An Energy Efficient Approach for C4.5 Algorithm using OpenCL Design Flow

Hai Peng[1], Xiaofan Zhang[2], Letian Huang[1]
[1] University of Electronic Science and Technology of China
[2] University of Illinois at Urbana-Champaign
huanglt@uestc.edu.cn

***Abstract*—C4.5 is an important data mining algorithm and has been widely applied in applications including face detection, character recognition and predictive analysis. Although C4.5 is highly accurate, its training process is time-consuming because the traditional algorithm has limited parallelism and frequent data hazards. In order to address these issues, we propose an energy efficient approach for C4.5 training with parallel search, low-latency memory accesses and a folded programming structure to achieve higher acceleration and energy efficiency. This proposed approach greatly improves the C4.5 training process by using a CPU-FPGA heterogeneous platform and OpenCL design flow. Three UCI data sets including spambase, Magic04, and MiniBooNE, are used to evaluate the performance of our method. Experimental results show that our approach has achieved 58X, 63X, 380X higher energy efficiency than the basic serial C4.5 algorithm (serial SPRINT) running on an Intel i7-3770K CPU and 10X, 1.6X, 12.7X higher energy efficiency than the existing parallel C4.5 algorithm (parallel SPRINT) running on the same CPU-FPGA heterogeneous platform. We also deliver 3X, 5X, 3X higher energy efficiency than the existing CUDA based parallel C4.5 (CUDT) in CPU-GPU heterogeneous platform when using these three datasets.**

Keywords—Heterogeneous System; OpenCL; Data Mining; Parallel Extension Method

I. INTRODUCTION

Decision trees are one of the most widely-used tools for data mining. By taking advantage of the recent evolutionary techniques in data extraction, enormous amounts of data can be produced and processed within a short period, enabling a machine to make accurate decisions. In recent years, enormous amount of data is being generated, extracted, and processed in large-scale clusters of computers (e.g. data centers, cloud computing platforms etc.). The sheer amount of data allows us to design and train machine learning algorithms that can deliver decisions with exceptional accuracy. However, these classification techniques are extremely computationally intensive, meaning that they are not appropriate for applications that require low energy consumption, such as embedded applications [1]. Hardware accelerators offer the potential to address this problem with significant speedup and energy efficiency [2, 3].

Recently, heterogeneous systems, which consist of CPUs and FPGAs, are of increasing interest, especially if both hardware architectures can be used efficiently, with each running various workloads. However, implementing algorithms with traditional design and verification methodologies on FPGAs (e.g. using HDL language such as Verilog, VHDL) is challenging and time-consuming. To mitigate this problem, progresses have been made to develop a portable, cross-platform high-level synthesis tools, which allow the computation-intensive task to be parallelized and optimized with high-level languages (e.g. C, C++). Open Computing Language (OpenCL) [4] is a widely supported open standard which can be applied not only in instruction processors such as CPUs and GPUs but also in programmable devices like FPGAs. By using OpenCL design flow, it is possible to reduce the design effort and time of FPGA-based solutions.

In terms of classification algorithms, the C4.5 algorithm is considered as one of the most representative algorithms in data mining [5, 6]. Since the computational cost of C4.5 is extremely intensive, it requires optimization efforts to achieve fast total turn-around time, which generally includes hardware accelerators such as GPUs and FPGAs. However, the high power consumption of GPU-based solutions hinder its use in energy-sensitive applications.

In this paper, to meet the demand of fast response time and low energy consumption of large-scale clusters of FPGAs (FPGA servers, FPGA clouds etc.); we present an energy efficient C4.5 algorithm with three main features: 1) parallel search: a breadth-first strategy with extended parallelism for rapid search in dataset; 2) low-latency memory access: an optimized index table strategy which uses a Ping-Pong strategy for fast data fetching and to eliminate race conditions; and 3) folded programming structure: a code folding strategy in OpenCL design flow for maximum performance under limited on-chip resources. To our knowledge, this is the first attempt to implement C4.5 training tree on the FPGA-based heterogeneous system with OpenCL design flow. Three data sets (spambase, Magic04 and MiniBooNE) are used to evaluate the proposed C4.5 implementation. Results show that the energy efficiency of our solution is 58X, 63X, 380X higher than the basic C4.5 algorithm (CPU version), 10X, 1.6X, 12.7X better than the parallel C4.5 (CPU-FPGA heterogeneous version) and 3X, 5X, 3X higher than the CUDA based parallel C4.5 (CPU-GPU heterogeneous version).

The remainder of this paper is organized as follows. In Section 2, we provide a brief background of the existing C4.5 algorithms and discuss related work. In Section 3, we explain the design challenges before presenting our solutions. Section 4

analyses our experimental results. Finally, we conclude this paper in Section 5.

II. BACKGROUND

A. C4.5 algorithms

C4.5 is one of the most widely deployed tree-based algorithms for data mining [7]. The input to C4.5 is a data set designed for training decision trees which consists of records with unique record IDs and different attributes.

