Parallel tree-ensemble algorithms for GPUs using CUDA

Karl Jansson and Håkan Sundell
School of Business and IT
University of Borås
501 90, Borås, Sweden
{ karl.jansson, hakan.sundell }@hb.se

Henrik Boström
Dept. of Computer and Systems Sciences
Stockholm University
Forum 100, 164 40, Kista, Sweden
henrik.bostrom@dsv.su.se

ABSTRACT

We present two new parallel implementations of the tree-ensemble algorithms Random Forest (RF) and Extremely randomized trees (ERT) for emerging many-core platforms, e.g., contemporary graphics cards suitable for general-purpose computing (GPGPU). Random Forest and Extremely randomized trees are ensemble learners for classification and regression. They operate by constructing a multitude of decision trees at training time and outputting a prediction by comparing the outputs of the individual trees. Thanks to the inherent parallelism of the task, an obvious platform for its computation is to employ contemporary GPUs with a large number of processing cores. Previous parallel algorithms for Random Forests in the literature are either designed for traditional multi-core CPU platforms or early history GPUs with simpler hardware architecture and relatively few number of cores. The new parallel algorithms are designed for contemporary GPUs with a large number of cores and take into account aspects of the newer hardware architectures as memory hierarchy and thread scheduling. They are implemented using the C/C++ language and the CUDA interface for best possible performance on NVidia-based GPUs. An experimental study comparing with the most important previous solutions for CPU and GPU platforms shows significant improvement for the new implementations, often with several magnitudes.

1. INTRODUCTION

Classification is an important problem within machine learning for predicting the classes to which previously unseen examples belong, given a training set of data with known classes. Building the classifier can be complex and often computationally expensive. With the increasing need of analyzing big data, solutions based on sequential execution are also increasingly infeasible. On the other hand, the hardware of modern computer systems offer an increasingly amount of parallelism, and even standard systems can easily be upgraded from multi-core to many-core systems using relatively affordable accelerators as graphics cards (GPU). The technology of GPUs are rapidly improving, contemporary systems have even advanced cache architectures and support for atomic operations together with offering a massive parallelism of several thousands of processing cores. However, to be efficiently used on these systems, the underlying algorithms of interest need to be properly parallelized.

The Random Forest (RF) classifier [3] is a well-known and well used machine learning algorithm that can produce competitive classifiers, in terms of classification performance, compared to more computationally demanding classifiers such as Neural Networks or SVMs [7]. The Random Forest algorithm is sometimes referred to as being embarrassingly parallel [2], since the trees of a forest may be generated independently of each other in parallel. However, in order to exploit the full parallelization potential on the GPU, a tree-level approach is not sufficient. There exist numerous examples of parallel implementations of the Random Forest algorithm on CPUs; An Erlang based implementation [2], a MapReduce based implementation called PLANET [12], a C++ implementation called Random Jungle [13] and a Java-based implementation called FastRF [15] created for the Weka platform [8] to name a few. There also exist implementations for the massively parallel platform that GPUs supply, which is the platform used in this paper; a GPU Random Forest implementation by Toby Sharp [14] built with DirectX, using shaders for kernels and texture buffers for storage. A more recent GPU implementation is the CudaRF implementation [6]. This implementation, like the name suggest, uses CUDA [11] as its way of communicating with the GPU. The implementation builds multiple trees in parallel on the graphics card by assigning trees to the available cores on the graphics card. No additional parallelization is performed on an internal tree level. Rather, each core sequentially generates a whole tree of the forest on its own. A comparison between using GPUs, FPGAs and multi-core systems have also been performed [16], with the conclusion that an FPGA is preferable in terms of raw performance. However, similar to the above studies, the comparison did not employ the most recent GPU technology, and this study will bring some light on whether further efficiency gains can be obtained from using these.

Focusing on the GPU implementations of the Random Forest algorithm currently available; they are not ideal to use on newer types of graphics cards available today. Toby Sharp's implementation for example does not use the improved APIs available for more advanced GPGPU programming available today. CudaRF was created at a time when the GPUs still had a relatively small number of cores and thus uses a one-tree-per-core model which does not scale well, when the sizes of the datasets increase, with contemporary GPUs having several thousands of cores. It is thus interesting to investigate how far the Random Forest algorithm can be parallelized to make use of all the cores available on contemporary (and future) GPUs.

