2017 IEEE 47th International Symposium on Multiple-Valued Logic

A Random Forest using a Multi-valued Decision Diagram on an FPGA

Hiroki Nakahara, Tokyo Institute of Technology, Japan
Akira Jinguji, Tokyo Institute of Technology, Japan
Simpei Sato, Tokyo Institute of Technology, Japan
Tsutomu Sasao, Meiji University, Japan

***Abstract*—A random forest (RF) is a kind of an ensemble machine learning algorithm used for a classification and a regression. It consists of multiple decision trees that are built from randomly sampled data. The RF has a simple, fast learning, and identification capability compared with other machine learning algorithms. It is widely used for various recognition systems. The conventional RF consisted of binary decision trees (BDTs), while in this paper, we used a multi-valued decision diagrams (MDDs). In the MDD, each variable appears only once on a path, however, in the BDT, some variable may appear multiple times. Since the path length is short in the MDD, it can be evaluated at a high speed. The disadvantage is that the number of nodes in the MDD increases with $O(2^N)$, where N denotes the number of input variables. Fortunately, random forests encourage to use the small number of N for each tree in order to avoid over fitting. Therefore, in several data sets used in the experimental, the number of nodes did not increase even if the MDD was used. To reduce the development time, the Altera SDK for OpenCL (AOCL), a kind of a high-level synthesis tool, was used. To accelerate the RF classification using the AOCL, we propose the fully pipelined architecture to increase the memory bandwidth using on-chip memories on the FPGA. Also, we apply optimal precision fixed point representation instead of 32 bit floating point one. We compared the performance with the CPU and the GPU implementations. As for the LPS (lookups per second), the FPGA realization was 10.7 times faster than the GPU one, and it was 14.0 times faster than the CPU one. As for the LPS per power consumption, the FPGA realization was 61.3 times better than the GPU one, and it was 12.1 times better than the CPU one.**

I. INTRODUCTION

A. Acceleration of the Random Forest (RF)

A decision tree (DT) based method is a popular method for various machine learning tasks. When a tree is grown very deeply to learn highly irregular patterns, it overfits training sets. In that case, it has low bias, while it has very high variance. Since the DT partitions the data set with a single feature variable, it often misclassifies out-lier labels. A random forest (RF) based method [5] is an ensemble learning method for a classification and a regression, and it consists of multiple binary decision trees (BDTs). For learning, each decision tree is built by different (randomized) samples from the same training set, with the goal of reducing the variance, in order to reduce the variance. Since the RF uses training feature variables selected at randomly sampled, BDTs with low correlation are built. As a result, it improves accuracy and versatility compared with a single DT based classification.

The RF is widely used for a classification. For example, they are used for a key point matching [19], a segmentation [3], a pedestrian detection [13], [10], a human pose estimation [23], a face direction estimation [11], and an IP address search for the Internet [15]. These applications are demanded to be recognized in real time. However, since the classification speed with the CPU is too slow, the hardware acceleration is necessary. Also, since it is often used in embedded systems, a low-power consumption is desired. However, a single instruction multiple data (SIMD) architecture, typified by GPU, is unsuitable for the RF with three reasons:

1. Higher accuracy precision:
   To evaluate BDTs in the RF, each node in the BDT can be evaluated by an *if-then-else* statements. A conditional expression in the *if-then-else* statement compares an input value with a constant value, which is represented by a floating point representation. Although the GPU supports a double precision floating point, such highly precision is not required for the RF classification. Therefore, a high-precision arithmetic circuit is inefficient both the amount of hardware and power consumption.
2. Uniformly processing (CUDA) cores:
   The GPU runs the SIMD operations, that is, having a large amount of uniformly processing core. These cores are specialized in data parallel computation. However, the RF consists of BDTs with a different size, that causes an unbalanced computation. Since a warp divergent is frequently occur, computation time would be bound by the BDT which has a longest path.
3. Higher cost for the all-to-all communication:
   The GPU can be performed at a relatively high speed communication between near processing cores with the same local memory, while its communication penalty is large for the all-to-all communication. Since a RF requires the whole of the majority detection after the evaluation of all the BDTs, the all-to-all communication would always occur.