During training, a decision tree classifier is built upon two phases [8, 9]: the growth phase and the pruning phase. Because the growth phase is much more computationally intensive, which requires more time and energy, our work focuses on this part.

Two stages of tree growth are shown in Figure 1. Phase 1 indicates the calculation of the best split attribute from a given data set. In this phase, first one calculates the class frequency of each attribute. After that, the best split point and IGR (Information Gain Ratio, which is used to evaluate the information value of attributes) of each attribute are determined based on the class frequency. After that, the best split attribute is decided according to IGRs and the decision tree can be generated with depth-first strategy as shown in phase 2. Based on the rules in phase 1, each node of the tree initializes its information and generates its child nodes with the best split point.

```
Algorithm :  Training Tree Construction of C4.5
Input : Dataset
Output : BestSplitAttribute
Phase 1 :   1: D=Dataset
            2: ComputeClassFrequency(D)
            3: for each attribute A in D do
                    ComputeBestSplitPoint(A)
                    ComputeIGR(A)
               end for
            4: BestSplitAttribute = AttributeWithMaxIGR
Input : Pointer to root node
Output : Decision Tree
Phase 2 :   1: If N is not root then
                    N = CreateNode()
               end If
            2: do phase 1
            3: Initialize(N, BestSplitAttribute)
            4: If N is leaf then
                    lchild = Null
                    rchild = Null
                    return N
               Else
                    do phase 2 with lchild
                    do phase 2 with rchild
                    return N
```

Fig. 1. Pseudocode of tree growth

Calculation of the best split point and the best split attribute in phase 1 are the two crucial parts in training decision trees. These have high computational requirements because the dataset needs to be scanned multiple times. Unfortunately, it requires a considerable amount of serial operations which result in long execution time during the tree growth phase.

While significant research has been placed into maximising accuracy of C4.5 [10, 11], execution time is still a major limitation. As a result, many acceleration schemes targeting the tree growth phase have been done since C4.5 was first proposed in 1986 [5]. This includes research on hardware accelerators for decision trees [12, 13]. In [14, 15], a general structure of the accelerator for running decision trees was proposed. However, this work did not study training, which is the most computationally intensive process, and only implemented a trained tree in hardware.

The first approach covering hardware acceleration scheme of decision tree training was presented in [16]. By using HDL design flow, the authors achieved a speedup of 5.58x on an FPGA as compared to software implementations. In [17], an accelerator implemented on a GPU achieved dramatic acceleration where it took only 1% of the time compared with the CPU based solution. Similarly, GPU-based implementations are widely accepted [18, 19, 20]. However, these GPU designs only focus on speedup. As mentioned earlier, energy efficiency should also be of interest.

B. Decision trees algorithm: SPRINT

SPRINT (Scalable Parallel Classifier for Data Mining) is a representative algorithm based on C4.5 for building parallel decision trees with considerable reduction of time and memory consumption [8, 9]. With its highly efficient data structure, called an attribute list [21], SPRINT reduces the access times to the data set and improves its overall performance. The attribute list in SPRINT is shown in Figure 2 where the left table is an attribute list of continuous attributes and the right table is an example of categorical attributes. This attribute list consists of three arrays: the attribute value (the first column), the class label (the second column), and the index of record in the last column. Because each attribute is independent, they can be sorted in one pass of the dataset, without extra sorting phases.

age	Class	Rid
18	High	1
23	Low	3
41	High	0
8	Low	4
65	High	3

Car type	Class	Rid
Truck	High	0
Family	Low	1
Sports	High	2
Family	Low	3
Truck	High	4

Fig. 2. Data structure of SPRINT

C. OpenCL architecture

OpenCL is a framework for parallel programming on heterogeneous platforms which is based on a run-time host library and C99 extensions for device programming adapted to support vectorized data types [4]. OpenCL offers a standard design flow which applies to a variety of devices. Figure 3 shows an FPGA-based heterogeneous system that supports OpenCL. This platform consists of two computational units: an x86 CPU as the master and an FPGA as the device. In terms of data exchange, the data flow towards devices is handled by the master through command queues. The data management of OpenCL relies on a relaxed memory consistency model and it

offers three different memories for designers: global, local and private memory. After taking into account the available computational resources, they can be organized and divided into groups called Computing Units (CU). In each CU, a large number of Processing Elements (PE) are deployed as the basic computational elements. In OpenCL, CUs are also called as work-groups, while PEs are known as work-items. Global memory on an FPGA is usually mapped as DIMMs which can be shared by all work-groups with the largest capacity and lowest I/O speed. Local memory is used as a cache for each work-group and it is visible to all work-items within the same work-group. Meanwhile, each work-item has its own private memory which is usually implemented as registers.

