Computing (2014) 96:403–413
DOI 10.1007/s00607-013-0343-z

Decision tree construction on GPU: ubiquitous parallel computing approach

Aziz Nasridinov · Yangsun Lee · Young-Ho Park

Received: 11 April 2013 / Accepted: 8 August 2013 / Published online: 24 August 2013

Abstract General Purpose Graphic Processing Unit (GPGPU) computing with CUDA has been effectively used in scientific applications, where huge accelerations have been achieved. However, while today's traditional GPGPU can reduce the execution time of parallel code by many times, it comes at the expense of significant power and energy consumption. In this paper, we propose ubiquitous parallel computing approach for construction of decision tree on GPU. In our approach, we exploit parallelism of well-known ID3 algorithm for decision tree learning by two levels: at the outer level of building the tree node-by-node, and at the inner level of sorting data records within a single node. Thus, our approach not only accelerates the construction of decision tree via GPU computing, but also does so by taking care of the power and energy consumption of the GPU. Experiment results show that our approach outperforms purely GPU-based implementation and CPU-based sequential implementation by several times.

Keywords Ubiquitous computing · GPU computing · CUDA · Decision tree

Mathematics Subject Classification 68T05 · 68Q32 · 68T30

A. Nasridinov · Y.-H. Park (✉)
Department of Multimedia Science,
Sookmyung Women's University, Seoul, Korea
e-mail: yhpark@sm.ac.kr

A. Nasridinov
e-mail: aziz@sm.ac.kr

Y. Lee
Department of Computer Engineering,
Seokyeong University, Seoul, Korea
e-mail: yslee@skuniv.ac.kr

1 Introduction

Graphic Processing Unit (GPU) is a ubiquitous device, which exists in every personal computing system. Conventionally, GPU has been used for graphics processing including images, graphical user interfaces (GUIs), videos, and games. However, the modern GPU is not a simple graphics processor, but it demonstrates a high throughput on certain problems, and GPU's near universal use in desktop computers means that it is cheap and ubiquitous source of processing power [1–3]. There is increasing interest in applying this power to general-purpose problems through frameworks such as NVIDIA's Compute Unified Device Architecture (CUDA). CUDA is an application programming interface developed to give programmers a standard way to execute general-purpose logic on GPUs. Programmers often use CUDA and similar interfaces to accelerate computationally intensive data processing operations, often executing them 50 times faster on the GPU [4].

General Purpose GPU (GPGPU) computing with CUDA has spread in scientific applications ranging from computational biology, to computational finance and electronic designs, where huge accelerations have been achieved [5]. However, while today's traditional GPGPU can reduce the execution time of parallel code by many times, it comes at the expense of significant power and energy consumption [6,7]. For example, the NVIDIA GTX Titan graphics card has a power requirement of 250 watts, which is as much as the rest of a compute node. Thus, minimum power supply required by this graphics card is 650 W. Consequently, the GPU is considered as a "non-green" computing solution.

In this paper, we propose a ubiquitous parallel computing approach for construction of decision tree on GPU. The reason why we selected decision tree algorithm to prove our point is that the implementation of the decision tree algorithm is by nature parallelized due to the low level of interdependency between data operations in the distance calculation and sorting phases [8,9]. It holds that CUDA would be an effective way to adapt this version of decision tree for parallel execution. In our approach, we exploit parallelism of well-known ID3 algorithm for decision tree learning by two levels: at the outer level of building the tree node-by-node, and at the inner level of sorting data records within a single node. Thus, our approach not only accelerates the construction of decision tree via GPU computing, but also does so by taking care of the power and energy consumption of the GPU. An experiment result shows that our approach outperforms purely GPU-based implementation and CPU-based sequential implementation.

The remainder of this paper is organized as follows. Section 2 presents background to our research. Section 3 discusses related work. Section 4 describes our proposed approach. Section 5 presents performance evaluation. Section 6 highlights conclusions and future work.

2 Background

In this section, we present background to our research. Specifically, we describe the data classification and a widely used solution to it, the ID3 algorithm for decision tree learning. CUDA and GPU architecture will also be explained.