B. FPGA Realizations of the RF

Since the GPUs have above demerits, the FPGAs are suitable to accelerate the RF. By using the FPGA, it is possible to configure a dedicated all-to-all communication circuit, heterogeneous cores for different size of a BDT, and an appropriate variable bit length circuit. Becker et al. used BDTs to accelerate object tracking. They focused on heavily on parallelizing the classification, and converted the input data into a dedicated representation in the FPGA [4]. Essen et

2378-2226/17 $31.00 © 2017 IEEE
DOI 10.1109/ISMVL.2017.40

IEEE computer society

al. showed a pipelined architecture and a single-instruction multiple thread (SIMT) algorithm for the RF on FPGA. Also, they compared the FPGA based implementation with the multi-core CPU and the GPU [12]. Oberg et al. implemented the RF on the FPGA with the Kinect depth-image sensor for the Forest Fire pixel classification algorithm [17]. However, the conventional realizations are designed by the RTL-level description. Compared with the software-based design, it takes an enormous amount of development time [1]. As for the RF design, since its structure is completely different for each dataset, it is not practical to tune the architecture by the RTL-level description. We have proposed a fast random forest which is suitable for the Altera SDK for OpenCL [20]. However, more acceleration is necessary to keep up with the real time recognition for an embedded system.

C. Contributions of the Paper

In the paper, we propose the multi-valued decision diagram (MDD) based classification instead of the BDT based one, which is a conventional data structure. Typically, in the MDD based one, since its path includes a variable at a time, it is faster than the BDT based one. Contributions of the paper are as follows:

1 We proposed an MDD based RF, while the conventional RF consisted of BDTs. In the MDD, each variable appears only once on a path, however, in the BDT, the same variable may appear several times. Since the path length is short for the MDD, it can be evaluated at a high speed. The disadvantage is that the number of nodes in the MDD tends to increase with $O(2^N)$, where N denotes the number of variables. Fortunately, random forests encourage to use the small N for each tree in order to avoid over fitting. Therefore, in several data sets used in the experimental, the number of nodes did not increase even if the MDD was used.

2 We compared the performance with the software based realization, such as the CPU and the GPU. Compared with the GPU realization, as for the LPS, the FPGA realization was 10.7 times faster than the GPU one, and it was 14.0 times faster than the CPU one. As for the LPS per power consumption, the FPGA realization was 61.3 times better than the GPU one, and it was 12.1 times better than the CPU one.

3 We opened the Python based code generation flow (RF2AOC).

The rest of the paper is organized as follows: Chapter 2 introduces the Altera SDK for OpenCL, and compares with the GPU based realization; Chapter 3 introduces the random forest (RF); Chapter 4 introduces the multi-valued decision diagram (MDD); Chapter 5 proposes an acceleration for the RF using MDDs, and shows its tool flow; Chapter 6 shows the experimental results; and Chapter 7 concludes the paper.

II. Altera SDK for OpenCL

The OpenCL is a framework for a parallel programming intending to the CPUs and the GPUs. With the OpenCL, a programmer can easily realize a parallel computation based on task parallelism and data parallelism. In recent years, the Altera Corp. has developed the Altera SDK for OpenCL for the FPGA development environment. However, when we directly applied the GPU programming model to the Altera SDK for OpenCL, since the target architecture are different, it is hard to accelerate the application.

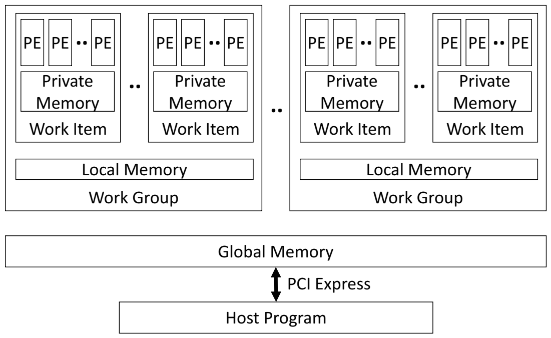


Fig. 1. Memory model for the OpenCL.

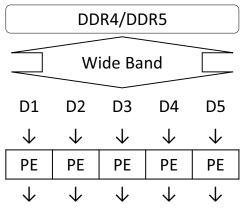


Fig. 2. Architecture model for the OpenCL on a GPU.

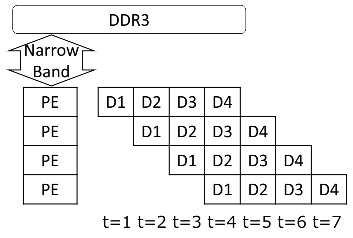


Fig. 3. Architecture model on an FPGA.