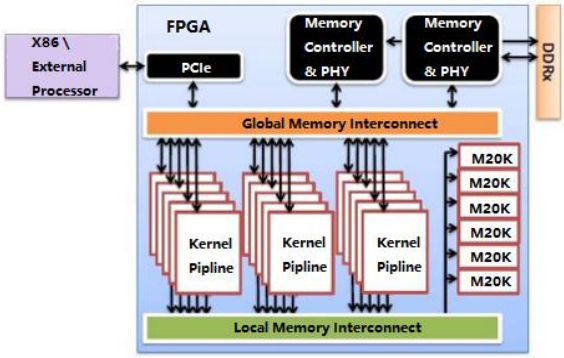

Fig. 3. OpenCL architecture

In this paper, we present an OpenCL-based heterogeneous system designed for the C4.5 algorithm. The energy consumption and development time of our design have been largely reduced while the execution speed is improved in comparison to GPU-based implementations and existing HDL designs.

III. Energy Efficient C4.5 Algorithm

Unlike the traditional development methods which use HDL for FPGA implementation, we design the C4.5 algorithm and map into the targeted CPU + FPGA heterogeneous system using OpenCL. The parallelism, resource utilization, and access violation are taken into consideration when we design OpenCL kernels. In this section, design challenges and our proposed solutions are also discussed.

A. Design challenges

Several challenges arise when implementing C4.5 on an FPGA for a given speedup and energy efficiency requirement. In particular, limited parallelism in C4.5 wastes the computation resources and slows down the speed during the tree growth phase. In this algorithm, huge demand for data comparisons and scan increase the computational workload and cause potential data hazards during memory access. In addition, limited resources on the FPGA and the gap between high-level descriptions in OpenCL and circuit-level structures of board level implementation are two further design challenges. In the following subsections, we discuss these design challenges in detail.

1) Low degree of parallelism in C4.5: A large portion of serial operations is adopted in the traditional tree growth algorithm, resulting in a relatively low degree of parallelism. This is because in C4.5 the depth-first strategy only allows one node to be processed; this forms a bottleneck during execution. Furthermore, serial searches of the data set slow down the turn around time, and there are no parallel schemes to find the best split attribute or the split point in a parallel way.

2) Huge demand for data comparison and scan: The tree growth phase includes a huge volume of data compare operations. It is necessary to scan the whole data set and conduct the same calculations multiple times before finally obtaining the results.

3) Potential data hazards in memory: As a result of frequent I/O operations, data hazards arise when parallel programs write data to the same location in memory. The use of traditional queue-based or lock based strategies is limited due to the available resources on the FPGA.

4) Limited resources on FPGA: Since OpenCL is based on high-level descriptions, it is nontrivial to map to circuit-level structures using hardware descriptions. This makes it harder to construct an efficient reuse scheme to maximise the use of limited resources on FPGAs. Resource utilization must be taken into account when designing OpenCL programs.

5) Cost of AOC commands: Although OpenCL provides many optimization commands (AOC) for FPGAs, to maximise performance, we still require careful use of optimization methods, to better tradeoff the available FPGA resources and achievable performance. For example, loop unrolling can reduce the execution time of loops, but it raises the DSP blocks utilization at the same time. Other optimizations such as vectorization and pipeline replication can enhance the parallelism, but they cause a higher demand for memory bits. As a result, it is necessary to take the most urgent resource and optimization costs into consideration before AOC optimizations are used.

To meet these challenges, three strategies are proposed to improve the performance and energy efficiency of C4.5. First of all, we introduce additional parallelism to enhance the performance and achieve a considerable speedup of C4.5. The second strategy focuses on storage access on an FPGA. By combining an index table with ping-pong strategy, data hazards can be successfully avoided and memory access bandwidth can be improved. The last strategy is using code folding. By folding similar operations in C4.5, the FPGA resource utilisation is reduced.

B. Additional parallelism for C4.5

To maximise performance and make full use of the available FPGA resources, we introduce further parallislism to the two most computational intensive processes in C4.5: (1) the calculation of best-split attribute and (2) the search of the best split point

1) Parallel calculation of best split attribute: Instead of the traditional depth-first tree construction for the tree growth

phase, we implement breadth first construction. As we can see from Figure 4, computations are executed at once for nodes in the same layer. Each work-group deals with the calculations of one node. As the tree grows deeper, the number of work-groups increases with the number of nodes. This means parallelism has been enhanced through a breadth-first strategy.

Secondly, for a data set which has a large number of attributes, column extension is adopted. Since the computation of different attributes is independent in C4.5, this does not influence the accuracy of final results.

In the end, for those big data sets, row extension can further reduce the number of records to be scanned by every work group. The data set is divided into partitions where every work-group only needs to deal with certain small numbers of records. By applying this reduction technique, results are finally obtained from all work groups.

