Parte I

ANEXOS

GESTIÓN DEL TIEMPO

A.1 TABLA DE HORAS DEDICADAS

Tabla 1: Horas dedicadas al proyecto

Tarea	Horas
Estudio del *framework* LightGBM y entrenamiento de árboles	9
Tratamiento de datos para su utilización en la **FPGA!**	27
Diseño e implementación del circuito	63
Diseño e implementación en C (Simulador e Interacción con **FPGA!**)	9
Implementación del módulo de depuración	53
Depuración del circuito	45
Redacción de la memoria	57
Redacción del artículo HiPEAC	11
Reuniónes con el director	7
Experimentación	11
Total	292

A.2 DIAGRAMA DE GANTT DEL PROYECTO

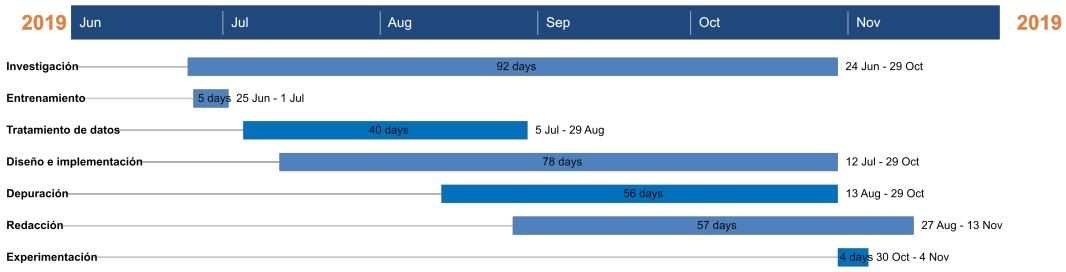

Figura 1: Diagrama de gantt

Preliminary FPGA implementation of an accelerator for Gradient Boosting Decision Trees

Vlad Teletin[1], Adrián Alcolea[1], and Javier Resano[1]

University of Zaragoza, Spain
{vlad, alcolea, jresano}@unizar.es

Abstract. A decision tree is a well-known machine learning technique. Recently their popularity has increased due to the powerful Gradient Boosting ensemble method that allows to gradually increasing accuracy at the cost of executing a large number of decision trees. In this paper we present an initial architecture to accelerate the execution of these trees while reducing the energy consumption. We have implemented it in an ARM-based System-On-Chip that includes an FPGA, and we have tested it with a relevant case-study: pixel classification of hyperspectral images. In our experiment both execution time and energy consumption are reduced by a factor of ten.

Keywords: Decision Trees · GBDT · FPGA · Energy efficiency

1 Introduction

Decision trees are a very light and efficient machine learning technique that have proved their effectiveness in several classification problems. In the context of embedded systems, energy efficiency is as much important as accuracy, so it is necessary to search for efficient algorithms liable to be accelerated. This makes the decision trees a perfect target to develop an FPGA accelerator.

A single decision tree is frequently not very accurate for complicated tasks but, thanks to ensemble methods, it is possible to combine several trees in order to deal with complex problems. Gradient Boosting is an ensemble method that allows to gradually increasing accuracy by adding new trees along several iterations with very simple operations.

In this paper we present an initial version of an accelerator for Gradient Boosting Decision Trees (GBDT). To demonstrate its potential we have implemented it in a Zynq-7000 evaluation board [10] which includes an FPGA embedded in an ARM-based System-On-Chip (SOC), and we have used it for a relevant case study: pixel classification of hyperspectral images. We have measured the execution time and power consumption of our accelerator and compare it to an equivalent C code executed in one of the ARM processors of the SoC. Our design requires one order of magnitude fewer time than the execution of the C algorithm compiled with the highest level of optimization of the gcc compiler, while the power overhead of the accelerator is almost negligible. Hence, it also reduces one order of magnitude the energy consumption.

