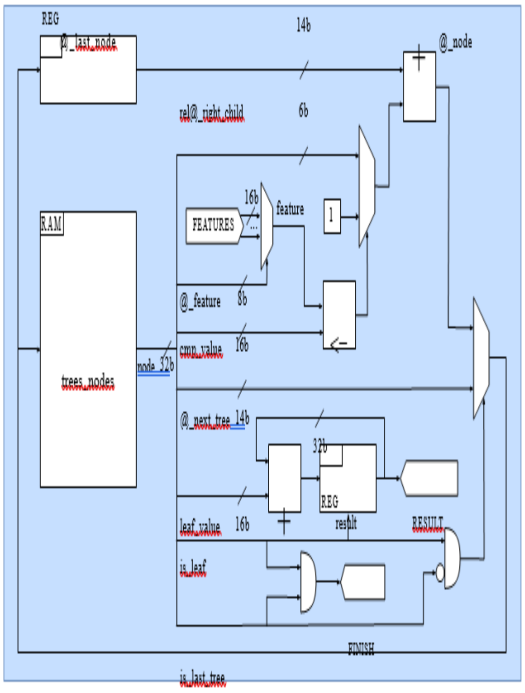


Figure 5. Class diagram.

The feature field of non-leaf nodes is utilised in this design to select one feature from all of the system's input features. The cmp value field is compared to the selected feature. If the feature's value is less than or equal to the cmp value, the non-leaf node's left child is chosen. To accomplish this, we increase the value of @ last node by one. Otherwise, the rel@ right child is used to choose the correct child. If the current node is a non-leaf (is leaf value is 0), this will generate the @ node that will be used to address the trees nodes RAM memory. If the current node is a leaf (is leaf = 1) but not the last tree (is last tree = 0), the Because we dedicate 14 bits to the @ next tree, the maximum size of the trees nodes RAM will be 214 words, and the theoretical maximum number of nodes in the same tree will be 26 due to the size of the relative jump rel@ right child, the maximum size of the trees nodes RAM will be 214 words. In terms of RAM capacity, we could address any number of trees by making @ next tree a relative address from the current node and adding it to @ last node, but the current size is significantly larger than our requirements. We use the 13 least significant of the 14 significant bits in our design to assign 8 BRAMs of 32 Kb to the trees of each class, resulting in 8192 words of 32 bits. We simply need to wait till every class module has completed before checking the output of the argmax module, which selects the best pixel, once we have received the features of one pixel.The number of students in the class receiving the best grade. Figure 6 shows a simplified version of the accelerator that demonstrates this behaviour, omitting the control lines and communications management.

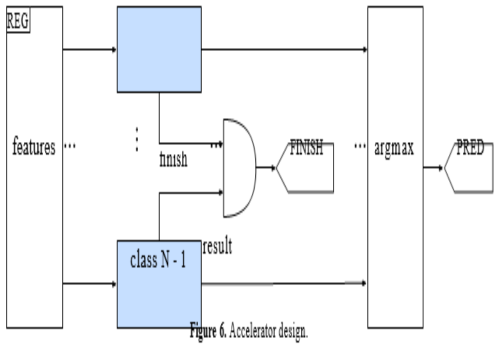


Figure 6. Accelerator design.

We simply need to wait till every class module has completed before checking the output of the argmax module, which selects the best pixel, once we have received the features of one pixel. The number of students in the class receiving the best grade. Figure 6 shows a simplified version of the accelerator that demonstrates this behaviour, omitting the control lines and communications management. Each clock cycle, this architecture can process one node per class. However, in our tests, if we construct a system that consumes the majority of the on-chip memory resources, the place&route procedure becomes complicated, and the clock frequency drops to 55 MHz. This can be fixed by utilising a more current FPGA with greater capacity and better integration technology, but it may also be improved by using computer architecture optimization techniques similar to those used to enhance the execution of general-purpose processors. The following diagrams will help to demonstrate this point. The execution of the previously stated version is shown in Figure 7a (single-cycle implementation). The execution of the nodes in one of the classes is depicted in the diagram. If we work together, we can