2.1 Classification using decision tree

Data classification is used to predict the value of a class in a data set based on the values of its other attributes [10,11]. Decision trees are one of the well-known classification techniques. They are widely used due to comparatively rapid to compute, simple to understand by humans, and they can reach accuracies similar to other well-known classification techniques. The ID3 algorithm [12] is a widely used data classification solution for decision tree learning. In this paper, we deal with ID3 algorithm.

A decision tree is a tree consisting of a root node, child nodes and edges. Each internal node is a test node that indicates the attribute; the edges indicate the possible values taken on by that attribute. Each non-leaf node consists of a splitting point, and the main task for building the decision tree is to identify the test attribute for each splitting point [13]. The ID3 algorithm uses the *information gain* to select the test attribute. Information gain can be computed using *entropy*. In the following, we assume there are m classes in the whole training data set. We know

$$\mathit{Entropy}(S) = -\sum_{j=1}^{m} Q_j(S) \log Q_j(S)$$

where $Q_j(S)$ is the relative frequency of class j in S. We can compute the information gain for any candidate attribute A being used to partition S:

$$\mathit{Gain}(S, A) = \mathit{Entropy}(S) - \sum_{v \in A} \left(\frac{|S_v|}{|S|} \mathit{Entropy}(S_v) \right)$$

where v represents any possible values of attributes A; S_v is the subset of S for which attributes A has value v; $|S|$ is the number of elements in S.

2.2 CUDA and GPU architecture

Graphic Processing Unit is a ubiquitous device, which exists in every personal computing system. There is increasing interest in applying this power to general-purpose problems through frameworks such as NVIDIA's CUDA, an application programming tool developed to provide programmers a standard way to implement general-purpose applications on NVIDIA GPUs. A CUDA program is a kind of C program and consists of functions that can be executed on both CPU and GPU. The functions launched in the threads on GPU are called device (kernel) functions, and the CPU functions are called host functions. The simplified architecture of the NVIDIA GPUs that supports CUDA is shown in Fig. 1.

The GPGPU chip consists of several streaming multi-processors (MP). Each MP has many number of CUDA cores (shared processors, SP). Each SP contains a large number of registers and a private local memory. Thread synchronization through the shared memory is only held between threads running on the same MP. The GPU is then constructed by combining a number of MPs. The graphics