2 Related work

Decision Trees are a well-known algorithm that can achieve good results for regression and classification problems with a very small amount of calculations. One of its advantages is that it demands less data preprocessing than other machine learning solutions, since it does not combine the different input parameters [3]. Furthermore, recent software solutions specialized on Decision Trees are arising, combining ensemble methods such as Gradient Boosting with new powerful training techniques that reduce training time while increasing accuracy. LightGBM is one of these algorithms optimized for efficiency [4], and they also provide an specialized framework with Python interface that allows very fast and light experimentation [6]. In this work we present an accelerator that can execute the GBDT trained with LightGBM.

There are also some previous works about FPGA acceleration of Decision Trees [7,5,8]. [7] focuses on the training processes. In our case we assume that training is carried out offline and we want to focus on inference, which must be computed online. [5] propose to use a high-level synthesis approach to design an FPGA accelerator. They focus on Random Forest, which is an ensemble technique that combines the output of many trees trained with different input data to generate a more accurate and robust final output. We have decided to focus on GBDT instead of Random Forest since recently GBDT have demonstrated and enormous potential [4]. Moreover we prefer to design a custom architecture instead of using a high-level synthesis approach in order to fully control the final architecture. [8] presented a custom pipelined architecture which demonstrated the potential of an accelerator for decision trees. However they do not support GBDT and they apply their techniques only to simple case studies.

3 Gradient Boosting Decision Trees

A Decision Tree is a decision algorithm based on a series of comparisons connected between them as in a binary tree structure, so that the node comparisons lead the search to one of the child nodes, and so on, until reaching a leaf node that contains the result of the prediction [3]. Figure 1 shows the operation of a Decision Tree on a series of feature inputs with a toy example. In the first place, this tree takes feature 3 of the input and compares its value with 14300; as the input value is lower it continues on the left child, and keeps with the same procedure until it reaches the leaf with 0.15 as output value.

One of the benefits of using Decision Trees over other techniques is that they do not need any input preprocessing such as data normalization, scaling or centering. They work with the input data as it is [3]. The reason is that features are never mixed. As can be seen in Figure 1, in each comparison the trees compare the value of an input feature with another value of the same feature. Hence, several features can have different scales. In other Machine Learning models, features are mixed to generate a single value, therefore, if their values belong to different orders of magnitude, some features will initially dominate

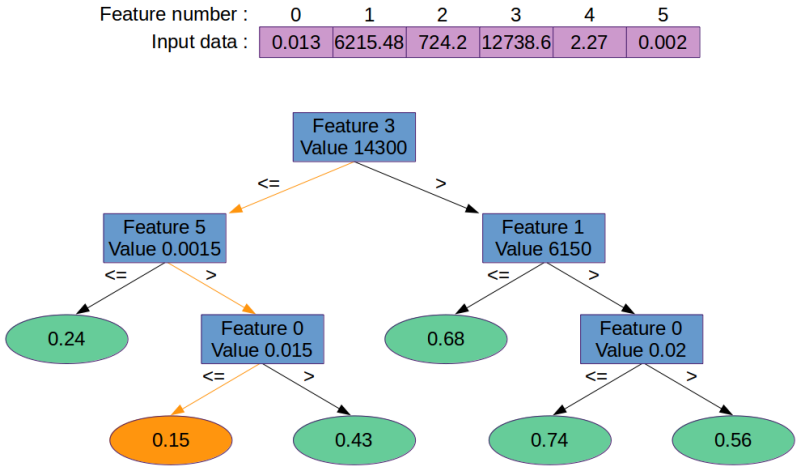

Fig. 1. Decision Tree example.

the result. This can be compensated during the training process, but in general normalization will be needed to speed up training and improve the results.

Besides, the size of the input data does not affect the size of the model, so dimensionality reduction techniques such as Principal Component Analysis are not needed to reduce the model size, which substantially reduces the amount of calculation needed at inference. During training the most meaningful features are selected and used for the comparisons in the tree. Hence the features that contain more information will be used more frequently for the comparison, whether those that do not provide useful information for the classification problem will simply be ignored. This is an interesting property of this algorithm since, based on the same decisions made during training to choose features, we can easily determine the feature importance. This means that Decision Trees can be used to find out which features carry the main information load, and that information can be used to train even smaller models keeping most of the information with less memory impact.