In this paper we present two new implementations of the tree-ensemble algorithms Random Forest and Extremely randomized trees for the GPU that exploits the parallelization potential all the way down to the inter-node level instead of stopping on the tree level. One implementation is presented that follows the original Random Forest scheme and one implementation that follows the Extremely randomized trees (ERT) scheme. The implementations are compared with three multi-core CPU implementations (FastRF,

This work has been supported by the Swedish Foundation for Strategic Research through the project High-Performance Data Mining for Drug Effect Detection (ref. no. IIS11-0053) at Stockholm University, Sweden.

cpuERT and cpuRF) and a GPU implementation (CudaRF) to assess the efficiency of the parallelization methods on newer types of GPUs.

2. NEW ALGORITHMS

For this work, two GPU-accelerated tree-ensemble algorithms are developed in CUDA; gpuRF and gpuERT. gpuRF uses the algorithm of the original Random Forest [3], that finds the best split on a certain attribute through checking every possible split. For numerical attributes, this entails sorting the instances and evaluating all possible split points. gpuERT uses the approach of Extremely randomized trees [5]. In contrast to the previous algorithm, a single split point is randomly selected for a numerical attribute, avoiding the need to sort and evaluate all possible split points. This obviously improves efficiency substantially and the idea is that this could be done without an expected reduced predictive performance of the forest, as the reduced performance of each single tree is compensated for by an increased diversity, something which strengthens the overall performance of the ensemble.

2.1 Parallelization scheme

The parallelization scheme employed aims at exposing as much of the parallel potential in the tree-ensemble algorithms as possible, while targeting the GPU architecture. To accomplish this, a node-level approach is taken rather than a tree-level approach, which is the easier way of handling tree generation and more commonly employed in CPU implementations. Furthermore, a breadth-first method of generating trees is employed, since a recursive depth-first approach is not very efficient on a GPU. This means that a whole level of the forest is evaluated until there are no more nodes generated. The node-level approach allows for a higher parallelization potential the further down in the trees the generating algorithm gets, since more nodes that can be processed in parallel are generated from the preceding levels.

The CPU portion of the gpuRF and gpuERT algorithm is presented in Algorithm 1, with the exception that `kernel_sort` is executed only for the gpuRF implementation.

Algorithm 1 $GenerateForest(nrTrees, nrFeatures)$

```
1: nrNodes ← nrTrees
2: kernel_baggingInit(nrTrees)
3: while nrNodes > 0 do
4:     bestSplit ← 0
5:     k ← nrFeatures
6:     while k > 0 do
7:         kernel_sort(nrNodes) % for gpuRF only
8:         bestSplit ← kernel_findSplit(nrNodes)
9:         k ← k − 1
10:    end while
11:    kernel_splitNodes(nrNodes, bestSplit)
12:    nrNodes ← kernel_createNewNodes(nrNodes)
13: end while
```

The gpuRF algorithm divides the tree-building algorithm into five larger phases. The first phase (`kernel_baggingInit`) performs the bootstrap sampling for the trees on the GPU, and also initializes the GPU buffers used in the subsequent phases. The second phase (`kernel_sort`) is a sorting phase where each node selects an attribute randomly and sorts its instances in descending order. The third phase (`kernel_findSplit`) searches through all the possible splits on an attribute and selects the best possible split that the attribute can produce, in accordance to the entropy split criterion. The fourth phase (`kernel_splitNodes`) performs the actual split of a node in memory based on the best attribute and split determined in the previous iterations of phase one and two. The fifth and last phase (`kernel_createNewNodes`) determines if the split of a node has generated a leaf node or an internal node and saves them accordingly to memory; for storage if a leaf node, for further processing in the next iteration if an internal node.

The gpuERT algorithm flow is nearly identical to the gpuRF flow with the exception that there is no sorting kernel due to the split point being selected on random as well as the attribute. This also means that the `kernel_findSplit` phase only need to evaluate one single split of a node instead of every possible split of the node, since the split point is already selected and it is not the intention of the algorithm to find the best possible split the attribute could potentially produce. Outside these, there are no other significant differences in terms of algorithm flow between the implementations.

The phases of the algorithms are represented in terms of kernels in the GPU implementations and are used for grid launches onto the GPU, where each grid launch processes one tree-level of nodes for the whole forest. For all kernels except `kernel_createNewNodes`, where one node is processed by one thread, each grid launch contains one block of threads for each node that is to be processed. In other words, one node is processed by one block of threads. This allows for efficient usage of the shared memory on the GPUs multiprocessors, since a whole block is guaranteed to reside on the same multiprocessor and have access to the same shared memory. This also means that the exposed parallelism for a grid launch can be calculated by the number of threads in a block, times the number of nodes in the grid launch. To expose additional parallelism, the block size can thus be increased. However, too large blocks can limit the number of blocks that can reside on a single multiprocessor of the GPU, and prevent optimal usage of the GPUs cores. Considering this, a relatively small block size of 64 has been chosen to support efficient execution on a variety of GPU-models.