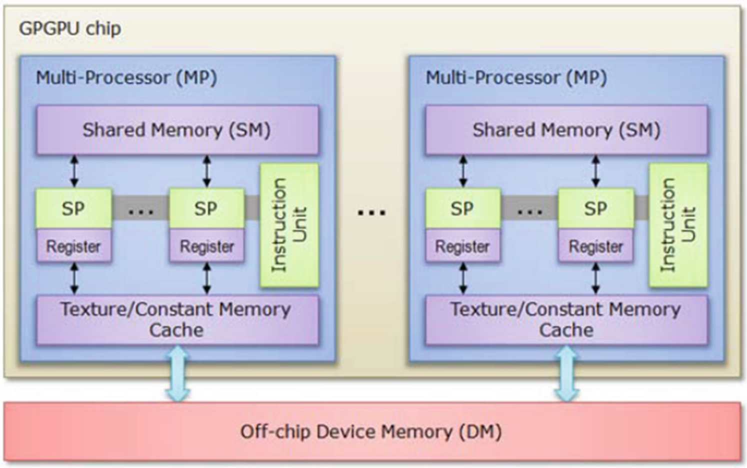

Fig. 1 The GPU architecture

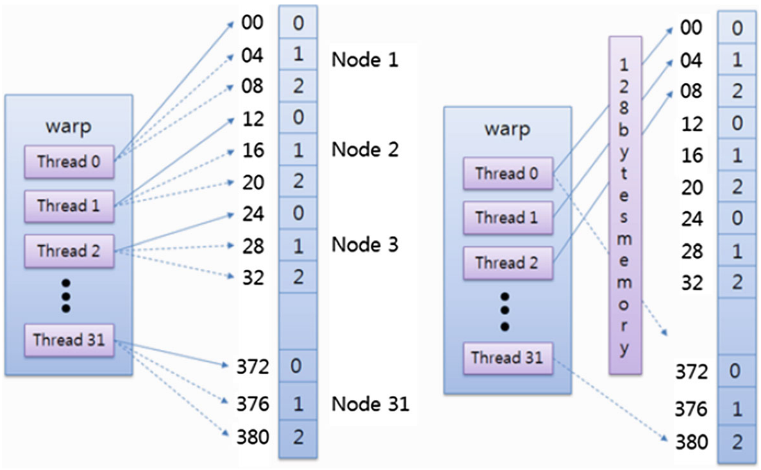

Fig. 2 Memory access comparison of CPU and GPU architecture

card also has a number of additional memories that are accessible from all SPs. Each SP executes a thread for a same kernel function, and hence the kernel function is executed in parallel in a massive number of concurrent threads on a GPU [14].

Before kernel function is executed, the required data must be relocated from the host memory to the device memory, and execute kernel function in a similar way as calling a regular C function. Memory access comparison of CPU and GPU architecture is shown in Fig. 2.

3 Related work

There have been many researches towards the performance improvement of decision tree. In this section, we will describe representative ones.

Shafer et al. [15] proposed a new classification algorithm called SPRING that eliminates all memory limitations that restrict decision-tree algorithms, and proves that proposed algorithm is fast and scalable. In their paper, the authors compare their algorithm with well-known SLIQ [16] algorithm and demonstrated that SPRINT is effective in both serial and parallel environments. The authors also demonstrate that SPRINT can handle datasets that are too large for SLIQ to handle. Consequently, the parallel efficiency of SPRINT improves as the problem size increases.

Researchers from Google Inc, Panda et al. [17] proposed PLANET, a scalable distributed framework for learning tree models over large datasets. PLANET considers tree learning as a series of distributed computations, implements each one using the MapReduce model of distributed computation. In their paper, the authors benefits and challenges of using a MapReduce compute cluster for tree learning and illustrate the scalability of proposed approach by using it for a real world learning task from the domain of computational advertising. Another approach is presented by Ben-Haim and Tom-Tov [18], where the authors proposed a new streaming parallel decision tree algorithm. The proposed algorithm constructs histograms at the processors, which collects the data to a fixed amount of memory. A master processor exploits this information to look for near optimal split points to terminal tree nodes. Proposed algorithm is launched in a distributed environment and is applicable for classifying large data sets and streaming data.

Used properly, computationally intensive data processing operations, such as decision tree, can be accelerated several times faster on the GPU. There are several approaches [19–21] who have tried to do so. For example, Sharp [19] proposes an approach for implementing the evaluation and training of decision trees and forests entirely on a GPU. The authors demonstrate how proposed approach can be used in the context of object recognition. Their strategy for evaluation involves mapping the data structure describing a decision forest to a 2D texture array. They traverse through the forest for each point of the input data in parallel using an efficient, non-branching pixel shader. For training, they calculate the responses of the training data to a set of candidate features, and distribute the responses into a appropriate histogram using a vertex shader. The histograms thus computed can be used together with a broad range of tree learning algorithms. The experiment results show that proposed approach gains speed of around 100 times.

The Random Forest learning algorithm, called CUDARF, is introduced by Breiman et al. [20]. The CUDA implemented random forests parallelize both building and classification. They use a CUDA thread to build a tree. All trees are built in parallel to each other but the trees themselves are built sequentially. However, this approach is not suitable for large data sets if the data set is bigger than GPU memory size. Identical approach is proposed by Chiu et al. [21] where authors have improved the implementation of decision tree using CUDA architecture.

4 Proposed method

Recall from Sect. 1 that while today's traditional GPGPU can reduce the execution time of parallel code by many times, it comes at the expense of significant power and energy consumption [6,7]. In this section, we describe proposed approach. In this paper, we propose a ubiquitous parallel computing approach for construction of decision tree on GPU. The reason why we selected decision tree algorithm to prove our point is that the implementation of the decision tree algorithm is by nature parallelized due to the low level of interdependency between data operations in the distance calculation and sorting phases [8,9].