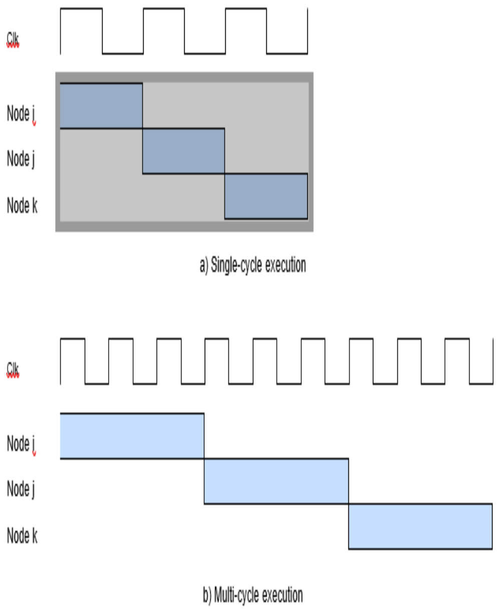


Figure 7. (a) Single-cycle execution scheme. (b) Multi-cycle execution scheme.

Our goal is to increase the processing speed while maintaining one node each cycle. We devised a multi-cycle approach to minimise the clock cycle. We looked at three alternative clock cycle options: two, three, and four. Additional registers have been added to the design in these versions in order to separate the longest combinational paths. We chose the version that executes the nodes in three cycles based on that study because it achieves a significant clock-period reduction while also providing a clear architecture that makes it easy to identify the operations that are carried out in each cycle. The benefits of four cycles were minimal, and the resulting execution pattern was not obvious. The results are shown in Figure 7b. The clock frequency has been improved, as shown in the diagram, but the system is still slower than before because each node requires three clock cycles. However, since the architecture has been partitioned in a logical manner, we may try a pipeline method. We have three different pipeline phases, as shown in Figure 8a, and each one consumes various hardware resources. As a result, as soon as the first node completes the first stage, we can begin executing a second node. The first stage in our design is to read the node (fetch), the second step is to identify the kind of node and read the required feature (decode), and the final step is to write the code. The problem with this technique is that we don't know which node will come next until the preceding node has completed its execution stage. As a result, we won't be able to get it ahead of time. As a result, a basic pipeline will be of little use. This is the same issue that traditional processors have when dealing with conditional branches in instructions. High-performance processors solve this problem by including complex support for speculative execution. However, this is not an efficient solution for embedded systems because it adds significant energy overheads, and it won't produce good results unless clear patterns for branch predictions can be identified. As a result, there is no simple way to benefit from the pipeline architecture.

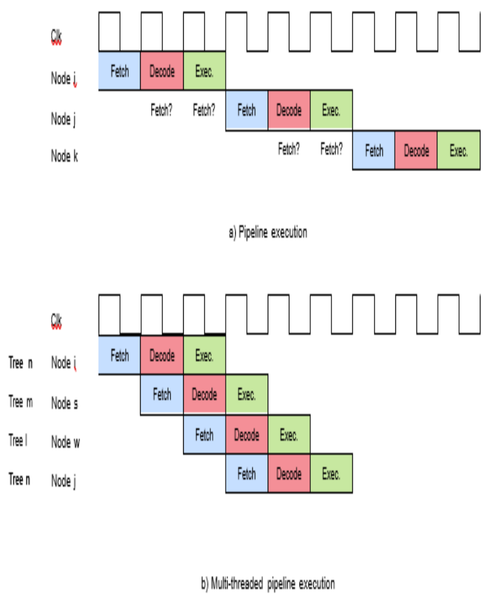


Figure 8. (a) Pipeline execution scheme. (b) Multi-threaded pipeline execution scheme.