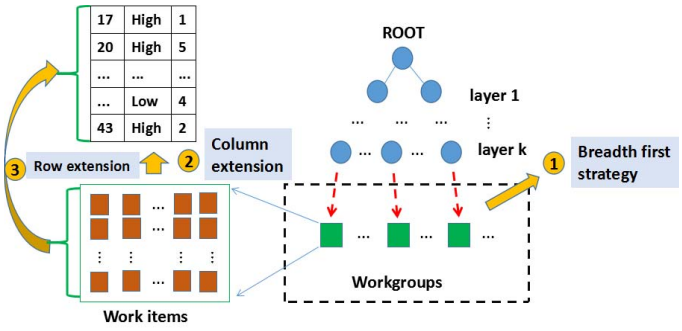

Fig. 4. Parallel extensions for C4.5

2) Parallel calculation of best split point: For continuous and categorical attributes with more than 2 classes of values, it is necessary for each tree node to find the best split point in the best split attribute. As shown in Figure 5, there is an attribute which has five different class values. To find the best split point, five repeated scans of data set are needed in the traditional processing method. By presorting the possible values of the attribute, multiple work-groups can deal with all calculations at the same time without repeatedly searching in data set.

C. Storage access strategy

The efficiency of the memory system is closely tied to the performance of the implementation. Unfortunately, OpenCL does not provide good support to avoid data hazards on FPGAs; data hazards can easily occur when multiple work items try to modify the same block of memory. Furthermore, the memory access latency rises significantly when frequent visits to global memory occur; this is the case for traditional C4.5 implementations.

Data hazards and frequent visits to global memory are two key factors that hinder the efficiency of the memory system on the FPGA.

Data hazards in C4.5 are raised by multiple work-items simultaneously requiring write operations to the same block of memory, especially to global memory on FPGA. Although work-items belonging to the same work-group deal with independent attributes, meaning no data hazards occur. When different work-items try to modify the result column of the up-to-date record, hazards may occur. Also, violations are generated when different work-items in different work-groups try to write to the same block of memory at the same time.

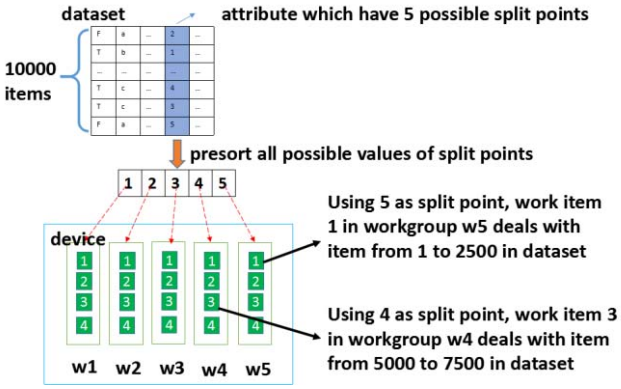

Fig. 5. Parallel serach for split point

With regard to the memory access latency, local memory has a much higher access efficiency than global memory on an FPGA. This means that reducing the frequency of global memory accesses can improve the overall performance significantly.

To solve data hazards and to reduce access times to global memory, two storage access strategies are proposed and shown below.

1) Index table splitting and ping-pong strategy: There is no need to search all records in the data set because only part of the data set is necessary for one node generation in the decision tree. We propose index sets to record the row indices, which are required to scan data sets for each node. To create index sets, line numbers of subspaces needed for child nodes are stored when the father node is being handled. Thus, index tables for each layer of the tree are iteratively generated from the root node.

For instance, during the calculation of the root node, each work-item will generate two index sets to store row indices of '1' and '2' from column a_m which is the best attribute selected (Figure 6). Once the computation is finished on the FPGA, the calculation of the second layer starts. Work-group 1 visits index set {0, 2, ...n-1} to obtain the row indices it needs and at the same time work-group 2 access set {1,3,...n}. Indices in the two sets are totally different as they are generated separately from '1' and '2'. In fact, index sets will never be the same as the tree grows deeper, which means work-groups will never be in conflict with each other. Thus, data access violations are avoided.

2) Optimization for local memory: Although index set techniques solve the problem of access violations to the data set, access to the index table may also cause violations. This is because work-items may write the new indices into the index table while reading it. The use of only one single Dual-Inline-

Memory-Module (DIMM) will limit the bandwidth and may give rise to violations. Based on this, two DIMMs are used as global memory on the FPGA. Once the index table of the upper layer of the tree is being read from one DIMM, the other DIMM is being written with new indices at the same time. As a result, the conflicts during access to index table have been solved, and at the same time, storage efficiency and bandwidth are improved.

Data transactions between host and device in the heterogeneous system are realized through global memory on the device. The results of every layer during the tree growth phase are stored in global memory so the host can visit. However, since access to local memory is much more efficient than access to global, increasing the use of local memory can improve performance.

