RF-SM: Random Forest Training Process
Acceleration with Subsampling Method on FPGA

Ping Lei
Graduate School of Information, Production and Systems
Waseda University
lei_ping@toki.waseda.jp

Shinji Kimura
Graduate School of Information, Production and Systems
Waseda University
shinji_kimura@waseda.jp

Abstract—Big data and machine learning algorithms have raised a great interest in hardware acceleration field in recent years. The high performance of FPGA calculations can overcome power constraints and can be used in data center acceleration. Random Forest (RF) is a well-known machine learning algorithm used in classification and prediction. Though RF has been implemented on FPGAs and GPUs by some studies to accelerate the training process, because of huge amount of data, the processing speed remains to be a bottleneck. To solve this problem, we proposed the subsampling method collaborated with FPGA for the acceleration of Random Forest training process. This would offload the computation intensive part to hardware to optimize the training process. This work implements the design using C in Vivado HLS for Xilinx Kintex 7 FPGA. We select a subsampling ratio of 10% for the training dataset. The average acceleration rate with the subsampling method can reach 10.24x compared to the C implementation. The classification accuracy of the evaluation result keeps at around 90%. The result shows an ideal acceleration speed while maintaining a satisfactory accuracy.

Keywords—random forest; FPGA; decision tree; subsampling; big data; acceleration

I. Introduction

As we step in the new era of artificial intelligence, there appears a lot of acceleration-aware machine learning algorithms. Among these popular algorithms, Random Forest (RF) is the one widely used in sentiment analysis, search engines and prediction models; because it can maintain a high accuracy when handling large datasets. Leading IT companies, such as Microsoft, uses FPGAs to accelerate Bing data center; Baidu also includes Xilinx FPGAs in cloud pools to accelerate machine learning inference.

RF is generally consisting of several Decision Trees (DTs), and is first proposed by Tin Kam Ho in [9] in 1995. Each decision tree is a well-trained classifier, by putting a data instance into each decision tree, we can get a result of the final classification k from each tree. The RF then gather these results and do majority voting to reduce the errors and out predicate the final class the instance should be in. In this way, we can avoid the overfitting problem appeared in decision tree models.

Most of today’s RF is implemented on CPUs, but it may take a huge amount of time when dealing with dataset of gigantic data quantity. The requirement of RF in data centers is also facing a problem of the large amount of power consumption. In this case, some researchers have developed the RF on the FPGA/GP-GPU platform to speed up the calculation while maintaining a satisfied accuracy.

Several previous works have studied on the FPGA based RF or DT acceleration. The main results of the related works have been compared in Table I. with parameters such as hardware/software speedup and average accuracy.

For RF training process acceleration, [1] proposed an optimized Gini unit in hardware with the speedup of 5.58x compared to the C implementation. The Gini unit can greatly reduce the hardware resource utilization by only using divider and adder in splitter calculation.

Ref. [2] proposed a FIFO sorter in RF training process on FPGA and achieve an average speed up of 171x over the employed problems. However, the accuracy based on 20 classification problems is just about 48% because of the batch learning approach.

For RF classification process acceleration, [3] compared RF classifiers on Multi-core, GP-GPU, and FPGA. This work proposed a compact random forest (CRF) among which the trees have low depth. Clumps of FPGA units are used to control the demands on the FFs, LUTs and routing resources. The result show that FPGA gets the best performance on accuracy, power, and scalability. A key contribution of this work is exploiting the low tree depth of the CRF model to regularize the decision trees and thus can create pipelined and SIMT algorithms.

Ref. [4] implemented a pipelined DT engine with distinct stages in FPGA with average classification accuracy around 91%. It focused on pipelining of various stages and efficient

![Table I. Related Works](image)

<table>
<thead>
<tr>
<th>Previous Works</th>
<th>Acceleration Part</th>
<th>Speedup vs. Software</th>
<th>Speedup vs. Hardware</th>
<th>Average Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE Work [1]</td>
<td>Training</td>
<td>5.58x</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>FPL Work [2]</td>
<td>Training</td>
<td>171x</td>
<td>/</td>
<td>48%</td>
</tr>
<tr>
<td>FCCM Work [3]</td>
<td>Classification</td>
<td>/</td>
<td>/</td>
<td>86%</td>
</tr>
<tr>
<td>TC Work [4]</td>
<td>Classification</td>
<td>130x</td>
<td>3.5x vs. Work [1]</td>
<td>91%</td>
</tr>
</tbody>
</table>
use of the on-chip memories.

In this work, we proposed a subsampling method to accelerate the big data processing speed using RF algorithm. The improvement on the Gini index calculation and data sorting procedure are also proposed. Besides, a local search procedure is added to the algorithm to compensate the accuracy loss caused by less training samples.

II. Preliminaries

Decision tree is a kind of classifier generated in the training process and output the classification or regression result. Each internal node represents one split attribute. Considering the training process, it is essential to decide which attribute should be used at each node from a top-down order. The original DT learning has several kinds of algorithms, CART (classification and regression tree) is considered in this work.