However, by combining the pipeline with a multi-threading technique, this problem can be alleviated. The concept is to incorporate functionality to allow the execution of three distinct trees to be interleaved. Three @ last node registers can be used to accomplish this (that is the same as having three programme counters in a processor). A class's trees are separated into three sets, and each counter is in charge of one of these sets. Figure 8b demonstrates how the three trees' executions are interleaved. Three distinct trees (n, m, and l) are used in this example. We have the same time with this approach as with the multi-cycle approach, and we may execute one node every cycle, just like with the single-cycle implementation.

Figure 9 depicts the multi-threaded class module's final structure. The hardware overhead is minimal, as shown in the diagram. We only need a few registers to store the information needed for each stage, the additional @ last node counters, two additional registers indicating the starting address of the first tree in each set, three 1-bit registers indicating whether the processing of each set has been completed, and finally modify the control unit. Furthermore, both versions have the same memory resources, which are the most important in our design.

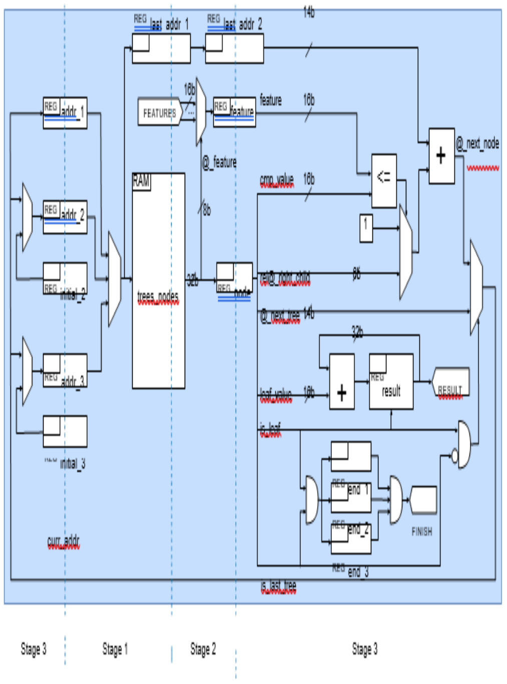


Figure 9. Multi-threading class diagram.

DATASET AND MODELS

Hundreds of spectral bands make up hyperspectral photographs, with each band capturing the responses of ground objects at a specific wavelength. As a result, each image pixel can be thought of as a spectral signature, as seen in Figure 10. Machine learning approaches, as discussed in [10], can classify pixels in hyperspectral images with high accuracy. Convolutional neural networks produced the most effective outcomes. However, because to the large size of hyperspectral images, it is a computationally costly process that cannot be used in on-board systems with limited resources. This study discovered that GBDTs present a highly intriguing trade-off between the usage of computing and hardware resources and the accuracy acquired, because it achieved high accuracy rates while carrying out the task. Our goal is to attain the same accuracy results as those described in [10] while running the trees through our accelerator, which is based on a tiny FPGA. We also want to process the pixels at the same rate as they are created by the hyperspectral sensors as a secondary goal.

The input for hyperspectral picture pixel classification is a single pixel made up of a series of characteristics, each of which is a 16-bit integer. The amount of features in a picture is determined by the sensor used to capture it; in our case, we employed datasets with 103 to 224 features. Each node in the tree chooses one of these characteristics and compares it to the value stored in the node to determine whether to choose the left or right child.

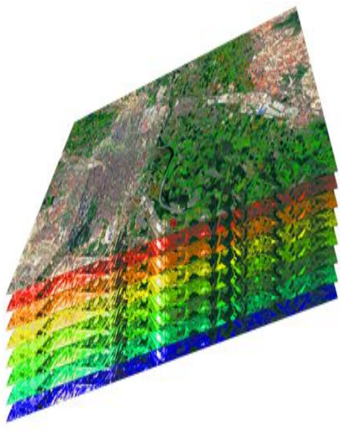


Figure 10. Hyperspectral image example.