Nevertheless, a single Decision Tree does not provide accurate results for complex classification tasks. The solution is to use an ensemble method that combines the results of several trees in order to improve the accuracy levels. Gradient Boosting is an ensemble method that combines the results of different predictors in such a way that each tree attempts to improve the results of the previous ones. Specifically, the gradient boosting method consists in training predictors sequentially so each new iteration try to correct the residual error generated in the previous one. That is, each predictor is trained to correct the residual error of its predecessor. Once the trees are trained, they can be used for prediction by simply adding the results of all the trees [3].

The GBDT model also allows designers to trade off accuracy for computation and model size. For example, if a GBDT is trained for 100 iterations, it will

generate 100 trees for each class. Afterwards, the designer can decide whether to use all of them, or to discard the final ones. It is possible to find similar trade-off with other ML models, for instance reducing the number of convolutional layers in a CNN. However, in that case, each possible design must be trained again, whereas in GBDT only one train is needed, and afterwards the designer can simply evaluate the results when using different number of trees and generate a Pareto curve with the different trade-offs.

4 Hyperspectral image classification as testing target

Hyperspectral images consist of hundreds of spectral bands, where each band captures the responses of ground objects at a particular wavelength. Therefore, each pixel of the image can be considered as a spectral signature. Machine learning techniques have proven good results for hyperspectral pixel classification [1]. Nevertheless, the size of hyperspectral images makes it a very computationally intensive task, so in on-board systems with very limited resources it is not possible to use these solutions. These make it a perfect target for our accelerator. Moreover, due to the characteristics of the hyperspectral images classification problem, the calculations of the decision trees during inference only need integer operands, while most of the machine learning techniques are based on floating point operations.

In hyperspectral images pixel classification, the input is a single pixel composed of a series of features, where each feature is a 16-bit integer. Each node of the tree only uses one of these features, meaning that, at the time of inference, one particular node performs a single comparison between its trained value and the value of the corresponding feature. Since the feature values of hyperspectral images are 16-bit integers, each node just need an integer comparison to made their decision; i.e. left or right child. This is a very important feature. In most ML models the inputs are multiplied by a floating-point value, hence even when the input model is an integer, as happens in image processing, all the computations will be floating point. However, a tree only need to know whether the input is smaller or greater than a given value, and if the input is an integer, the comparison value can also be an integer without any accuracy loss. LightGBM follows a one-vs-all strategy for classification problems that consists in training a different estimator for each class, so each one of them predicts the probability of belonging to that class. With this approach each class has their own trees, so we just need to add the results of the trees of each class separately, as shown in Figure 2. This can be done in parallel.

So in the case of hyperspectral images pixel classification, this technique behaves exceptionally in terms of computation. As we explained, GBDT only need a few comparisons and some final accumulations, and for the hyperspectral images we only need integer arithmetic. For these reasons, FPGAs are a perfect target to accelerate GBDT for on-board processing.

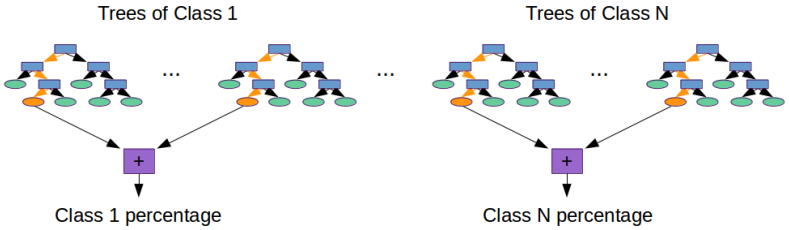

Fig. 2. GBDT results accumulation with one-vs-all approach.