The training process of a decision tree corresponds to determining the judgement (i.e., the split point) of each of the internal nodes and the prediction value of each of the leaf node. For simplicity of explanation, only binary tree is considered in this paper, so that there are only two edges emanating from each internal node.

In the training process, there will be a set of data samples with the corresponding classIDs already known, which is called training set by convention. The judgement is determined from the top to the bottom. At each branch node, the judgement is calculated using the Gini index. The tree executes this procedure until the stop criterion is satisfied. Each decision tree in the RF is generated in this way by using different training datasets. The tree depth and the number of trees are key parameters of the algorithm.

The flow-chart in Fig.2 shows the RF training process. The building of the next split is extended out in detail because it is the most essential part we focused on in this work. As we can see from the beginning, different data subsets are chosen for every single decision tree generation.

For each of the binary trees in the RF, the building process is conducted on a subset of the attributes. The building process consists in building nodes from the top (root node) to the bottom (leaves nodes) and will stop when the stop criterion is satisfied, which is the same with the training procedure that has been introduced in the previous section.

Gini index (Gini impurity) is used to measure how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. The mathematical form of Gini index is as follows:

$$Gini_i = 1 - \sum_{j=0}^{2} \left( \frac{R_{ij}}{R} \right)^2$$

$$Gini_{total} = \sum_{i=0}^{2} \left( \frac{R_i}{R} \right) \cdot Gini_i$$

where $R_i$ is the number of records in partition $i$, among which $R_{ij}$ records bear the class label $j$; $R$ represents the records in the current node. The Gini index $Gini_i$ measures the impurity of the $i$th partition; the total Gini index $Gini_{total}$ measures the average impurity of all the partitions. We assume there are only two distinct values of classIDs, thus there are only two partitions in the equation. The average Gini index can be used to measure the fitness of a partition of the dataset for the decision tree made by a split point. The smaller the $Gini_{total}$, the better the split point.

The Gini unit architecture proposed in work [1] is used to calculate the total Gini index, in which the calculation of total Gini index is changed to:

$$Gini_{total}' = \frac{R_{00} \cdot R_{01}}{R_{00} + R_{01}} + \frac{R_{10} \cdot R_{11}}{R_{10} + R_{11}}$$

For simplicity, the constant value $R$ is omitted to save computation. Another thing to mention is that the multiplication is also unnecessary, because if the calculation of the Gini index is in the order of increasing value of the attribute, either $R_{00}$ or $R_{01}$ will increase by 1 every time calculation is conducted. By storing the previous value of $R_{00} \cdot R_{01}$, the next $R_{00} \cdot R_{01}$ can be simply calculated by adding either $R_{00}$ or $R_{01}$ to the stored value. Similar
III. Random Forest Training with Subsampling Method

Considering the large amount of data need to be processed, we propose the subsampling method for RF on FPGA to improve the training speed. The subsampling method is used in data preprocessing step. For the usual bootstrap aggregating (bagging) algorithm for sampling, it iterates over the whole dataset. The time complexity for this global comparison is high. Instead, our proposed idea is to randomly select y subdatasets on the sorted dataset distinct by attributes with the sampling ratio x (10% to 15%). The data preprocessing step on software is necessary. Each attribute dataset is sorted according to the value from small to large. Considering the performance of classification, we set the subsampling ratio to 10% which is a moderate value to get the well tradeoff between accuracy and speed. Whether the ratio is too high or too low may cause the drop of classification accuracy. For each tree construction, an essential part in our proposal is to calculate k estimated average Gini index on the subsampling dataset. For each tree node, by searching around the estimated result in the whole dataset on both direction, the smallest Gini index is found, and the corresponding attribute is decided to be the best splitter for the current node.

The original design assigns these two cases with label: “0” and “1”. Case 0 means the next search object is a upside search instance with the classID of 0. Case 1 means the next search object is a upside search instance with the classID of 1. We extended the Gini unit by assign the value “-0” and “-1” for the downside search. Case -0 means the next search object is a downside search instance with the classID of 0. Case -1 means the next search object is a downside search instance with the classID of 1. The detail of the calculation is shown in Fig.5. This double-sided search is used in the local search step to compensate the accuracy loss.

An example of the calculation for the next four inputs is shown in Fig.6. We suppose that there are four Gini unit used. The calculation of each Gini unit is shown in the right. Each result is compared with the former one which is stored in the register. The calculation stops until no smaller results are found, as the dotted circle for Gini unit 3 and Gini unit 4. Then, the best split of the current node is found.

For the training on the subsampling dataset and the local search process, we need to sort the result output by the Gini
module. The structure of the sorting cell is shown in Fig. 7. For each new input value, it is put into all sorting cells and compared with the former one; if the value is smaller, it is pushed out the former one and stored the new value in; if the new value is larger or equal to the original stored value, it is pushed out. We compare the values cell by cell until the local search is finished.