Here, we explain the programming model for the Altera SDK for OpenCL. Fig. 1 shows the memory model for the standard OpenCL [18]. The global and constant memories are accessible from all the work-items. The global memory is possible to read and write, while the constant one is read-only. The local memory is a shared one by a work-group, and the private memory is occupied by each work-item. Fig. 2 shows an architecture model for the GPU, while Fig. 3 shows that for the FPGA. For the GPUs for the OpenCL, it runs the threads in the work-group to the data parallel model by using a large number of CUDA cores and wide band data transfer memory such as DDR5 off-chip memories. On the other hand, for the FPGA, since data communication bandwidth, that is, that for off-chip memories are narrow, it tends to configure the pipeline model in the work-group. When the pipeline stall is free, communications with the off-chip memory are only input and output of the pipeline. Therefore, the FPGA can perform a high-throughput operation even if it has a narrow band to the off-chip memories. Fortunately, since the FPGA has more on-chip memories than the GPU, data transfer between the pipeline stages is often done on the FPGA. The Altera SDK for OpenCL supplies a channel to make an on-chip communication between pipeline stages. Furthermore, another advantage of the FPGA has an ability to realize a customized pipeline stage (in other words, it can realize a heterogeneous core) in parallel. When the latency of each parallel operation is different, its computation time for the GPU would be bound by the longest one. On the other hand, since the hardware resources in the FPGA are appropriately distributed, it is possible to realize a

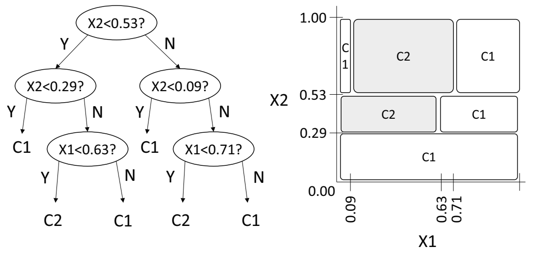


Fig. 4. Example of a binary decision tree (BDT).

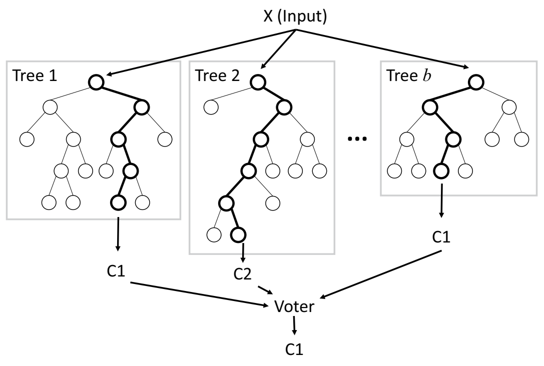


Fig. 5. Example of a random forest (RF).

heterogeneous parallel architecture with uniform latency.

Another feature of the Altera SDK for OpenCL is that it supplies the board support package (BSP) for recommended FPGA boards. The BSP prepares the IP cores for the PCI express and external memories to bridge host program and the kernel program. For the conventional FPGA design, since the programmer designed them in the RTL-level description, it could not respond to frequent changes in long term design. By using the Altera SDK for OpenCL, it can generate the configuration data that automatically connects with the user program. Therefore, since it is possible to reduce design time remarkably, the programmer can concentrate on tuning architectures.

III. Random Forest (RF)

A BDT tree based method is a popular for various machine learning tasks. When a tree is grown very deep to learn highly irregular patterns, it overfits training sets. In that case, it has low bias, while it has very high variance. Fig. 4 shows an example of a BDT which classifies a data set. In Fig. 4, X_i denotes a feature variable for the dataset, and C_i denotes a label. Since the BDT partitions the data set using a single feature variable, it often misclassifies out-lier labels.

A random forest (RF) based method is an ensemble learning method for a classification and a regression, and it consists of multiple BDTs. At learning, each BDT is built by different (randomized) sub-sampling data from the same training set in order to reduce the variance. Fig. 5 shows an example of the RF, which consists of b BDTs and a voter. First, BDTs branch corresponding to given feature variables. Then, they output the matched label. Next, the voter performs a majority decision of the labels from BDTs. Finally, it detects the most frequent label as a classification result. Since the RF uses training feature variables selected at random sampled, BDTs with low correlation are built. As a result, it improves accuracy and versatility.

Followings are an algorithm to built the RF from a given dataset.

1. Randomly selects b sub-samples from given dataset (boot strap sampling)
2. Learns b BDTs from b sub-samples
3. Creates a node to reach the specified number of nodes N_{min}

3.1. Selects r samples at a random

3.2. Computes a constant values for *if-then-else* statements in a node, which classifies sub sampling data

4. Terminate

The advantages of the RF are shown as follows [5]:

1. Classification accuracy is high, and it operates correctly even if the feature variables are from several hundreds to thousands
2. It is possible to estimate the importance of the feature variables for each label variable
3. It effectively works with dataset even if it lacks several feature variables
4. The number of individual error are maintained even in unbalanced dataset

On the other hand, the disadvantages are follows:

1. Too deep BDTs fall into over fitting
2. Classification accuracy is low with a small number of learning data

In addition, classification accuracy is greatly affected by hyper parameters. By using a grid search algorithm and encourage parameters [1], the RF can be built with a relatively appropriate hyper parameters.