5 FPGA implementation

A full binary tree can be easily represented in memory without pointers, following an in-order, pre-order or post-order traversal. Nevertheless, GBDT binary trees do not need to be full, so it is necessary to define a node structure that points to their children locations. Figure 3 shows the non-leaf node structure which contains the feature to compare with, the threshold value and the addresses of their left and right children. Figure 4 shows the leaf node structure which contains the result of the prediction, a flag that indicates if it is the last tree and, if needed, the address of the next tree. Both of them have a flag to indicate whether they are leaves or not. In this first approximation we decided to maintain a size of 64b and a fixed structure for leaf and non-leaf nodes even if there are some unused space, so we can directly map each part as signals in our design.

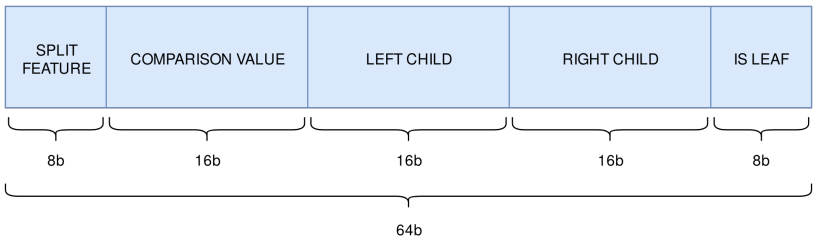

Fig. 3. Non-leaf node representation

As we use one-vs-all approach, trees are grouped by classes, so each group of trees predicts the probability of belonging to that class. The accelerator is composed of N modules working in parallel, each one of them dedicated to a different class, i.e. dedicated to the group of trees that predict the probability of that class. Figure 5 depicts the main structures and communications. Each module executes the trees adding their predictions, so at the end it gets the final value of the probability of belonging to that class. Once every module finishes, we just need to check which one of them reached the highest value to find the predicted class.

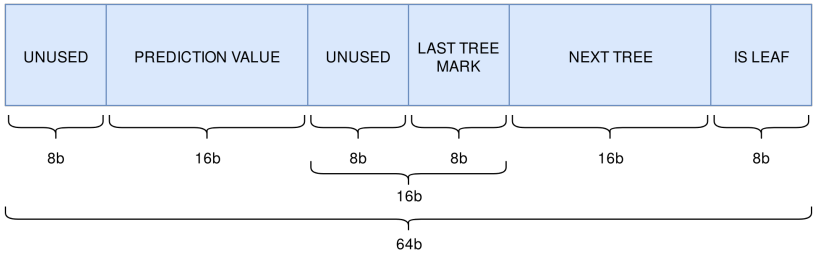

Fig. 4. Leaf node representation

The AXI block manages the communications between the ARM processor (an ARM Cortex-A9 [10]) and the FPGA using the ARM Advanced Extensible Interface (AXI) [9]. In particular in includes an AXI-lite interface that defines a set of registers that are visible to the ARM processor, and a AXI-stream interface that uses a Direct Memory Access (DMA) controller to send the data to process from the main memory to the accelerator. The Debug and the Checksum blocks allow to check that the trees are properly loaded in the accelerator in order to detect any communication error.

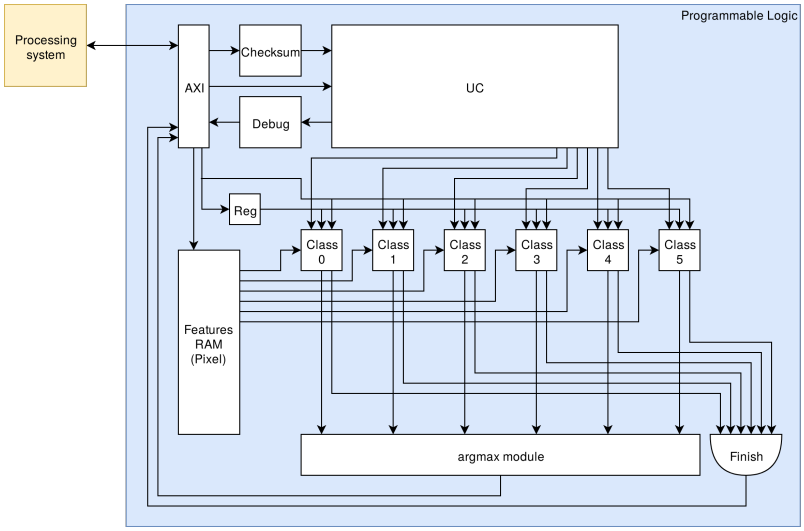

Fig. 5. Accelerator diagram

The operation of each node is depicted in 6. All the trees of the class are mapped into a RAM memory local to the module. When a non-leaf node is found, it performs the comparison of the input feature value with the node