3. EXPERIMENTAL SETUP

To assess the performance of the GPU implementations, a comparison is made with CPU variants as well as with the GPU implementation CudaRF. The chosen CPU implementation is FastRF 0.99. It supports multi-threaded execution, which is crucial for giving the CPU versions a fair chance against the GPU versions. The code for FastRF has been acquired through its website [15] and the code for CudaRF has been obtained through personal communication with the authors. The code has only been modified to output the required metrics, no modifications are performed on the actual algorithms. In addition to these existing implementations two parallel C++ variants of Random Forest and Extremely Randomized Trees (cpuRF and cpuERT) are developed and used as additional reference points.

3.1 Evaluation

Three different characteristics of the implementations are measured and evaluated; Classification performance, Tree scaling and Instance scaling.

The classification metrics chosen for the experiments are Accuracy and area under ROC curve (AUC). Some datasets are heavily unbalanced and accuracy is not an ideal measurement of classifications for such datasets [4]. Thus AUC will be the more important measurement for these datasets. The classification results are not directly presented due to space constraints.

The time efficiency metric employed, in all experiments concerning time efficiency, is the average time (in seconds) taken to train and evaluate a single fold in stratified 10-fold cross validation. This method of measuring is aimed at giving more stable measurements when dealing with the inherent randomness of the algorithms that

Table 1: Time efficiency (seconds)

	gpuERT		gpuRF		CudaRF		cpuERT		cpuRF		FastRF	
Data set	**100x**	**1000x**	**100x**	**1000x**	**100x**	**1000x**	**100x**	**1000x**	**100x**	**1000x**	**100x**	**1000x**
Census-Income	**3,942**	**33,929**	11,196	86,031	-	-	4,914	47,204	5,271	49,243	13,302	127,946
Bank-Marketing	**0,399**	**3,301**	1,136	7,491	-	-	0,813	6,129	0,741	6,880	1,601	11,524
Adult	**0,351**	**2,925**	0,824	4,972	584,179	2287,529	0,795	6,496	0,628	5,683	1,553	12,348
Mushroom	0,031	0,119	**0,019**	**0,094**	2,164	6,913	0,038	0,287	0,040	0,336	0,266	1,528
Spambase	**0,105**	**0,645**	0,317	1,535	18,683	71,486	0,130	0,964	0,086	0,761	0,217	1,152
kr-vs-kp	0,047	0,264	**0,028**	**0,176**	4,196	14,131	0,081	0,632	0,046	0,417	0,150	0,607
Eula-Freq	**0,038**	**0,114**	0,047	0,117	0,953	3,357	0,038	0,266	0,030	0,228	0,694	4,647
Breast-Cancer-Wis	**0,004**	**0,021**	0,012	0,050	0,130	0,464	0,015	0,078	0,007	0,047	0,040	0,123
Skin-Disorder	0,185	0,335	**0,059**	**0,257**	3,816	6,522	0,190	1,259	0,085	0,710	2,449	20,797
House-Votes	**0,006**	0,021	0,009	**0,019**	0,123	0,310	0,017	0,090	0,005	0,027	0,031	0,118

are evaluated.

The instance scaling experiment is performed by re-sampling the census-income dataset in 10 different sizes (1-10 percent) and measurements are then taken for each re-sampled dataset respectively. For the efficiency experiments, the metrics are collected for ten measurement points ranging from 100 trees to 1000 with incremental steps of 100 trees for each dataset. The number of features considered in each split follows the formula $log_2(FeatureCount)+1$ (denoted k) and the tree depth of the forest is limited to a maximum of 100 in all experiments.

3.2 Experiment environment specifications

The experiments are conducted on a system with the following specifications:

- CPU: 2x Intel(R) Xeon(R) E5-2690 2.90Ghz
  16 physical cores (32 logical cores)
  Memory: 28 GB DDR3
- GPU: NVidia Tesla K20c (ECC enabled)
  2496 CUDA cores, Driver version: 320.00
  Memory: 5 GB GDDR5
- OS: Microsoft Windows Server 2012

This system can run both GPU and CPU implementations very well due to its high-end GPU and CPUs. The GPU of the system is at an equal price point to that of the two CPUs at the time of writing. All CPU implementations are run with 32 worker threads to make full use of the hyper-threaded processors of the test environment.