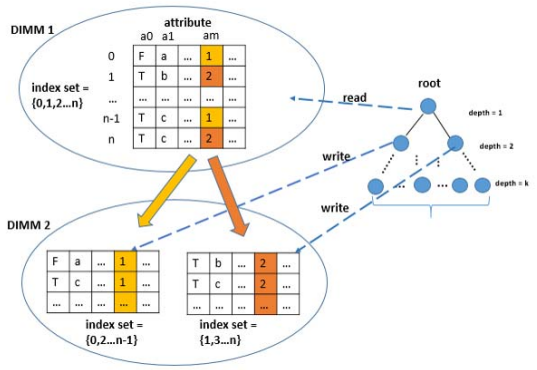

Fig. 6. Index table and ping pong strategy

As shown in Figure 7, to calculate the best split point, every work group will generate an index table to record row numbers needed in generation of nodes in the next layer of the tree. However, only the index table storing the best split point is necessary. To take advantage of local memory, we store all index tables in local memory. Then comparisons to IGRs are done to find the best split point. Only the index table generated by the best split point will be copied to global memory for the visit of the host.

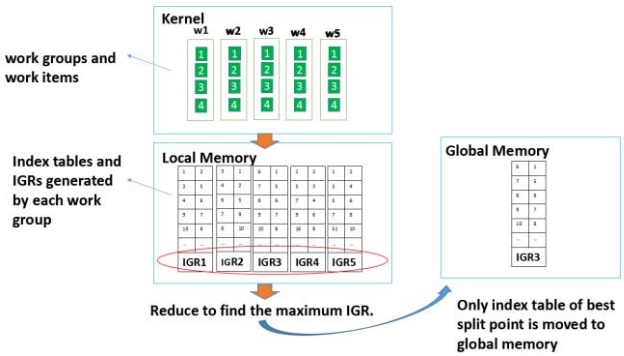

Fig. 7. Optimization for local memory

D. OpenCL optimization techniques

As there are limited resources (Logic gates, memory blocks, DSP blocks etc.) on FPGAs, it is necessary to take resources into consideration when implementing C4.5. To reduce the resources usage on board and enhance the throughput of the system, code folding and several optimization attempts towards Altera's OpenCL Compiler(AOC) are proposed.

1) Code folding strategy: IGR (Information Gain Ratio) is required for the tree growth phase. The formulas to compute IGR are shown in Figure 8. It is undesirable to generate several circuits for every formula, especially since there are loops and repeated calculations in the kernel, but limited FPGA resources.

$$Info(D) = \sum_{i=1}^{m} p_i \log(p_i)$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} Info(D_j)$$

$$Gain(A) = Info(D) - Info_A(D)$$

$$SplitInfo(A) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2 \frac{|D_j|}{|D|}$$

$$IGR(A) = \frac{Gain(A)}{SplitInfo(A)}$$

Fig. 8. Formulas in C4.5

Figure 9 shows the code folding strategy proposed in this paper. As there are similarities among formulas in C4.5, we can fold the similar formulas in one loop with array operations. Using this strategy, AOC generates only one circuit and effectively reuses FPGA resources.

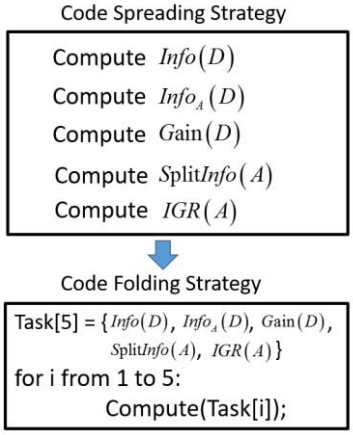

Fig. 9. Code folding strategy

Table 1 presents the reduction in resource utilization through code folding. There are savings for all resources, in particular, the DSP blocks utilization is reduced by 63% with code folding.

TABLE I. RESUOUCE UTILIZATION WHEN USING CODE FOLDING STRATEGY

Stratix V 5SGX FPGA			
	Traditional Scheme	***Folded Scheme***	***Improvement***
Logic Utilization	50%	44%	12%
Dedicated Logic Registers	24%	20%	16%
Memory Bits	54%	47%	13%
DSP Block (18-bit)	147%	55%	63%

2) Optimization towards AOC: Pipeline replication and vectorization are implemented through AOC commands : _attribute (num_compute_units(m)) and _attribute (num_simd_work_items(n))
Here m represents the pipeline replication, and n stands for times of vectorization. When the pipeline is being replicated, computations can be done independently from one another. Vectorization corresponds to SIMD: Single Instruction, Multiple Data. As a result of the parallel design in C4.5, replication and vectorization are combined when kernels are executed.Moreover, since there are several loops in the kernel, it is efficient to unroll the loops within the limitation of resources through #pragma unroll n, where n represents the unroll times. Computations which should be done n times with one loop are finished with n circuits at one time.

As discussed in Section 3.1.5, different kinds of AOC optimization techniques always bring about different costs in resource utilization. To implement the proposed parallel design and split table technique, large numbers of intermediate results are generated and stored. It follows that the memory bits become the most critical resource on FPGA. As a result, the AOC optimisation aims to minimise the use of memory bits.