Our design was created in VHDL with generic parameters to allow for a lot of customization. As a result, it may be applied to a variety of images with varying numbers of classes, input features, and trees per class... Table 1 summarises the main characteristics of the hyperspectral images analysed, Indian Pines (IP), Kennedy Space Center (KSC), Pavia University (PU), and Salinas Valley (SV), as well as the GBDT model configuration chosen based on those presented in [10], which analysed the various GBDT parameters and chose the best in each case. The table displays the number of features (ft.) and classes (cl.) in each image, as well as the total number of trees employed (trees), as well as several essential model parameters used to train the models, such as the minimum number of trees. Finally, it shows the accuracy gained using the LightGBM library to execute the models in [10], as well as the accuracy reached using identical models customised for our accelerator. The adjustments in the initial decision trees have essentially no influence on the final accuracy, as seen in the table.

[11] contains the models that have been codified using the node representation format specified.

Table 1. Datasets and model configuration.

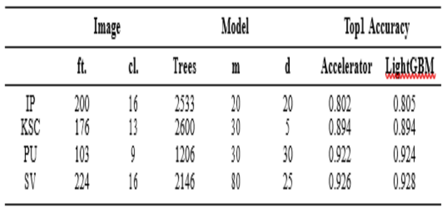
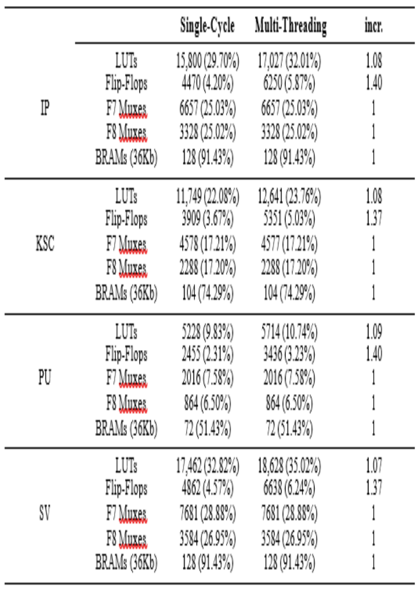


	Image		Model			Top1 Accuracy	
	ft.	cl.	Trees	m	d	Accelerator	LightGBM
IP	200	16	2533	20	20	0.802	0.805
KSC	176	13	2600	30	5	0.894	0.894
PU	103	9	1206	30	30	0.922	0.924
SV	224	16	2146	80	25	0.926	0.928

Experimental Results

The accelerator was created to fit in the FPGA of the Zedboard Xilinx Zynq-7000 evaluation board (Digilent, Hong Kong, China) [12]. Because this is a relatively small FPGA with outdated technology, the design requirements are fairly limited. The major purpose of employing this device is to replicate the properties of rad-hard and rad-tolerant FPGAs that have been certified for embedded on-board devices. Because these devices do not use the most up-to-date integration technology, using the most recent generation of FPGAs will be impractical. The hardware resources of single-cycle and multi-threading designs in this FPGA are shown in Table 2.

Table 2. Single-cycle vs. multi-threading resources.