The decision tree construction process in hybrid CPU–GPU method is called with two parameters: *D*, attribute list, and attribute selection method. We refer to *D* as a data partition. Initially, it is the complete set of training tuples and their associated class labels. The parameter attribute list is a list of attributes describing the tuples. Attribute selection method specifies a heuristic procedure for selecting the attribute that "best" discriminates the given tuples according to class. This procedure employs an attribute selection measure, such as information gain. Whether the tree is strictly binary is generally driven by the attribute selection measure. Some attribute selection measures, such as information gain, do not, therein allowing multi-way splits meaning that two or more branches to be grown from a node.

The following steps illustrate the main execution steps in our implementation. Further, Fig. 3 shows which parts of the execution are done on the host CPU and on

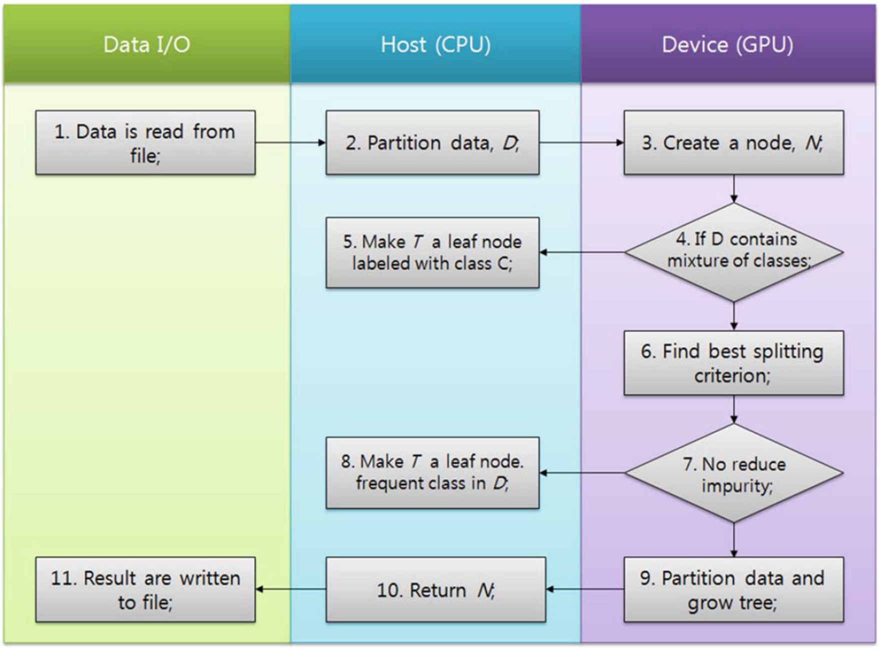
Data I/O
Host (CPU)
Device (GPU)
1. Data is read from file;
2. Partition data, D;
3. Create a node, N;
4. If D contains mixture of classes;
5. Make T a leaf node labeled with class C;
6. Find best splitting criterion;
7. No reduce impurity;
8. Make T a leaf node. frequent class in D;
9. Partition data and grow tree;
10. Return N;
11. Result are written to file;

Fig. 3 Execution flow for communication of host and devices in decision tree algorithm