3.3 Datasets

The datasets used in the experiments range from smaller datasets with few instances and features to datasets with many instances and features, as well as balanced and unbalanced datasets. The intent with this selection is to explore where the presented implementation's strengths and weaknesses lie. Most of the datasets are from the UCI repository [1] except for Eula-Freq that comes from [10] and is used in the article on CudaRF [6] and the Skin-Disorder[1] dataset that comes from the article by Karlsson et al. [9]. The selection of the UCI datasets is primarily motivated by what datasets have been used in similar studies, to support comparisons as far as possible, while keeping a good mix of diverse datasets. The dataset specifications are presented in Table 2. For implementations that do not have full support for handling missing values in its tree generating algorithm (CudaRF, gpuRF and gpuERT), the missing values are substituted with the means or modes of the attribute in which the missing value resides.

[1]This data set has been approved by the Regional Ethical Review Board in Stockholm (Etikprövningsnämnden i Stockholm), permission number 2012/834-31/5.

Table 2: Dataset specifications

Dataset	Nr Instances	Nr Features	k	Nr Missing values
Census-Income	199523	40	6	415717
Bank-Marketing	45211	16	5	0
Adult	32561	14	4	4262
Mushroom	8124	22	5	2480
Spambase	4601	57	6	0
Kr-vs-kp	3196	36	6	0
Eula-Freq	996	1268	11	0
Breast-Cancer-Wis	569	30	5	0
Skin-Disorder	462	1669	11	0
House-Votes	435	16	5	392

4. EXPERIMENTAL RESULTS

The classification performances in terms of accuracy and AUC are comparable between the implementations with slight advantages for the CPU implementations.

4.1 Tree Scaling

The average efficiency across datasets is presented in Figure 1 and run-times for 100 and 1000 trees on the datasets are presented in Table 1.

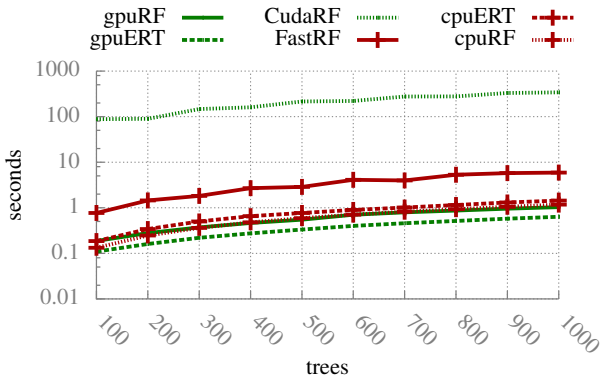

Figure 1: Average efficiency for the datasets (excluding Census-Income and Bank Marketing).

Census-Income and Bank Marketing are excluded from the average, since all implementations were not able to run these large datasets. It can be observed that the gpuRF, cpuRF and cpuERT implementations have a similar average efficiency for the included datasets. FastRF and CudaRF are the least efficient implementations according to these results and gpuERT is the most efficient. It can be observed in Table 1 that gpuERT is the most efficient

on most datasets, but gpuRF is more efficient than gpuERT on the datasets with mostly nominal values.

4.2 Instance scaling

The results for time efficiency scaling with instances are depicted in Figure 2. For the GPU implementations, it can be observed that the gpuRF and gpuERT scales more or less linearly, while the CudaRF suffers from a much lower scaling than linear when the instance count is increased. The main reason for this scaling is that CudaRF only exposes the inherent tree-level parallelism in Random Forest and therefore assigns a very heavy workload on single GPU cores when the instance count increases. Looking at the CPU implementations, it can be observed that all of the implementations scale more or less linearly, with the cpuERT implementation having the best scaling while FastRF being slightly worse.

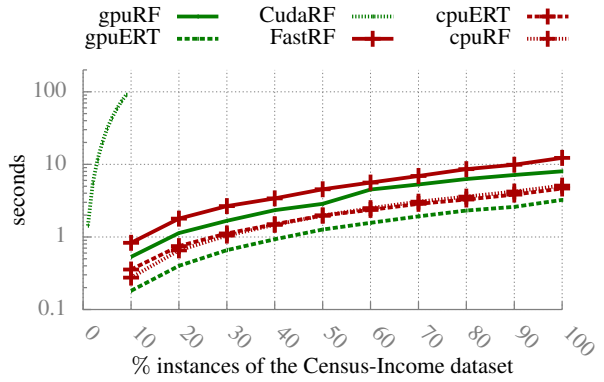

Figure 2: Instance scaling on the census-income dataset.