threshold to select between left and right children. When a leaf node is found, its value is accumulated in the output register. If it is not the final tree of the class, then the root of the next tree is loaded. Once all the modules reach the final tree, the argmax function is applied to find the selected class, as shown in 5.

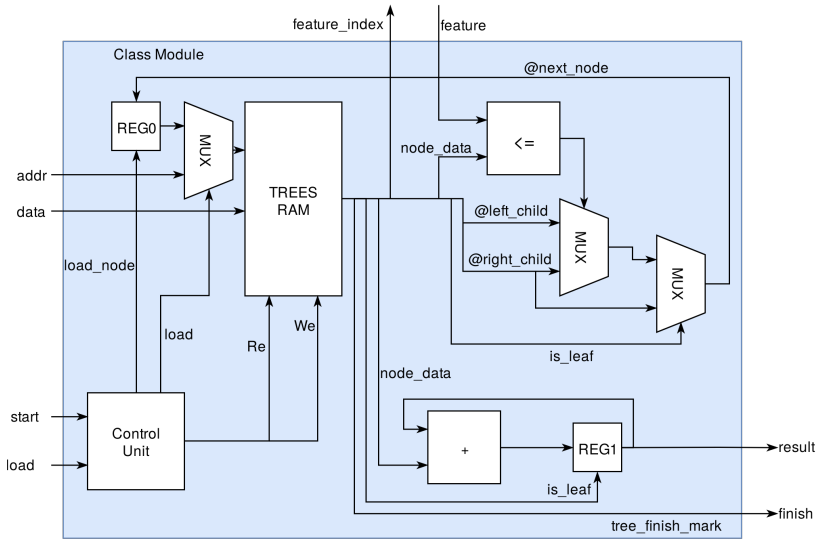

Fig. 6. Class diagram

The entire FPGA design require 2961 Flip-Flops (3% of the FPGA resources) and 17533 LUTs (30% of the FPGA resources). An important detail to consider on this design is that the local RAM memories have been directly mapped into LUTs, without using the BRAM blocks available in the FPGA. Actually, 13110 of the 17533 LUTs are used as memory, while only 4423 are used as logic for the design. In the next design iteration we will use the BRAM blocks provided in the FPGA, which are faster and more energy efficient for large memories.

6 Experimental results

For the experiments we trained a GBDT model on the Salinas-A Hyperspectral scene [2]. We use the 15% of the pixels of each class for training and we divide the rest into validation and testing sets, so at the end we keep 2276 pixels for testing. We trained our model for 100 iterations using the LightGBM framework [4] that generated 100 trees for each class. With this setup the classification accuracy was 99%. We compared the execution time and the energy consumption during

inference between the algorithm in C code running in the ARM Cortex A9 CPU and the FPGA implementation of the described design.

These experiments have been realized in the CPU and the FPGA of the SoC XC7Z020 of the Xilinx Zynq-7000 evaluation board [10]. For the CPU measures we tried different gcc compiler optimization levels. The CPU clock runs at its maximum frequency: 667MHz, while the FPGA design is clocked at 100MHz, which is the frequency of the communication lines with the CPU. Table 1 shows the total time consumed for the inference process of the complete test set in each case. As can be seen, the FPGA achieves a 8.8 speedup when compared with the -O3 execution of the CPU. In the case of the FPGA, the measure includes the time needed to send the input data form the main memory (512 MB DDR3) to the FPGA. 7.479ms of the total 15.277ms are due to the communication latency, while the remaining 7.798ms are the execution time of the computations. As this is a preliminary approach, we started with a very simple sequential design. In the next iteration, the logical evolution will be a pipelined architecture that allows to send the data of the next pixel while the current one is being processed.