IV. RESULTS

To test the performance of our energy efficient C4.5 algorithm, we implement it on an FPGA with OpenCL. To verify the reliability of test results, both serial and parallel versions of SPRINT in OpenCL are implemented separately on the CPU and FPGA. Furthermore, a CUDA based SPRINT (CUDT) [21] is used to verify the performance with different sizes of dataset; three data sets are tested in the evaluation section. Energy efficiency ratio (EER) is adopted to compare the energy efficiency of EEC4.5 with SPRINT. In the following subsections, the test environment and result comparisons of these implementations are presented.

A. Test environment

As Figure 10 shows, two target technologies are considered: a CPU and an FPGA. The targeted CPU is an Intel Core i7-3770K running at 3.5GHz. The operating system running on CPU is a Windows 7 64-bit. Visual Studio 2010 is used to debug and test OpenCL program. The FPGA board is a Terasic DE5-Net with a Stratix V 5SGX. Global memory is combined with two independent banks of 8 GB DDR3 SO-DIMM RAM (at an 800MHZ clock rate) and is accessible from the host through a PCIe gen3.0 8x connection. The PCIe connection has a maximum bandwidth of 1GB/s per lane, which means the total bandwidth of the DE5-net is 8GB/s. The local memory implemented on board is M20K RAM blocks running at 600MHZ (total memory of 51200k bits). Private memory on the FPGA is implemented as registers within the data flow and runs at the kernel's frequency. The Kernel program is implemented on the FPGA using Quartus 14.0 64 bits. For comparison purposes, the serial version of SPRINT algorithm in OpenCL is tested on CPU with 8 threads. Both parallel version SPRINT and EEC4.5 in OpenCL are implemented on the same DE5-NET FPGA. To acquire the real-time power of CPU and FPGA, a PingYi PY-G8 power socket is used.

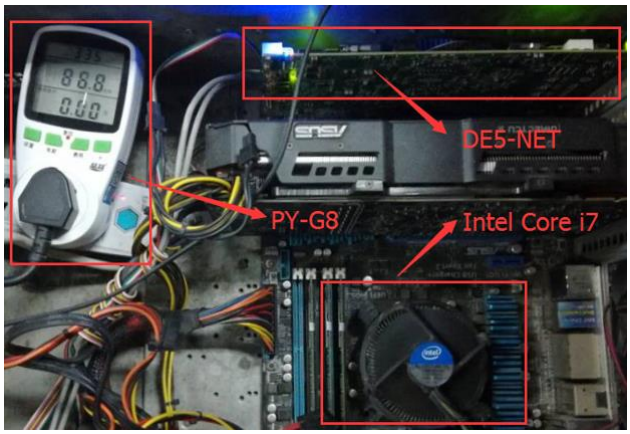

Fig. 10. Test environment

B. Hardware Resources Management and Utilization

The UCI [22] data set is used as training data. Table 2 shows the information of the three data sets adopted in our test. The spambase is a collection of mail data which has 57 continuous attributes of features of spam mails. A categorical attribute denotes spam and nonspam. The number of data instances is 4601. The second data set is Magic Gamma Telescope. It is a physical data set of high energy gamma particles. There are 19020 data in this data set. It has 11 continuous attributes of each record. A categorical attribute denotes the data into two classes. The final data is also physical data. It has 50 attributes and 130065 instances.

TABLE II. DATA SET

Data Set	***Spambase***	***Magic04***	***MiniBooNE***
Attribute Type	Integer, Real	Real	Real
Instances	4601	19020	130065
Attributes	57	11	50
Area	Computer	Physical	Physical

Based on the structure of C4.5 decision tree, the number of work-groups varies according to the number of nodes in tree growth. The sizes of work-groups decrease with the depth of the tree. We update them each time before the kernel is ready

to execute. We stop the split of a node when its max Gain Ratio reaches 0. Since the CPU periodically communicates with devices, I/O time for the control signal is also considered in comparisons which provides a good compromise between computation speed and hardware constraints (regarding memory resources).

Table 3 shows the final resource utilization after using AOC optimizations. To take full advantage of memory resources, the statistic process has been vectorised twice. The pipeline replication is implemented from layer to layer of the tree. Loop unrolling is implemented 8 times in class frequency statistic phase and 4 times in the IGR calculation loop. Four formulas are folded in the IGR calculation phase. Since memory bits utilization reaches 93%, no more optimization methods can be implemented for fear of performance degradation and excessive resources.

