Learning Random Forests on the GPU

Yisheng Liao **Alex Rubinsteyn** **Russell Power** **Jinyang Li**
Department of Computer Science
New York University
{yisheng,alexr,power,jinyang}@cs.nyu.edu

Abstract

Random Forests are a popular and powerful machine learning technique, with several fast multi-core CPU implementations. Since many other machine learning methods have seen impressive speedups from GPU implementations, applying GPU acceleration to random forests seems like a natural fit. Previous attempts to use GPUs have relied on coarse-grained task parallelism and have yielded inconclusive or unsatisfying results. We introduce CudaTree, a GPU Random Forest implementation which adaptively switches between data and task parallelism. We show that, for larger datasets, this algorithm is faster than highly tuned multi-core CPU implementations.

1 Introduction

Random Forests [3] are a popular [6, 23] learning algorithm for tackling complex prediction problems. They are able to achieve high accuracy across a wide range of datasets, while exposing few hyper-parameters in need of tuning. There are many high-quality CPU implementations of Random Forests, such as scikit-learn [16], wiseRF [18], SPRINT [14], and Random Jungle [20]. .

Many other machine learning algorithms have been dramatically accelerated using graphics cards, including convolutional neural networks [11], support vector machines [4], and Latent Dirichlet Allocation [25]. However, none of the commonly available Random Forest libraries can make use of a GPU. At first glance, Random Forests may appear to be an ideal candidate for GPU parallelization: the trees of a random forest can be trained independently, so why not do this in parallel?

As we show in this paper, coarse-grained parallelization yields disappointing results on a GPU, which consists of a large number of a individually weak cores. To reap the benefits of many-core hardware, it is necessary to find sources of data parallelism within the training procedure for a single tree. However, data parallelism alone breaks down toward the "bottom" of tree construction, where the number of samples being considered can become small. We show that is possible to overcome this performance barrier by switching from data parallelism to batching the construction of smaller sub-trees.

2 Related Work

In the most widely cited work on GPU decision tree training, Sharp [21] used Direct3D pixel and vertex shaders to implement Random Forests for visual object recognition. His GPU implementation, however, was also tasked with computing visual feature responses and thus it is unclear to what degree the same code is suitable for general-purpose (non-visual) learning tasks. Other Random Forest implementations on the GPU [7, 15] seem to under-utilize the available parallelism of graphics hardware and have only undergone cursory evaluations.

Aside from previous attempts to use GPUs for Random Forest learning, there is an older and deeper literature describing the implementation of single decision trees on (non-GPU) parallel proces-

sors [1]. Some of these parallel algorithms, such as SLIQ [13], partition data across multiple processors. Other parallel decision tree learning algorithms [12, 22] blend data and task parallelism. This is similar in spirit to the approach taken by CudaTree.

3 Design

To parallelize learning of a single decision tree, we considered two high-level strategies: data parallel *depth-first* tree construction and fine-grained task-parallel *breadth-first* construction.

Depth-First Use the GPU to compute the optimal split-point for a single node of the decision tree. Each CUDA thread block is responsible for a subset of the samples from a single feature. We first perform a parallel prefix scan [8] to compute label count histograms. This is then followed by a parallel evaluation of the GINI impurity score for all possible feature thresholds. Lastly, we perform a parallel reduction to determine which feature/threshold pair has the lowest impurity score.

Breadth-First Instead of constructing a single tree node for each set of kernel launches (as in the Depth-First algorithm), we can instead construct a whole level of the decision tree simultaneously. Each CUDA thread block is tasked with computing the optimal split for a single tree node in the current level.

During the course of our work, we found that each of these techniques has a "sweet-spot" in which it performs well. The Depth-First algorithm is efficient during the early stages of tree construction, when large numbers of examples are being processed. As trees become deeper, however, the overhead of invoking GPU kernels to evaluate small numbers of samples becomes dominant. *Breadth-first* construction is less efficient at the top of a tree but can significantly decrease the kernel launch overhead by processing many sites on a tree at the same time. It seems natural to combine the strengths of both strategies into a hybrid tree construction algorithm.

Hybrid This strategy starts tree construction using the Depth-First algorithm, but finishes growing smaller sub-trees using the Breadth-First algorithm. The crossover point between Depth-First and Breadth-First is determined by a linear model whose parameters are estimated from performance on randomly generated data.

Finally, for comparison, we consider the coarse-grained task parallelism used by multi-core CPU implementations:

Coarse Task Parallel In imitation of the usual multi-CPU task parallelism, this algorithm uses a single CUDA thread block to build each tree of the ensemble. The actual tree induction is performed using a sequential recursive algorithm similar to that used by scikit-learn. Recursion on the GPU is implemented with a manually managed stack of sample index ranges.