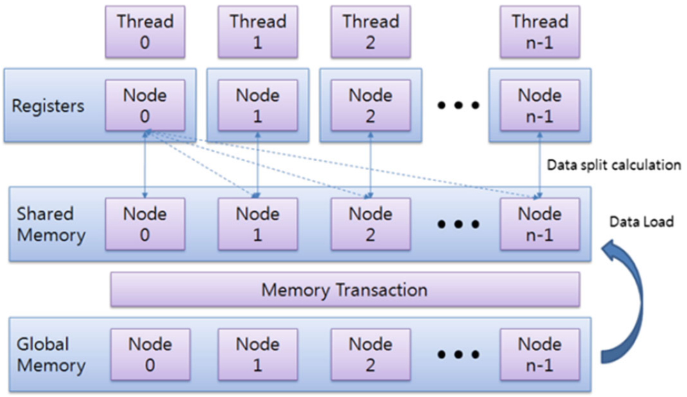

Fig. 4 GPU-based decision tree architecture

```
__global__ void entropy(float *dest, float *freqs, int total)
{
        __shared__ float sdata[SHARED_LENGTH];
        int tid = threadIdx.x + blockIdx.x * blockDim.X;
        float partial = freqs[tid]/total;
        sdata[threadIdx.x] = partial * log2f(partial);

        __syncthreads();

        for(int s = blockDim.x/2; s > 0; s >>= 1)
        {
                if(tid < s)
                        sdata[tid] += sdata[tid + s];
                else if(tid == s && (tid & 0x01))
                        sdata[blockIdx.x] += sdata[tid + s];
        {
        __syncthreads();

        if (threadIdx.x == 0) dest[blockIdx.x] = stada[0];

}
```

Fig. 5 A decision tree on CUDA

the device GPU, respectively, as well as the data transfers that take place between the host and the device (GPU).

The data is read from file and passed to the data partition to the CPU (step 1 and 2). The tree starts as a single node, N, representing the training tuples in D (step 3). Step 3 is performed on the device GPU. Figure 4 illustrates how decision tree algorithm is mapped in GPU architecture. Further, Fig. 5 demonstrates the pseudocode of implementation of decision tree on CUDA.

If the tuples in D are all of the same class, then node N becomes a leaf and is labeled with that class (steps 4 and 5). Note that steps 4 and 5 are terminating conditions in host CPU. Otherwise, the process calls Attribute selection method to determine the splitting criterion. The splitting criterion tells us which attribute to test at node N by determining the best way to separate or partition the tuples in D into individual classes (step 6). The splitting criterion also tells us which branches to grow from node N with respect to the outcomes of the chosen test. More specifically, the splitting criterion indicates the splitting attribute and may also indicate either a split-point or a splitting subset. The splitting criterion is determined so that, ideally, the resulting partitions at each branch are as "pure" as possible. If splitting criteria do not significantly reduce the impurity (entropy reduction), then N becomes a leaf and is labeled with that class, which is the most frequent class in D (step 8). A branch is grown from node N for each of the outcomes of the splitting criterion and the tuples in D are partitioned accordingly. The algorithm uses the same process recursively to form a decision tree for the tuples at each resulting partition, D_j, of D (steps 9). The ready tree is returned and saved in file (step 10 and 11).

The proposed approach not only accelerates the construction of decision tree via GPU computing, but also does by taking care of the power and energy consumption of the GPU. Similar approaches to our approach is a research carried out by Park et al. [22], where the performance metrics of well-known algorithms are studied.

5 Performance evaluation

In this section, we present performance evaluation of our approach. The aim of the experiment is to compare the computation time of the proposed CUDA-based decision tree with its state-of-the art sequential and parallel CPU counterparts.

5.1 Experimental Setup

The software platform used consists of Microsoft Windows 7 together with CUDA version 5.0. The hardware platform consists of an Intel Core i5 CPU and 4 GB of DDR3 RAM. Table 1 summarized GPU characteristics used in the experiments. The GPU used is an NVIDIA GT220 card with 1 GB.

5.2 Experiment results

We have carried out experiments on power consumption and execution time. We have compared our method, ubiquitous parallel computing approach with a well-known decision tree classification tool, Weka and decision tree tool that's solely implemented on GPU. Figures 6 and 7 demonstrate the results of this experiment.

First, Fig. 5 reveals the results of execution time comparison. In Fig. 5, horizontal axis indicates the number of elements in decision tree and vertical axis represents processing time units measured in milliseconds.