TABLE III. FINAL RESOURCE UTILIZATION

Stratix V	
Logic Utilization	75%
Dedicated Logic Registers	36%
Memory Bits	93%
DSP Block (18-bit)	55%
Clock Frequency	131.42MHZ

C. Comparison

Three data sets are adopted to test the performance of EEC4.5. Comparisons are made among serial SPRINT, parallel SPRINT, CUDT and EEC4.5. Three kinds of real-time power are measured: the standby power of the whole system, the power of the whole system with the CPU version program running, and the power of the whole system with FPGA version running. The standby power of the whole machine varies from 84.3w to 89.2w. The power with CPU version C4.5 running ranges from 180w to 185w. When FPGA version of C4.5 (proposed solution and parallel version SPRINT) is running, the power of the whole machine varies from 112w to 116w. Powers of CPU and FPGA are separately measured ten times. The average power of CPU is 97.5w, and the average power of FPGA is 18w. The results of a CUDA based SPRINT (CUDT) from are also taken into consideration. The test platform of CUDT is an Intel Core2 Quad Q6600 with four cores and a GeForce 9800GT. The power of GeForce 9800GT is 110w.

Energy Efficiency Ratio (EER) is used here to make a comparison on energy efficiency. As Figure 11 shows, DP represents the number of data points which is acquired from the number of instances and attributes. EER is calculated from throughput (TOP) and power.

We use ST(s), ST(p) and EEC4.5 to represent serial version SPRINT, parallel version SPRINT, and energy efficient C4.5. Table 4 illustrates the performance of serial SPRINT, parallel SPRINT, CUDT and our EEC4.5 using spambase. As spambase is a data set with a small number of instances and large attributes, the result shows that the EER of EEC4.5 is 3X higher than CUDT and almost 9X higher than parallel SPRINT.

$$DP(dp) = Num\_Inst.Num\_Attr$$

$$TOP(dp.10^3.s^{-1}) = \frac{DP}{Latency}$$

$$EER(dp.10^3.J^{-1}.s^{-1}) = \frac{TOP}{Power}$$

Fig. 11. Calculation of EER

TABLE IV. RESULTS OF SPAMBASE

Spambase	***ST(s)***	***ST(p)***	***CUDT***	***EEC4.5***
Platform	CPU	FPGA	GPU	FPGA
Accuracy (%)	96.25	96.25	97.82	95.41
Tree Size	385	385	385	397
Leaf Node Size	193	193	193	196
Latency (ms)	1989.7	2233.86	124.78	250.8
Speedup	/	0.9	15.9	6
TOP(dp.10^3/s)	131	117	2101	1045
EER(dp.10^3/J)	1	6	19	58

Table 5 shows the results for the Magic04 data set. Compared to spambase, Magic04 is consistent with above. As a result, the parallelism for attributes is restricted. Although the EER of EEC4.5 is only 1.6 times higher than parallel SPRINT, it is amazing to find the EER of EEC4.5 is almost 5 times higher than that of CUDT.

TABLE V. RESULTS OF MAGIC04

Magic04	***ST(s)***	***ST(p)***	***CUDT***	***EEC4.5***
Platform	CPU	FPGA	GPU	FPGA
Accuracy (%)	92.8	92.8	93.54	92.3
Tree Size	1579	1579	1579	1658
Leaf Node Size	790	790	790	824
Latency (ms)	3927	497.06	257.72	309.26
Speedup	/	7.9	15.4	12.7
TOP(dp.10^3/s)	53.2	420.9	811.8	676.5
EER(dp.10^3/J)	0.6	23	7	38

Table 6 presents the performance of implementations with the MiniBooNE data set. The size of MiniBooNE is much larger than spambase and Magic04. The results show that the EER of EEC4.5 is about 13 times higher than parallel SPRINT and is 3 times higher than that of CUDT.

TABLE VI. RESULTS OF MINIBOONE

MiniBooNE	*ST(s)*	*ST(p)*	*CUDT*	*EEC4.5*
Platform	CPU	FPGA	GPU	FPGA
Accuracy (%)	98.31	98.31	98.31	97.78
Tree Size	8127	8127	8127	8231
Leaf Node Size	4064	4064	4064	4192
Latency (ms)	288539	56870	2452	4739
Speedup	/	5.07	117.67	60.9
TOP(dp.10^3/s)	23	114	2652	1372
EER(dp.10^3/J)	0.2	6	24	76

V. CONCLUSIONS

In this paper, a CPU-FPGA heterogeneous platform based solution was proposed to maximize the energy efficiency of the C4.5 algorithm using OpenCL design flow. Design challenges we met include the low degree of parallelism in C4.5, the huge data comparison and scan demand, the potential data hazards in memory, the limited logic/memory resources on FPGA, and the cost of AOC optimization methods. To address the challenges and achieve better performance, we proposed parallelism extensions for C4.5, a Ping-Pong based index table, and a code folding strategy based on OpenCL for fast board-level deployment. Three data sets were selected (including spambase, Magic04 and MiniBooNE) to evaluate our proposed design. By running these data sets, the serial SPRINT, the parallel SPRINT and the CUDT were compared. Results showed that our approach achieved 58X, 63X, 380X higher energy efficiency than the basic serial C4.5 algorithm (serial SPRINT) running on CPU; achieved 10X, 1.6X, 12.7X higher energy efficiency than the parallel SPRINT running on the same CPU-FPGA heterogeneous platform; and achieved 3X, 5X, 3X higher energy efficiency than the existing CUDA based parallel C4.5 (CUDT) running on CPU-GPU heterogeneous platform

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their constructive and helpful suggestions and comments. This paper was supported by the National Natural Science Foundation of China (NSFC) under Grant: No.61534002, the Chinese National Program Support of Top-Notch Young Professionals (1st Batch), Central Universities under Grant: ZYGX2016J042.