When comparing the CPU and GPU implementations with each other, it can be observed that gpuRF and gpuERT both are more efficient than FastRF, but also that gpuRF is slightly less efficient than cpuERT and cpuRF. It should be noted that comparing GPU and CPU implementations in this way should be taken with a grain of salt, since they are running on entirely different platforms and it is not necessarily the efficiency of the algorithms that is being compared but rather a combination of platform and algorithm.

5. CONCLUSIONS

We have presented two new GPU-accelerated tree-ensemble generating algorithms geared toward utilizing the extremely parallel architecture of contemporary and future high-end GPUs. The node-level approach of the developed algorithms, compared to a tree-level approach on the GPU (CudaRF), far exceeds the tree-level approach in terms of efficiency and exposes more than enough parallelism to fully utilize contemporary high-end GPUs. Furthermore the GPU implementations, when compared to CPU implementations run on CPUs in the same price range as the used GPU, performs similarly or better for their respective implementation types. The classification performance for the developed GPU implementations is overall competitive with the classification performance of state-of-the-art CPU implementations. In future studies it would be interesting to explore adaptive block-sizes, dependent on the dataset size, to further reduce the amount of sequential work needed by a single core on the GPU when the dataset size increases. This would expose even more parallelism potential from the problem domain and could potentially increase the efficiency of the algorithms. It would also be interesting to experiment with even bigger datasets to see how far the developed GPU implementations can be pushed in terms of handling huge datasets. In closing, it can be concluded that the two developed GPU tree-ensemble algorithms performs very well on the GPU architecture. Primarily this can be attributed to the node-level approach that exposes the necessary amount of parallelism needed to make full use of the available computational power of the GPU. It can also be noted that the exposed parallelism of the algorithms far exceed the number of cores on the GPU that have been used in the experiments, and can be further increased with modifications to the block-size. They should therefore scale well with future GPUs, given that the trend with an ever increasing number of cores per GPU continues.

6. REFERENCES

[1] BACHE, K., AND LICHMAN, M. Uci machine learning repository, 2013.

[2] BOSTRÖM, H. Concurrent learning of large-scale random forests. In *Proceedings of Scandinavian Conference on Artificial Intelligence (SAIS)* (2011), pp. 20–29.

[3] BREIMAN, L. Random forests. *Machine Learning 45* (2001), 5–32.

[4] CHAWLA, N. V. Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*. Springer, 2005, pp. 853–867.

[5] GEURTS, P., ERNST, D., AND WEHENKEL, L. Extremely randomized trees. *Machine learning 63*, 1 (2006), 3–42.

[6] GRAHN, H., LAVESSON, N., LAPAJNE, M. H., AND SLAT, D. Cudarf: A cuda-based implementation of random forests. In *Computer Systems and Applications (AICCSA)* (2011), IEEE-ACS, pp. 95–101.

[7] HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. *Elements of Statistical Learning*. Springer, 2009.

[8] HOLMES, G., DONKIN, A., AND WITTEN, I. H. Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on* (1994), IEEE, pp. 357–361.

[9] KARLSSON, I., ZHAO, J., ASKER, L., AND BOSTRÖM, H. Predicting adverse drug events by analyzing electronic patient records. In *Artificial Intelligence in Medicine*. Springer, 2013, pp. 125–129.

[10] LAVESSON, N., BOLDT, M., DAVIDSSON, P., AND JACOBSSON, A. Learning to detect spyware using end user license agreements. *Privacy-Invasive Software* (2010), 129.

[11] NVIDIA. *Cuda C Programming Guide*, 2013.

[12] PANDA, B., HERBACH, J. S., BASU, S., AND BAYARDO, R. J. Planet: massively parallel learning of tree ensembles with mapreduce. *Proceedings of the VLDB Endowment 2* (2009), 1426–1437.

[13] SCHWARZ, D. F., KÃŰNIG, I. R., AND ZIEGLER, A. On safari to random jungle: a fast implementation of random forests for high-dimensional data. *Bioinformatics 26* (2010), 1752–1758.

[14] SHARP, T. Implementing decision trees and forests on a gpu. In *10th European Conference on Computer Vision (ECCV)* (2008), Springer, pp. 595–608.

[15] SUPEK, F. Fast random forest (fastrf), mar 2013.

[16] VAN ESSEN, B., MACARAEG, C., GOKHALE, M., AND PRENGER, R. Accelerating a random forest classifier: multi-core, gp-gpu, or fpga? In *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on* (2012), IEEE, pp. 232–239.