IV. Multi-valued Decision Diagram

A **binary decision diagram (BDD)** [6], [16] is obtained by applying **Shannon expansions** repeatedly to a logic function f. Each non-terminal node labeled with a variable x_i has two outgoing edges which indicate nodes representing cofactors of f with respect to x_i. **A multi-terminal BDD (MTBDD)** [7] is an extension of a BDD and represents an integer-valued function. In the MTBDD, the terminal nodes are labeled by integers.

Let $X = (X_1, X_2, \ldots, X_u)$ be a partition of the input variables, and $|X_i|$ be the number of binary variables in X_i. X_i is called **a super variable**. When the Shannon expansions are performed with respect to super variables X_i, where $|X_i| = k$, all the non-terminal nodes have 2^k edges. In this case, we have a **multi-valued multi-terminal decision diagram (MTMDD(k))** [14]. Note that, an MTMDD(1) corresponds to an MTBDD. In the paper, we simply call the MDD (k) as the MTMDD (k). **The width of the MDD (k) at the height** i is the number of edges crossing the section

[1]For example, when the number of BDTs is N, its depth is encouraged to $\sqrt{N}$ [5].

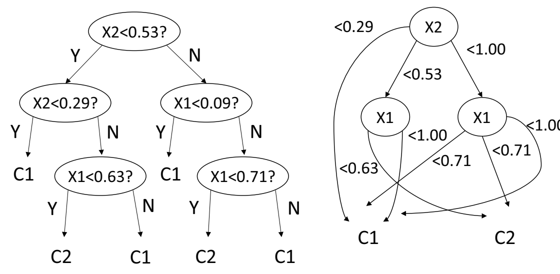
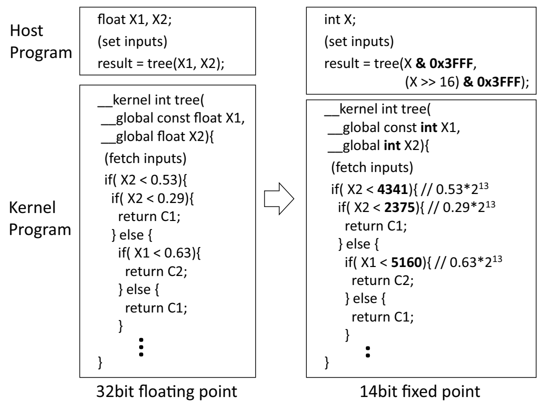


Fig. 6. Comparison the BDT with the MDD.



```
float X1, X2;
(set inputs)
result = tree(X1, X2);
```

```
__kernel int tree(
 __global const float X1,
 __global float X2){
 (fetch inputs)
 if( X2 < 0.53){
   if( X2 < 0.29){
     return C1;
   } else {
     if( X1 < 0.63){
       return C2;
     } else {
       return C1;
     }
       ⋮
}
```

```
int X;
(set inputs)
result = tree(X & 0x3FFF,
              (X >> 16) & 0x3FFF);
```

```
__kernel int tree(
 __global const int X1,
 __global int X2){
 (fetch inputs)
 if( X2 < 4341){ // 0.53*2^13
   if( X2 < 2375){ // 0.29*2^13
     return C1;
   } else {
     if( X1 < 5160){ // 0.63*2^13
       return C2;
     } else {
       return C1;
     }
       ⋮
}
```

Fig. 7. Example of a fixed point representation.

of the MDD (k) between super variables X_{i+1} and X_i, and denoted by μ_i, where the edges incident to the same node are counted as one.

Definition 4.1: In the MDD, a sequence of edges and non-terminal nodes leading from the root node to a terminal node is a **path**.

Example 4.1: Fig. 6 shows an example of the MDD corresponding to the BDT shown in Fig. 4. ■

In this paper, we propose the dedicated hardware to evaluate multiple MDDs on the FPGA. Since it realizes by the fully pipeline circuit, if the evaluation time for all the MDD nodes are the same, then it is proportional to the **longest path length (LPL)**. We will show that the LPL for the MDD based one is shorter than that for the BDT based one.