3.1 Determining the Breadth-First vs. Depth-First Crossover Threshold

The Hybrid GPU algorithm must know when a sub-tree is sufficiently small to switch from Depth-First to Breadth-First learning. At first, we attempted to use a fixed number of samples as the switching parameter. Switch-over points between 2000 and 25,000 samples all tend to perform better than scikit-learn, whereas using thresholds significantly outside this range severely degrades performance. However, we observed that no single fixed parameter yields optimal performance across different datasets.

In addition to the complexity of the prediction task at hand (which often implicitly determines tree height), the performance of GPU decision tree learning depends on the number of samples in a dataset, along with the number of classes and features. To better incorporate these properties of a dataset, we cast the problem of determining an optimal crossover parameter as regression.

We generated 75 different synthetic datasets with the number of samples ranging from 20,000 to 250,000, the number of categories between 2 and 500, and the number of features between 8 and 512. We trained a CudaTree forest on each dataset with ten transition parameters between 1000 and

30,000 samples. The transition parameter with the shortest running time was then recorded as a target output (associated with dataset characteristics). Estimating ordinary least squares coefficients results in the following formula for the Depth-First vs. Breadth-First transition:

$$\boxed{3702 + 1.58c + 0.0577n + 21.84f}$$

where c is the number of classes, n is the number of samples, and f is the number of features considered at each split. Though this formula was estimated from performance on randomly generated data, in our experience it results in reasonable switch-over points for real data.

3.2 Implementation

All of the above algorithms were written in Python using PyCUDA [9] to compile GPU kernels and transfer data to/from the GPU. Numerically intensive computations on the host were accelerated using Parakeet [19], a runtime compiler for numerical Python. The source for the CudaTree is available online at `https://github.com/EasonLiao/CudaTree`.

4 Evaluation

4.1 Test Setup

All tests were conducted on two machines with 6-core Xeon E5-2630 processors and 24GB of memory. One machine had an NVIDIA Tesla C2075 GPU (448 cores at 1.15GHz) and the other had a GTX Titan (2,688 cores at 836MHz). The software used was Ubuntu Linux (12.10), Python (2.7.5), NumPy (1.7.1), scikit-learn (0.14), wiseRF (1.5.11), PyCUDA (2013.1.1) and Parakeet (0.17.1). All the Random Forest implementations below were trained with 100 trees grown to full height (no early stopping criteria). Every Random Forest was configured to evaluate $\sqrt{n}$ features per split (where n is the total number of features in a dataset).

4.2 Comparison of GPU Algorithms

We compared the performance of our four GPU learning algorithms as we vary both the number of trees 1 and the number of features 2. In both experiments, Hybrid Parallelism is faster than the alternative GPU algorithms, so we will not focus on them any further.

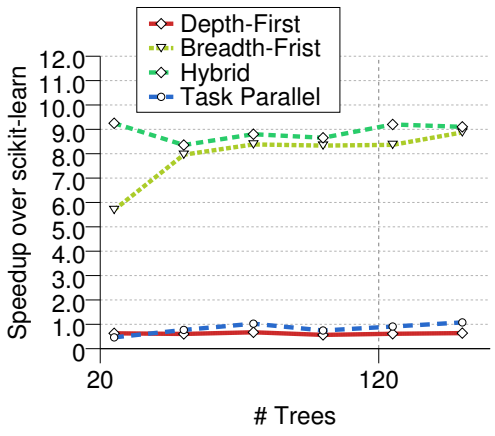

Figure 1: Speedup relative to scikit-learn over varying numbers of trees when learning on the raw pixel data of the CIFAR-10 dataset.

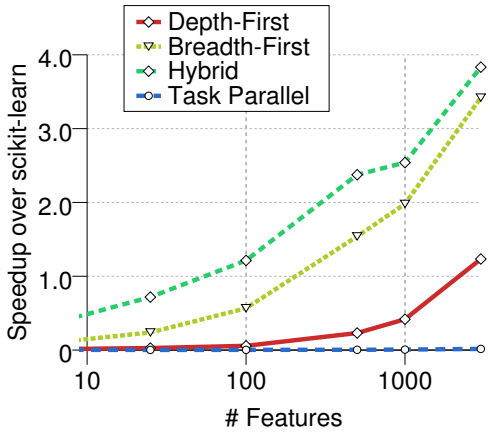

Figure 2: Speedup relative to scikit-learn on varying numbers of features on a synthetic random dataset with 50k samples.