From Fig. 6, it is obvious that Weka algorithm shows the highest execution time because we implemented it via CPU. Thus, as the number of elements in decision tree increases the execution time of Weka algorithm increases exponentially. CUDA-RF and Hybrid CPU–GPU enjoys parallel features of GPGPU described in Sect. 3. Thus, they demonstrate better performance than Weka algorithm. Our approach outperforms CUDA-RF, because in our approach, we exploit divide-and-conquer parallelism in ID3 at two levels: at the outer level of building the tree node-by-node in a top-down, recursive manner, and at the inner level of sorting data records within a

Table 1 Characteristics of Geforce GT220 graphics card

Property	Value
CUDA cores	48
Compute capabilities	1.2
Graphic/Processor Clock	625 Mhz/1.36 Ghz
Total amount of memory	1 Gb
Memory interface	128-bit DDR3
Power consumption	450 W

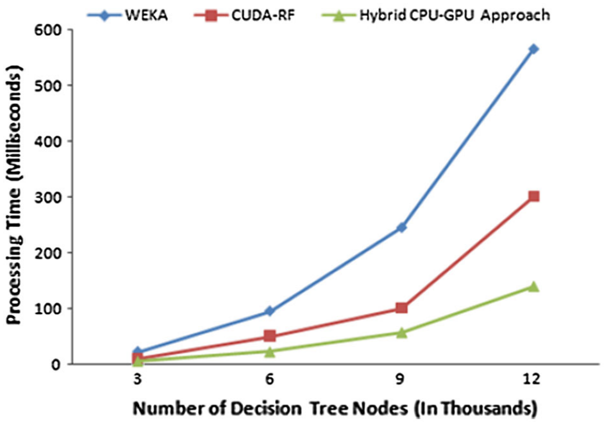

Fig. 6 Comparison of execution time

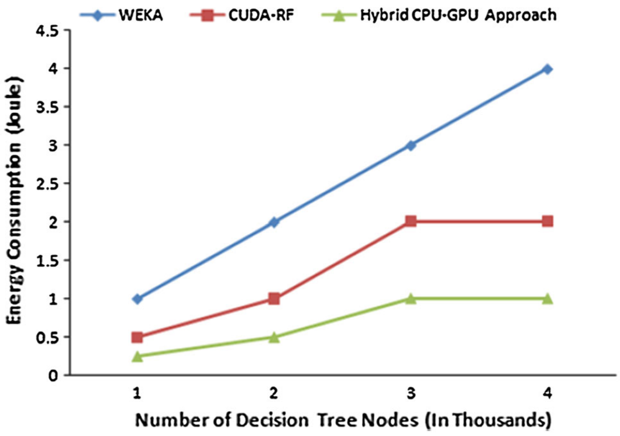

Fig. 7 Comparison of energy consumption

single node. Thus, the proposed Hybrid CPU–GPU approach outperforms CUDA-RF by 1.5 times. We have carried out experiments on power consumption and execution time.

We have compared our method, ubiquitous parallel computing approach with a well-known decision tree classification tool, Weka and decision tree tool that's solely implemented on GPU. Figures 6 and 7 demonstrate the results of this experiment.

The comparison results are shown in Fig. 7. In Fig. 7, horizontal axis indicates the number of elements in decision tree and vertical axis represents energy consumption units measured in Joule $\times 10^3$.

From the graph in Fig. 7, it is obvious that CUDA-RF which is implemented solely on GPU consumes much energy comparing to other approaches. This is because of the GPGPU can reduce the execution time of a parallel code by many times, but it comes at the expense of significant power and energy consumption. CPU's energy consumption is lowest because conventionally, CPUs are energy efficient units. However, our approach is not too much different. This is because we employed GPU only to solve the calculation problems and left manipulation problems to the CPU. Thus, our approach not only accelerates the construction of decision tree via GPU computing, but also does so in the context of characterizing the power and energy consumption of the GPU.

6 Conclusion

In this paper, we have proposed ubiquitous parallel computing approach for construction decision tree on GPU. We exploited divide-and-conquer parallelism in ID3 at two levels: at the outer level of building the tree node-by-node in a top-down, recursive manner, and at the inner level of sorting data records within a single node. Thus, our approach not only accelerates the construction of decision tree via GPU computing, but also does so in the context of characterizing the power and energy consumption of the GPU. The experiment results showed that hybrid implementation outperforms a purely GPU-based implementation and a CPU-based sequential implementation.