V. Acceleration Techniques using Altera SDK for OpenCL

A. Fixed Point Representation

As shown in Figs. 2 and 3, bandwidth between the off-chip memory and the kernel on the FPGA becomes a bottleneck. Each node in the BDT can be expressed by *if-then-else* statements. For many of random forest software libraries, a conditional expression in the *if* statement is a comparison of a feature variable with a constant value which is represented by a 32-bit floating point representation. In this paper, we use an n-bit signed fixed point representation instead of 32-bit floating point one. Fig. 7 shows the pseudo codes for the BDT using a floating point and 14-bit signed fixed point representation. Note that, in the right side pseudo code, the most signification

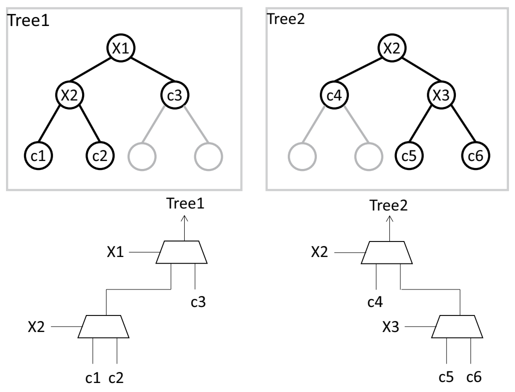
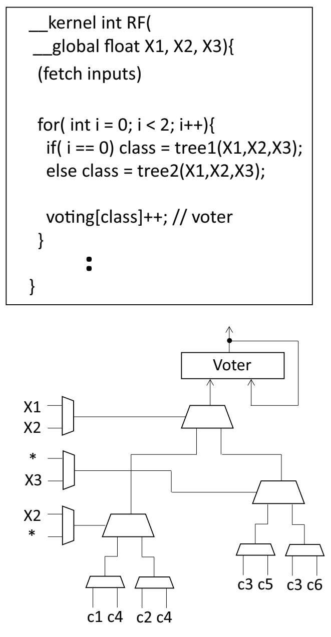


Fig. 8. Example of a multiplexer tree realization.

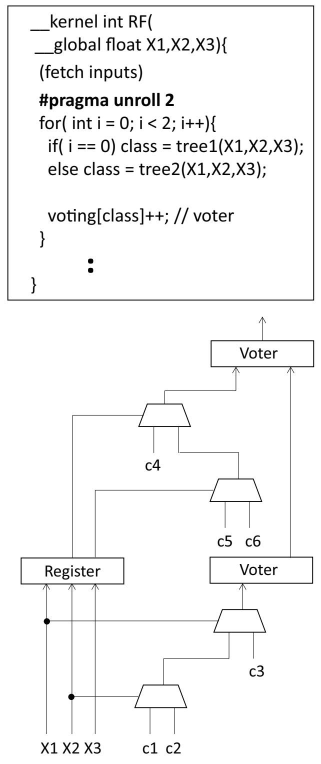


```
__kernel int RF(
 __global float X1, X2, X3){
 (fetch inputs)

 for( int i = 0; i < 2; i++){
   if( i == 0) class = tree1(X1,X2,X3);
   else class = tree2(X1,X2,X3);

   voting[class]++; // voter
 }
       ⋮
}
```

Fig. 9. Sequential realization.



```
__kernel int RF(
 __global float X1,X2,X3){
 (fetch inputs)
 #pragma unroll 2
 for( int i = 0; i < 2; i++){
   if( i == 0) class = tree1(X1,X2,X3);
   else class = tree2(X1,X2,X3);

   voting[class]++; // voter
 }
       ⋮
}
```

Fig. 10. Fully pipeline realization.

bit (MSB) represents a sign, while latter bits represent a number. For the OpenCL standard, although it supports only 8, 16, 32, and 64 bit integers, the Altera SDK for OpenCL supports a variable bit integers by using an appropriate mask. Depending on the bit width, a fixed point representation lacks accuracy compared with a floating point one. Thus, a fixed-point representation may cause misclassification. However, since it compresses the bandwidth of the off-chip memory to $\frac{n}{32}$, it can accelerate the classification. Furthermore, since multiplexer trees for a fixed point are simplified, they are faster and smaller than floating point based multiplexers.

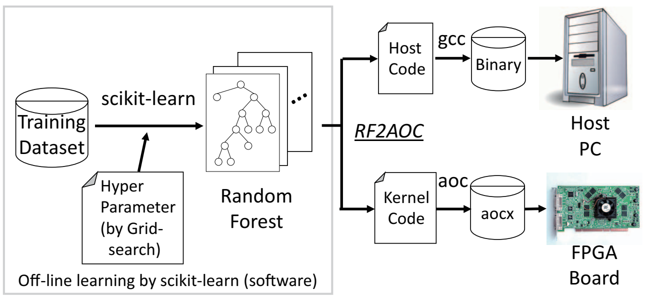


Fig. 11. Proposed tool flow.