		Single-Cycle	Multi-Threading	incr.
IP	LUTs	15,800 (29.70%)	17,027 (32.01%)	1.08
	Flip-Flops	4470 (4.20%)	6250 (5.87%)	1.40
	F7 Muxes	6657 (25.03%)	6657 (25.03%)	1
	F8 Muxes	3328 (25.02%)	3328 (25.02%)	1
	BRAMs (36Kb)	128 (91.43%)	128 (91.43%)	1
KSC	LUTs	11,749 (22.08%)	12,641 (23.76%)	1.08
	Flip-Flops	3909 (3.67%)	5351 (5.03%)	1.37
	F7 Muxes	4578 (17.21%)	4577 (17.21%)	1
	F8 Muxes	2288 (17.20%)	2288 (17.20%)	1
	BRAMs (36Kb)	104 (74.29%)	104 (74.29%)	1
PU	LUTs	5228 (9.83%)	5714 (10.74%)	1.09
	Flip-Flops	2455 (2.31%)	3436 (3.23%)	1.40
	F7 Muxes	2016 (7.58%)	2016 (7.58%)	1
	F8 Muxes	864 (6.50%)	864 (6.50%)	1
	BRAMs (36Kb)	72 (51.43%)	72 (51.43%)	1
SV	LUTs	17,462 (32.82%)	18,628 (35.02%)	1.07
	Flip-Flops	4862 (4.57%)	6638 (6.24%)	1.37
	F7 Muxes	7681 (28.88%)	7681 (28.88%)	1
	F8 Muxes	3584 (26.95%)	3584 (26.95%)	1
	BRAMs (36Kb)	128 (91.43%)	128 (91.43%)	1

The amount of LUTs and Flip-Flops is the only discernible increase; the rest is null or minor. The design doesn't care about these increments because it only uses a small portion of these resources. The on-chip memory resources are the system's bottleneck, as seen in the table. In fact, some photos consume nearly all of the available RAM. As a result, it was critical to optimise the memory format for storing the trees.

The performance results are shown in Table 3. In this example, the multi-threaded architecture improves speed by 67–85 percent. As a result, the computer architecture modifications in this accelerator deliver a significant performance boost at a low space cost. The Multi-threaded design incurs a slight penalty in terms of the number of cycles required to process each node (Avg. Cycles/Node), resulting in a greater number of cycles per pixel (Avg. Cycles/px.). The reason for this is that the trees are separated into three groups, each of which will not be properly balanced at all times. Only two of the three threads have useful work when one of the sets finishes. To alleviate this issue, we divided the trees into groups based on their average depth, attempting to balance the burden across all groups. We can't guarantee a perfect balance because the depth of the trees depends on the path picked and will be different for each input, thus these modest penalties arise. In terms of communications, we've implemented a DMA to read the input data from the external off-chip memory in our design. Because the latency of transferring one pixel's input data is less than the computation itself, it can be completely overlapped with the processing time of the preceding input data and has no effect on throughput.

We ran the inference of the models described in [10] in a high performance (HP) CPU, an Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz (Intel Corporation, Santa Clara, CA, USA), as well as an embedded system (ES) CPU, the Dual-core ARM Cortex-A9 @ 667 MHz (Arm Ltd., Cambridge, England) that is part of the same system-on-a-chip. gcc 7.3.1 was used to compile this code, with the -O3 optimization setting. Table 4 displays the total energy required for each image to complete the entire test set, as well as the performance in pixels per second. These tests were carried out using a Yokogawa WT210 digital power metre, which has been approved by the Standard Performance Evaluation Corporation (SPEC) for power efficiency benchmarking [21]. Only the dynamic energy consumption, i.e. the consumption caused by the trees' execution, is included in the table. To that aim, we examined the power consumption of the FPGA, Cortex-A9, and i5-8400 for five minutes, both in idle mode and during the execution of the models, and calculated the average increase in power consumption from those measurements.

Table 3. Single-cycle vs. multi-threading throughput.

		Single-Cycle	Multi-Threading	Gain
IP	Freq. (MHz)	60.61	105.263	1.69
	Avg. Cycles/px.	1658.15	1700.9	
	Avg. µs/px.	27.36	16.16	
	Avg. px/s.	36,552.78	61,880.05	
	Avg. Cycles/Node	1	1.026	
KSC	Freq. (MHz)	62.5	105.263	1.67
	Avg. Cycles/px.	2542.76	2564.55	
	Avg. µs/px.	40.69	24.36	
	Avg. px/s.	24,579.60	41,045.41	
	Avg. Cycles/Node	1	1.009	
PU	Freq. (MHz)	66.67	125	1.80
	Avg. Cycles/px.	1857.34	1938.70	
	Avg. µs/px.	27.86	15.51	
	Avg. px/s.	35,895.42	64,476.20	
	Avg. Cycles/Node	1	1.044	
SV	Freq. (MHz)	55.56	105.263	1.85
	Avg. Cycles/px.	1447.13	1479.37	
	Avg. µs/px.	26.05	14.05	
	Avg. px/s.	38,393.23	71,153.94	
	Avg. Cycles/Node	1	1.022	

Table 4. Energy and performance comparisons.