Table 1. Comparison of execution time and energy consumption.

	CPU(-O0)	CPU(-O1)	CPU(-O2)	CPU(-O3)	FPGA
Time (ms)	806.910	189.942	134.001	133.980	15.277
Power (W)	1.436	1.436	1.436	1.436	1.453
Energy (J)	1.159	0.273	0.192	0.192	0.022

The energy consumption has been measured with the Digital Power Meter Yokogawa WT210 [11]. First, we measured the average static power of the evaluation board (3.1 W in our setup), so the power measures in table 1 correspond to the average dynamic power consumption. i.e. the average power consumption measurements after removing this 3.1 W. We can observe that FPGA and CPU executions have a very similar power consumption, so the energy savings of the FPGA design are proportional to the time savings, i.e. the energy is reduced by a factor of 8.7 compared to the -O3 execution of the CPU.

7 Conclusions

We have presented a preliminary version of an accelerator for GBDT inference. We have evaluated our design with a relevant case study and the results demonstrate that FPGAs are a suitable architecture for our accelerator since it achieves a x8.8 speedup and reduce the energy consumed during the data processing also by a similar factor when compared to an optimized C version executed in a ARM Cortex A9 processor in the same SoC.

We believe that this is a fair comparison since both, the FPGA and the processor, are implemented in the same technology and use the same memory resources. However, there is clearly room for improvement. First, our design uses

the LUT resources to store the trees information, but the FPGA includes specific RAM blocks (BRAMs) that are faster and more energy efficient, hence in the next version we will use them for our memory blocks. Second, we compute all the classes in parallel but all the trees in the same class (100 in our experiments) are executed sequentially. However it is also possible to execute these trees in parallel. In the next version we will explore this level of parallelism. Finally, half of the time consumed by our accelerator was due to the communication latency. If we add an additional input buffer it will be possible to overlap the communications of the next pixel with the computations of the current one. With these optimizations we expect to significantly improve the results in a future version.

References

1. Ghamisi, P., Plaza, J., Chen, Y., Li, J., Plaza, A.J.: Advanced spectral classifiers for hyperspectral images: A review. IEEE Geoscience and Remote Sensing Magazine **5**(1), 8–32 (March 2017)
2. GIC: Hyperspectral remote sensing scenes, grupo de inteligencia computacional de la universidad del país vasco. [online] http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes#Salinas-A_scene, accessed November 2019
3. Géron, A.: Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. O'Reilly Media, Inc (3 2017)
4. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: A highly efficient gradient boosting decision tree. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 3146–3154. Curran Associates, Inc. (2017), http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf
5. Kułaga, R., Gorgon, M.: Fpga implementation of decision trees and tree ensembles for character recognition in vivado hls. Image Processing & Communications **19** (09 2014). https://doi.org/10.1515/ipc-2015-0012
6. Microsoft: Lightgbm documentation. [online] https://lightgbm.readthedocs.io, accessed November 2019
7. Narayanan, R., Honbo, D., Memik, G., Choudhary, A., Zambreno, J.: An fpga implementation of decision tree classification. In: 2007 Design, Automation Test in Europe Conference Exhibition. pp. 1–6 (April 2007). https://doi.org/10.1109/DATE.2007.364589
8. Saqib, F., Dutta, A., Plusquellic, J., Ortiz, P., Pattichis, M.S.: Pipelined decision tree classification accelerator implementation in fpga (dt-caif). IEEE Transactions on Computers **64**(1), 280–285 (Jan 2015). https://doi.org/10.1109/TC.2013.204
9. Xilinx: Amba axi4 interface protocol. [online] https://www.xilinx.com/products/intellectual-property/axi.html, accessed November 2019
10. Xilinx: Zynq-7000, soc and fpga platform. [online] https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf
11. Yokogawa: Wt210/wt230 digital power meters. http://tmi.yokogawa.com/products/digital-power-analyzers/digital-power-analyzers/wt210wt230-digital-powermeters/#tm-wt210_01.htm, accessed November 2019