Ackowledgments This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012003797). The work was also supported by the IT R & D program of MKE/KEIT. (10041854, Development of a smart home service platform wth real-time danger prediction and prevention for safety residential environments).

References

1. Bakum P, Skadron K (2010) Accelerating SQL database operations on a GPU with CUDA. In: GPGPU'10: proceedings of the third workshop on general-purpose computation ongraphics processing units. pp 94–103
2. Kim J, Kim SG, Nam B (2013) Parallel multi-dimensional range query processing with R-trees on GPU. J Parallel Distrib Comput 73(8):2164–2179
3. Cayrel PL, Hoffmann G, Schneider M (2011) GPU implementation of the Keccak Hash function family. Int J Secur Appl 5(4):123–132
4. Che S, Boyer M, Meng J, Tarjan D, Sheaffer JW, Skadron K (2008) A performance study of general-purpose applications on graphics processors using CUDA. J Parallel Distrib Comput 68(10):1370–1380
5. Nguyen H (2007) GPU Gems 3. Addison-Wesley Professional, Reading

6. Huang S, Xiao S, Feng W (2009) On the energy efficiency of graphics processing units for scientific computing. In: Proceeding of the 2009 IEEE international symposium on parallel and distributed processing. pp 1–8
7. Yu CD, Wan W, Pierce D (2011) A CPU–GPU hybrid approach for the unsymmetric multifrontal method. Parallel Comput 37(12):759–770
8. Garcia V, Debreuve E, Barlaud M (2008) Fast k nearest neighbour search using GPU. In: Computer vision and pattern recognition workshops. pp 1–6
9. Kufrin R (1997) Decision trees on parallel processors. Mach Intell Pattern Recognit 20:279–306
10. Chui FCF, Bindoff I, Williams R (2009) Applying feature extraction for classification problems. J Signal Process Image Process Pattern Recognit 2(1):1–16
11. Cheng X, Xu J, Pei J, Liu J (2010) Hierarchical distributed data classification in wireless sensor networks. Comput Commun 33(12):1404–1413
12. Quinlan JR (1986) Induction of decision trees. Mach Learn 1(1):81–106
13. Teng Z, Du W (2007) A hybrid multi-group privacy-preserving approach for building decision trees. In: Proceedings of the 11th Pacific-Asia conference on advances in knowledge discovery and data mining. pp 296–307
14. Van Der Laan WJ, Jalba AC, Roerdink JBTM (2011) Accelerating wavelet lifting on graphics hardware using CUDA. IEEE Trans Parallel Distrib Syst 22(1):132–146
15. Shafer CJ, Agrawal R, Mehta M (1996) SPRINT: a scalable parallel classifier for data mining. In: Proceedings of the 22th international conference on very large data bases. pp 544–555
16. Mehta M, Agrawal R, Rissanen J (1996) SLIQ: a fast scalable classifier for data mining. In: Proceedings of the 5th international conference on extending database technology: advancesin database technology. pp 18–32
17. Panda B, Herbach JS, Basu S, Bayardo RJ (2009) PLANET: massively parallel learning of tree ensembles with MapReduce. J VLDB Endow 2(2):1426–1437
18. Ben-Haim Y, Tom-Tov E (2010) A streaming parallel decision tree algorithm. J Mach Learn Res 11(3):849–872
19. Sharp T (2008) Implementing decision trees and forests on a GPU. In: Proceeding 10th European conference on computer vision. pp 595–608
20. Grahn H, Lavesson N, Lapajne HM, Slat D (2011) CudaRF: a CUDA-based implementation of random forests. In: Proceedings of the 2011 9th IEEE/ACS international conference on computer systems and applications. pp 95–101
21. Chiu CC, Luo GH, Yuan SM (2011) A decision tree using CUDA GPUs. In: Proceedings of the 13th international conference on information integration and web-basedapplications and services. pp 399–402
22. Park YH, Whang KY, Lee BS, Han WS (2006) Efficient evaluation of linear path expressions on large-scale heterogeneous XML documents using information retrieval techniques. J Syst Softw 79(2):180–190