		Energy (J)			Performance (Pixels/s)		
	Test Pixels	FPGA	HP CPU	ES CPU	FPGA	HP CPU	ES CPU
IP	8721	0.280	27.750	9.858	62,293	23,568	1327
KSC	4435	0.136	6.150	3.024	65,221	53,860	2200
PU	38503	1.194	58.500	15.960	64,494	49,336	3619
SV	48726	1.370	171.000	36.863	71,133	21,332	1983
Average		0.500	36.147	11.508	65,706	22,422	2139

According to these findings, the HP CPU execution consumes 72 times more energy than the FPGA design when doing test bench inference, despite the fact that the FPGA design is twice as quick. The ES CPU consumes 23 times more energy than the FPGA architecture while being 30 times faster. The last comparison is particularly intriguing because this processor is on the same chip as the FPGA, uses the same main memory, and runs at a nearly seven-fold quicker speed. This highlights the performance and energy efficiency advantages of a bespoke accelerator. Finally, one of our goals was to analyse the pixels at the same rate as they were produced by the hyperspectral sensors. According to [22], the AVIRIS sensor used to collect the majority of our datasets needs to process 62, 873.6 pixels per second to attain complete real-time performance. Ourmulti-threading design achieves this speed in three of the four images (KSC, PUandSV), and achieves 98.4% of that speed in a third one(IP). Hence, wecanconcludethatthisdesigniscapableofachievingreal-timeperformancein many scenarios even using a small FPGA.

CONCLUSION

GDBT is a machine learning model with a lot of power. Because its main operations are simple comparisons, its properties make it particularly well suited to being accelerated in FPGAs. Memory constraints were a major consideration in the accelerator's design. The key to allowing sophisticated models to be implemented in small FPGAs has been optimising the format used to store the trees. Furthermore, after examining our accelerator's execution, we discovered a pipeline and multi-threading execution scheme that maximises FPGA resource consumption and produces a 67–85 percent speed gain over single-cycle execution. For a relevant case study, we tested our accelerator with complex models and demonstrated that it can handle data at high speeds. with a very low power usage overhead during runtime Furthermore, when compared to LightGBM execution on a high-performing CPU, our model achieves 2x performance while consuming 72x less energy. In the instance of an embedded system CPU, our solution improves performance by 30 times while consuming 23 times less energy during execution. As a result, this architecture is ideal for high-performance embedded systems, which was our initial goal.

REFERENCES

J.H. Friedman, J.H. Friedman, J.H. Friedman, J.H. Friedman, J.H. Friedman, J.H. Comput. Stat. Data Anal., vol. 38, no. 3, pp. 367–378. [CrossRef]

XGBoost: A Scalable Tree Boosting System, T. Chen and C. Guestrin. pp. 785–794 in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16), San Francisco, CA, USA, August 13–17, 2016.

CatBoost: unbiased boosting using categorical features. Prokhorenkova, L.; Gusev, G.; Vorobev, A.; Dorogush, A.V.; Gulin, A.

arXiv:1706.09516, arXiv:1706.09516, arXiv: 1706.09516, arXiv:17 G. Ke, Q. Meng, T. Finley, T. Wang, T. Chen, W. Ma, W. Ye, Q. Ye, T.Y. Liu, T.Y. Liu, T.Y. Liu, T.Y. Liu, T.Y. Liu, T.Y. Liu, T.Y. Liu, T.Y. Liu, T.Y. Li A Highly Efficient Gradient Boosting Decision Tree is LightGBM. In Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., Eds., Advances in Neural Information Processing Systems 30; Curran Associates, Inc.: Red Hook, NY, USA, 2017; pp. 3146–3154.