REFERENCES

[1] Young Choon Lee and Albert Y Zomaya. Energy efficient utilization of resources in cloud computing systems. Journal of Supercomputing, 60(2):268{280, 2012.

[2] Yuliang Pu, Jun Peng, Letian Huang, and John Chen. An efficient knn algorithm implemented on fpga based heterogeneous computing system using opencl. In International Symposium on Field-Programmable Custom Computing Machines, pages 13{35, 2015.

[3] Xiaofan Zhang, Xinheng Liu, Anand Ramachandran, Chuanhao Zhuge, Shibin Tang, Peng Ouyang, Zuofu Cheng, Kyle Rupnow, and Deming Chen. High-performance video content recognition with long-term recurrent convolutional network for fpga. In International Conference on Field Programmable Logic and Applications, 2017.

[4] John E Stone, David Gohara, and Guochun Shi. Opencl: A parallel programming standard for heterogeneous computing systems. Computing in Science & Engineering, 12(3):66{72, 2010.

[5] J. Ross Quinlan. Induction on decision tree. Machine Learning, 1(1):81{106, 1986.

[6] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. Knowledge & Information Systems, 14(1):1{37, 2008.

[7] Salvatore Ruggieri. Efficient c4.5. IEEE Transactions on Knowledge & Data Engineering, 14(2):438{444, 2002.

[8] John Shafer, Rakesh Agrawal, and Manish Mehta. Sprint: A scalable parallel classifier for data mining. In Proceedings of the 22th International Conference on Very Large Data Bases, pages 544{555, 2000.

[9] Shanshan Fei, Qiaoyan Wen, and Zhengping Jin. Analysis and Improvement of SPRINT Algorithm Based on Hadoop. Springer International Publishing, 2014.

[10] Ping He, Ling Chen, and Xiao-Hua Xu. Fast c4.5. In International Conference on Machine Learning and Cybernetics, pages 2841{2846, 2007.

[11] Jason R Beck, Maria Garcia, Mingyu Zhong, Michael Georgiopoulos, and Georgios C Anagnostopoulos. A backward adjusting strategy and optimization of the c4.5 parameters to improve c4.5's performance. In International Florida Artificial Intelligence Research Society Conference, pages 35{40, 2008.

[12] Amine Bermak and Dominique Martinez. A compact 3d vlsi classifier using bagging threshold network ensembles. IEEE Transactions on Neural Networks, 4(5):1097{1109, 2003.

[13] Santos Lopez-Estrada and Rene Cumplido. Decision tree based fpga-architecture for texture sea state classification. In IEEE International Conference on Reconfigurable Computing and Fpga's, pages 1{7, 2006.

[14] Weirong Jiang and Viktor K Prasanna. Scalable packet classification on fpga. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 20(9):1668{1680, 2012.

[15] Fareena Saqib, Aindrik Dutta, Jim Plusquellic, Philip Ortiz, and Marios S Pattichis. Pipelined decision tree classification accelerator implementation in fpga (dt-caif). IEEE Transactions on Computers, 64(1):280{285, 2015.

[16] Ramanathan Narayanan, Daniel Honbo, Gokhan Memik, Alok Choudhary, and Joseph Zambreno. An fpga implementation of decision tree classification. In Design, Automation & Test in Europe Conference & Exhibition, pages 1{6, 2007.

[17] Toby Sharp. Implementing decision trees and forests on a gpu. In European Conference on Computer Vision, pages 595{608, 2008.

[18] Sebastian Nowozin, Carsten Rother, Shai Bagon, Toby Sharp, Bangpeng Yao, and Pushmeet Kohli. Decision tree fields. In IEEE International Conference on Computer Vision, pages 1668{1675, 2011.

[19] Toby Sharp. Evaluating decision trees on a gpu., October 16 2012. US Patent 8,290,882.

[20] Aziz Nasridinov, Yangsun Lee, and Young-Ho Park. Decision tree construction on gpu: ubiquitous parallel computing approach. Computing, 96(5):403{413, 2013.

[21] Win-Tsung Lo, Yue-Shan Chang, Ruey-Kai Sheu, Chun-Chieh Chiu, and Shyan-Ming Yuan. Cudt: A cuda based decision tree algorithm. Scientific World Journal, 2014(2014):745640, 2014.

[22] C. Blake. Uci repository of machine learning databases. http://www.ics.uci.edu/~aAIJmlearn/MLRepository.html ˘ , 1998.