TABLE I. DATASET USED IN THE EXPERIMENT.

Dataset				Hyper Parameters		
Name	#Rules	#feat.	#Class	#Trees	Depth	Max#feat.
Dermatology	366	33	6	30	5	7
Contraceptive Method	1473	9	3	25	5	5
Glass Identification	214	10	7	30	7	2
Hayes-Roth	160	5	3	15	7	4
Hepatitis	155	19	2	30	5	3
Ionosphere	351	34	2	25	15	10
Iris	150	4	3	50	20	2

B. Pipeline Stages by a Loop Unrolling

For the GPU, as shown in Fig. 2, its performance is degraded when all-to-all communications occur. Therefore, in our implementation, we connect BDTs in series to form a deep pipeline with on-chip memory. Also, we realize the voter by the pipeline circuit. Fig. 8 shows an example of BDTs, and their hardware realization by a multiplexer tree. When the BDTs is written by a *for* statement, as shown in Fig. 9, the Altera SDK for OpenCL sequentially traces BDTs by a shred multiplexer circuit. On the other hand, we can increase the throughput by using an *#pragma unroll* which expands a sequential circuit to a pipelined one. Fig. 10 shows a pipeline circuit with an *#pragma unroll*. In the pipeline circuit, registers and voters are inserted between the multiplexer trees. In that case, the number of registers and the memories tend to be increased, and they are realized by the on-chip memories. Since communication with the off-chip memory does not occur, it can accelerate throughput.

C. Proposed Tool Flow

Fig. 11 shows the proposed tool flow. First, we use the scikit-learn software [22] to learn the RF from given dataset. Note that, we find the optimum hyper parameter set by a grid-search algorithm. Then, we generate the host code and the kernel code for the Altera SDK for OpenCL by using the *RF2AOC* which is written by Python script. The generated codes are converted into the bit stream (aocx) file by using the Altera SDK for OpenCL. Since the proposed tool flow automatically generates the bit stream from given dataset, we can concentrate the parameter tuning to accelerate the RF.

VI. Experimental Results

A. Implementation Environment

We implemented the MDD based random forest for the UC Irvine machine learning repository [24] to the Teraisc Inc. DE5-NET FPGA board, which has an Altera Corp. Stratix V A7 FPGA, two DDR3 SO-DIMMs, and the PCI express Gen3. For the host PC, we used the Intel's Xeon (R) E5607 Processor (2.26 GHz, 4 cores) with 32 GB DDR3 off-chip memory, and CentOS 6.4 (64 bit version) operating system. Table I shows the used dataset and parameters for them. Note that, to find the optimum parameter set, we used a grid search algorithm which is available in scikit-learn.

TABLE II. COMPARISON THE MDD BASED WITH THE BDT BASED.

	BDT		MDD			Ratio	
Name	Path Len.	#Nodes	Max. Path	Path Len.	#Nodes	Path	Nodes
Dermatology	720	676	15	322	118336	2.2	175.1
Contraceptive Method	600	1055	9	198	7360	3.0	7.0
Glass Identification	952	1260	10	268	17204	3.6	13.7
Hayes-Roth	480	577	5	73	448	6.6	0.8
Hepatitis	720	1040	15	357	145664	2.0	140.1
Ionosphere	1196	1077	20	381	671744	3.1	623.7
Iris	1056	777	4	199	517	5.3	0.7

TABLE III. COMPARISON WITH THE CPU AND THE GPU.

	GPU@86W GeForce Titan		CPU@13W Xeon (R) E5607		FPGA@15W Stratix V A7	
Name	LPS	LPS/W	LPS	LPS/W	LPS	LPS/W
Dermatology	336.2	3.9	211.6	16.3	3221.2	214.7
Contraceptive Method	521.9	6.1	286.4	22.0	10924.3	728.3
Glass Identification	726.7	8.5	587.5	45.2	6442.3	429.5
Hayes-Roth	1512.9	17.6	1165.5	89.7	12884.6	859.0
Hepatitis	739.1	8.6	662.7	51.0	8209.9	547.3
Ionosphere	821.0	9.5	595.9	45.8	9663.5	644.2
Iris	446.6	5.2	436.7	33.6	4831.7	322.1

B. Misclassification Rate for Fixed Point Representation

We changed the number of bits n for an n-bit fixed point representation in the conditional constant of *if-then-else* statements of the MDD. Then, we obtained misclassification rate compared with a 32-bit floating point representation. Fig. 12 compares of misclassification rate for n-bit fixed point representation. As shown in Fig. 12, as for the small number of n, misclassification rate tends to be large. Thus, for the implementation, we choose appropriate bits for a fixed point representation.