IV. Experiment and Result

The original method and the subsampling method is both implemented in Vivado HLS v2016. The device family is Xilinx Kintex7; target device is xc7k160tfbg484-2. The data preprocessing step is done in software using Python in Anaconda. Two datasets are used in both training and testing. The first one is a self-generated dataset by Sklearn named Toy, with the number of instances of 100,000. The second one is UCI Occupancy Detection Data Set, with the number of instances of 20,560. The data of UCI Occupancy dataset are collected according to the realistic scene of the sensor, which is experimental data used for binary classification (room occupancy) from Temperature, Humidity, Light and CO2. Ground-truth occupancy was obtained from time stamped pictures that were taken every minute. The detail of the dataset configuration is shown in the above table.

In common sense, the training speed on FPGA can always outperform the software implementation. Hence, we concentrate on how much speedup can be achieved by using subsampling method on hardware compared with the original one on hardware. We set 6 trees in the RF. The pipelined method is also inserted into the loop to reduce latency.

![Fig. 7. Sorting module for sequential input](image)

**TABLE II.**
Toy and UCI Occupancy Dataset Configuration

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#whole data</th>
<th>#subsampling</th>
<th>#testing data</th>
<th>#attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toy</td>
<td>100,000</td>
<td>10,000</td>
<td>100,000</td>
<td>6</td>
</tr>
<tr>
<td>UCI</td>
<td>20560</td>
<td>2056</td>
<td>Test1:2665</td>
<td>Test2:9752</td>
</tr>
</tbody>
</table>

**TABLE III.**
Hardware Utilization Analysis

<table>
<thead>
<tr>
<th>RF training method</th>
<th>( f_{\text{max}} ) (MHz)</th>
<th>Throughput (Gbps)</th>
<th>FF</th>
<th>LUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-pipelined</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF-om</td>
<td>471</td>
<td>124.6</td>
<td>3.99</td>
<td>2114</td>
</tr>
<tr>
<td>RF-sm</td>
<td>543</td>
<td>104.5</td>
<td>3.34</td>
<td>2490</td>
</tr>
<tr>
<td>pipelined</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF-om</td>
<td>134</td>
<td>127.9</td>
<td>4.09</td>
<td>5551</td>
</tr>
<tr>
<td>RF-sm</td>
<td>135</td>
<td>138.5</td>
<td>4.43</td>
<td>5419</td>
</tr>
</tbody>
</table>

**TABLE IV.**
Speedup Results of Non-pipelined and Pipelined Implementation

<table>
<thead>
<tr>
<th>RF training method</th>
<th>call_num</th>
<th>non-pipelined</th>
<th>pipelined</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(<em>L</em>)</td>
<td>Speed up</td>
<td>(<em>L</em>)</td>
</tr>
<tr>
<td>Toy Dataset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF-om</td>
<td>606,623</td>
<td>471</td>
<td>8.94x</td>
</tr>
<tr>
<td>RF-sm</td>
<td>58,876</td>
<td>543</td>
<td>7.45x</td>
</tr>
<tr>
<td>UCI Dataset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF-om</td>
<td>9967</td>
<td>471</td>
<td>8.94</td>
</tr>
<tr>
<td>RF-sm</td>
<td>1160</td>
<td>543</td>
<td>7.45</td>
</tr>
</tbody>
</table>

*\(L\): latency; * RF-om: original method; * RF-sm: subsampling method

![Fig. 8. Subsampling method speedup v.s. the original hardware implementation](image)
Fig. 9. Accuracy evaluation

The accuracy is calculated by putting the testing data into the classifier generated in the training process. The accuracy comparison is shown in Fig.9. We evaluate the Toy dataset for once with the whole dataset and UCI dataset with part of the dataset twice. As the results shows, the average accuracy can maintain at around 90% or even higher varied by the different training dataset used.

Compare with the original method which uses the whole UCI dataset for classifier training, we can see the performance of our method is almost the same. For the Toy dataset testing with all data, we can see the accuracy only drops about 4%. This accuracy drop is in exchange for 10.23x of speedup, which is worth it. For common subsampling method, the accuracy is always low because of the training data is not comprehensive. The above mentioned local search step is the idea to compensate the demerit of the subsampling method. Depending on different datasets, the parameter may create an influence on both accuracy and speedup.

V. Conclusion

In the paper, the subsampling method for RF training process acceleration on FPGA is proposed. The implementation has an average accuracy of 90% with a large dataset containing instances up to 100,000. The implementation result of the test bench shows a satisfactory performance with a small amount of hardware resource utilization under the constraint of maximum 6 attributes for different datasets. The speedup compared with the normal FPGA implementation without subsampling method can be 10.23x faster, which outperforms the previous work. The proposed method can reach an ideal acceleration speed while maintaining a satisfactory accuracy. Our future work is to enhance the power efficiency in hardware to attain higher throughput and to evaluate the FPGA performance with different number of trees in the random forest.

References