Winning Solutions for Microsoft's Machine Learning Challenge. Microsoft, 2020. Git Hub: https://github.com/microsoft/LightGBM/blob/master/examples/README.md (accessed on 28 January 2021).

6. Li, S.; Lin, Y.; Zhu, T.; Fan, M.; Xu, S.; Qiu, W.; Chen, C.; Li, L.; Wang, Y.; Yan, J.; et al. Li, S.; Lin, Y.; Zhu, T.; Fan, M.; Xu, S.; Qiu, W.; Chen, C.; Li, L.; Wang, Machine learning technology was used to develop and externally evaluate prediction models for

COVID-19 patient mortality. 2021, Neural Comput. Appl. [CrossRef] [PubMed].

C. Deotte. 1st Place Solution in the IEEE-CIS Fraud Detection Contest. https://www.kaggle.com/c/ieee-fraud-detection/discussion/111284 (retrieved on January 28, 2021).

8. Zhang, T.; He, W.; Zheng, H.; Cui, Y.; Song, H.; Fu, S. Zhang, T.; He, W.; Zheng, H.; Cui, Y.; Song, H.; Fu, S. A gradient boosting decision tree is used to estimate ground PM2.5 using satellite data. 128801 in Chemosphere 2020. [CrossRef] [PubMed]

R. Sun, G. Wang, W. Zhang, L.T. Hsu, and W. Ochieng A GPS signal reception categorization system based on a gradient boosting decision tree. Appl. Soft Computing, vol. 86, no. 105942, 2019. [CrossRef]

A. Alcolea, M.E. Paoletti, J.M. Haut, J.M. Resano, J. Plaza, A. Alcolea, M.E. Paoletti, J.M. Haut, J.M. Resano, J.M. Resano, J.M. Resano, J Supervised Spectral Classifiers for On-Board Hyperspectral Inference

An Overview of Imaging 534 in Remote Sensing 2020. [CrossRef]

FPGA Accelerator for GBDT. Source Code and Models. 2021. Alcolea, A.; Resano, J. AdrianAlcolea/FPGA accelerator for GBDT is available on GitHub: https://github.com/ AdrianAlcolea/FPGA accelerator for GBDT (Accessed on January 28th)

vPipelined Decision Tree Classification Accelerator Implementation in FPGA by F. Saqib, A. Dutta, J. Plusquellic, P. Ortiz, and M.S. Pattichis (DT-CAIF). IEEE Transactions on Computers, vol. 64, no. 2, pp. 280–285. [CrossRef]

FPGA Implementation of Decision Trees and Tree Ensembles for Character Recognition in VivadoHls. 15. Kuaga, R.; Gorgon, M. Commun. 2014, 19, 71–82. Image Process. [CrossRef]

B. Van Essen, C. Macaraeg, M. Gokhale, R. Prenger, B. Van Essen, C. Macaraeg, C. Macaraeg, C. Macaraeg, C. Macaraeg, C. Macarae Multi-Core, GP-GPU, or FPGA for Random Forest Classifier Acceleration?

232–239. In IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, Toronto, ON, Canada, April 29–May 1, 2012; pp. 232–239.

17. Oberg, J., Eguro, K., Bittner, R., and Forin, A. FPGA-based random decision tree body part recognition. The Proceedings of the 22nd International Conference on

FPL (Field Programmable Logic and Applications) is an international conference on field programma

Citation: *Annadasu Saiprathima et al, "Gradient Boosting Decision Trees with FPGA Accelerator", International Journal of Research Studies in Science, Engineering and Technology. 2019; 6(11): 74-86. DOI: https://doi.org/10.22259/2349-476X.0611007*