C. Comparison of the MDD based with the BDT based

Table II compares the BDT with the MDD. In the MDD, since each variable appears only once in a path, the path length is from 2.0 to 6.6 times shorter than that of BDTs. Since the number of nodes of the BDT is $O(N)$, the number of nodes for the MDD (k) increases with $O(k^N)$. Thus, a data set with a longer path length (*Dermatology, Hepatitis, and Ionosphere*) has increased the number of nodes by 140 to 623 times. On the other hand, the number of nodes decreased in the data set (*Iris, and Hayes Roth*) with shorter path length. From these results, to generate a random forest using the MDD, we should generate many MDDs with shorter path lengths.

D. Comparison with Other Platforms

As for the lookups per second (LPS) and the power consumption efficiency (LPS/W), we compared the FPGA with the CPU and the GPU. Fig. 13 shows the execution flow used in the experiment. As for the CPU platform, we used the Intel's Xeon (R) E5607 Processor (2.26GHz, 4 cores) with 32GB DDR3 off-chip memory, and Ubuntu 14.04 LTS (64 bit version) operating system. To generate the executable code,

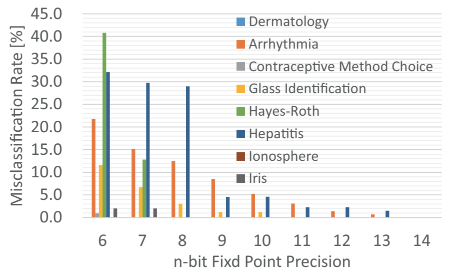


Fig. 12. Misclassification rate for n-bit fixed point representation.

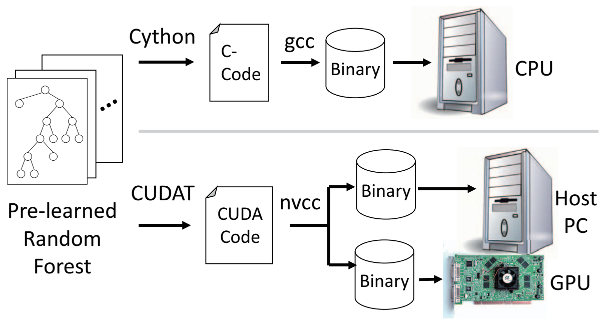


Fig. 13. Execution flow for the CPU and the GPU.

first, we generated the RF by the scikit-learn with the same parameters. Then, we converted the RF to C-codes by Cython [8], and compiled to executable code by gcc compiler. As for the GPU platform, we used the nVidia Geforce Titan (876 MHz, 2,496 CUDA cores, and 6GB DDR5 off-chip memory) with the same processor and the main memory running on the Ubuntu 14.04 LTS. To generate the executable code, first, we generated the RF by the scikit-learn, then, we used the CUDA Tree (CUDAT) [9] to generate the executable code. To measure the LPS, we used 10,000 random test vectors, while to measure the power consumption excluding the idle power, we inserted the power measurement instrument between the host PC and the power source. As for the FPGA realization, from above experiments, we set appropriate number of bits and unrolls.

Table III compares the performance with the CPU and the GPU realizations. Compared with the GPU realization, as for the LPS, the FPGA realization was 10.7 times faster than the GPU one, and it was 14.0 times faster than the CPU one. As for the LPS per power consumption, the FPGA realization was 61.3 times better than the GPU one, and it was 12.1 times better than the CPU one.

VII. Conclusion

To reduce the development time and to archive the high-speed classification, we used MDD based random forest with the Altera SDK for OpenCL. To accelerate the RF classification, we used the fully pipelined architecture to increase the memory bandwidth through on-chip memories on the FPGA. Also, we applied optimized bit fixed point representation instead of 32 bit floating point one. In the MDD, since each variable appears only once in a path, the path length was from 2.0 to 6.6 times shorter than the BDTs. To generate a random forest using the MDD, we should generate many MDDs with shorter path length. We compared the performance with the CPU and the GPU implementations, As for the LPS (lookups per second), the FPGA realization was 10.7 times faster than the GPU one, and it was 14.0 times faster than the CPU one. As for the LPS per power consumption, the FPGA realization was 61.3 times better than the GPU one, and it was 12.1 times better than the CPU one.

VIII. Acknowledgments

This research is supported in part by the Grants in Aid for Scientific Research of JSPS, an Accelerated Innovation Research Initiative Turning Top Science and Ideas into High-Impact Values program (ACCEL) of JST, and Altera Corp. University Program.

References

[1] M. S. Abdelfattah, A. Hagiescu and D. Singh, “Gzip on a chip: high performance lossless data compression on FPGAs using OpenCL,” *Int’l Workshop on OpenCL 2013 and 2014 (IWOCL)*, No. 4, 2014.

[2] Altera Corp., “Altera SDK for OpenCL,” http://www.altera.com/

[3] Y. Amit and D. Geman, “Shape quantization and recognition with randomized trees,” *Neural Computation*, Vol. 9, 1996, pp. 1545-1588.

[4] T. Becker, Q. Liu, W. Luk, G. Nebehay and R. Pflugfelder, “Hardware-accelerated object racking,” *Computer Vision on Low-Power Reconfigurable Architectures Workshop,* 2011.

[5] L. Breiman, “Random forests,” *Machine learning*, Vol. 45, No. 1, 2001, pp. 5-32.

[6] R. E. Bryant, “Graph-based algorithms for boolean function manipulation,” *IEEE Trans. on Comput.*, Vol. C-35, No. 8, pp. 677-691, 1986.

[7] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, “Spectral transforms for large Boolean functions with applications to technology mapping,” *Design Automation Conference (DAC)*, pp. 54-60, 1993.

[8] “Cython: C-Extensions for Python,” http://cython.org/

[9] W. T. Lo, Y. S. Chang, R. K. Sheu, C. C. Chiu and S. M. Yuan, “CUDT: A CUDA based decision tree algorithm,” *The Scientific World Journal,* Vol. 2014, No. 745640, 2014, pp. 1-12.

[10] T. Danhang, Y. Liu and T. K. Kim, “Fast pedestrian detection by cascaded random forest with dominant orientation templates,” *BMVC*, 2012, pp. 58.1-58.11.

[11] M. Dantone, J. Gall, G. Fanelli and L. V. Gool, “Real-time facial feature detection using conditional regression forests,” *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 2578-2585.

[12] B. V. Essen, C. Macaraeg, M. Gokhale and R. Prenger, “Accelerating a Random Forest Classifier: Multi-Core, GP-GPU, or FPGA?,” it IEEE Annual Int’l Symp. on Field-Programmable Custom Computing Machines (FCCM), 2012, pp. 232-239.

[13] S. Hinterstoisser, V. Lepetit, S llic, P. Fua and N. Navab, “Dominant orientation templates for real-time detection of texture-less objects,” *CVPR*, 2010.

[14] T. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, “Multi-valued decision diagrams: Theory and applications,” *Multiple-Valued Logic: An International Journal*, Vol. 4, No. 1-2, pp. 9-62, 1998.

[15] H. Le, W. Jiang and V. K. Prasanna, “A SRAM-based architecture for Trie-based IP lookup using FPGA,” *FCCM*, 2008, pp. 33-42.

[16] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design: OBDD Foundations and Applications*, Springer, 1998.

[17] J. Oberg, K. Eguro, R. Bittner and A. Forin, “Random decision tree body part recognition using FPGAs,” *The Int’l Conf. on Field Programmable Logic and Applications (FPL)*, 2012, pp. 330-337.

[18] “The OpenCL specification, version 1.2,” http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf

[19] M. Ozuysal, M. Calonder, V. Lepetit and P. Fua, “Fast keypoint recognition using random ferns,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 32, No. 3, 2010, pp. 448-461.

[20] H. Nakahara, A. Jinguji, T. Fujii, and S. Sato, “An Acceleration of a Random Forest Classification using Altera SDK for OpenCL,” *Int’l Conf. on Field-Programmable Technology (FPT)*, (accepted for publication).

[21] R. Narayanan, D. Honbo, G. Memik, Al Choudhary and J. Zambreno, “An FPGA implementation of decision tree classification,” *Design Automation and Test Europe Conference (DATE)*, 2007, pp.1-6.

[22] “Scikit-leran: Machine Learning in Python,” http://scikit-learn.org/stable/

[23] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman and A. Blake, “Reali-time human pose recognition in parts from single depth images,” *CVPR*, Vol. 2, 2011, pp. 1297-1304.

[24] “UCI Machine Leraning Repository,” http://archive.ics.uci.edu/ml/datasets.html