4.3 Comparison against CPU algorithms

We compared the performance of our GPU algorithm with scikit-learn and wiseRF on the six datasets described in Table 1. We tested CudaTree on both a latest generation NVIDIA GTX Titan graphics card and an older Tesla C2075. In addition, we created a heterogeneous Random Forest implementation which trains simultaneously trains using both CudaTree and a multi-core Random Forest library. We tested this heterogeneous implementation on the Titan GPU running concurrently with wiseRF on 6 CPU cores.

The results of this comparison are shown in Table 2. Across the six datasets, CudaTree is 1.8x - 7.6x faster than scikit-learn and for the four larger datasets, also faster than wiseRF. Combining CudaTree with wiseRF yields significant improvements over the performance of either library individually.

DATASET	SAMPLES	FEATURES	CATEGORIES	DESCRIPTION
ImageNet subset	10k	4,096	10	Output from conv. layer of a CNN [11]
CIFAR-100	50k	3,072	100	Raw pixel values of images [10]
covertype	581k	57	7	Forest cover data from UCI [2]
poker	1M	11	10	Poker hands [5]
PAMAP2	2.87M	52	13	Physical activity monitoring [17]
intrusion	5M	41	24	Data from KDD Cup '99 [24]

Table 1: Dataset Descriptions

DATASET	WISERF	SCIKIT-LEARN	GPU (TITAN)	GPU (C2075)	GPU + CPU
ImageNet subset	**23s**	50s	27s	55s	25s
CIFAR-100	160s	502s	197s	568s	**94s**
covertype	107s	463s	67s	125s	**52s**
poker	117s	415s	59s	122s	**58s**
PAMAP2	1,066s	7,630s	934s	1,636s	**757s**
intrusion	667s	1,528s	199s	400s	**153s**

Table 2: Random Forest Training Times

5 Conclusion

We compared four different approaches to Random Forest construction on the GPU and found Hybrid parallelism to be faster than task or data parallelism alone. We then compared the performance of our hybrid parallel algorithm to two commonly used multi-core Random Forest libraries: scikit-learn and wiseRF.

On all six datasets used, CudaTree was able to construct a 100 tree Random Forest faster than scikit-learn. Furthermore, on the four larger datasets, CudaTree was faster than wiseRF. When we combined CudaTree with wiseRF and used both simultaneously, the training times on the five larger datasets were significantly lower than any individual implementation.

The ability to efficiently construct decision trees on the GPU opens up interesting avenues for future work. It is possible to use the same techniques to accelerate decision tree boosting algorithms, which thus far have been often relegated to single core CPU implementations. Furthermore, the basic design of CudaTree should work well across multiple GPUs, as well as multiple machines.

6 Acknowledgments

"Random Forests" is a trademark of Leo Breiman and Adele Culter. We would like to thank the NVIDIA corporation for kindly donating a GPU for our experiments.

References

[1] AMADO, N., GAMA, J., AND SILVA, F. Parallel implementation of decision tree learning algorithms. In *Progress in Artificial Intelligence*. Springer, 2001, pp. 6–13.

[2] BAY, S. D., KIBLER, D., PAZZANI, M. J., AND SMYTH, P. The UCI KDD archive of large data sets for data mining research and experimentation. *ACM SIGKDD Explorations Newsletter 2*, 2 (2000), 81–85.

[3] BREIMAN, L. Random forests. *Machine learning 45*, 1 (2001), 5–32.

[4] CATANZARO, B., SUNDARAM, N., AND KEUTZER, K. Fast support vector machine training and classification on graphics processors. In *Proceedings of the 25th international conference on Machine learning* (2008), ACM, pp. 104–111.

[5] CATTRAL, R., OPPACHER, F., AND DEUGO, D. Evolutionary data mining with automatic rule generalization. *Recent Advances in Computers, Computing and Communications* (2002), 296–300.

[6] CUTLER, D. R., EDWARDS JR, T. C., BEARD, K. H., CUTLER, A., HESS, K. T., GIBSON, J., AND LAWLER, J. J. Random forests for classification in ecology. *Ecology 88*, 11 (2007), 2783–2792.

[7] GRAHN, H., LAVESSON, N., LAPAJNE, M. H., AND SLAT, D. CudaRF: A CUDA-based implementation of Random Forests. In *AICCSA* (2011), H. J. Siegel and A. El-Kadi, Eds., IEEE, pp. 95–101.

[8] HARRIS, M., SENGUPTA, S., AND OWENS, J. D. Parallel prefix sum (scan) with cuda. *GPU gems 3*, 39 (2007), 851–876.

[9] KLÖCKNER, A., PINTO, N., LEE, Y., CATANZARO, B., IVANOV, P., AND FASIH, A. Pycuda and pyopencl: A scripting-based approach to gpu run-time code generation. *Parallel Computing 38*, 3 (2012), 157–174.

[10] KRIZHEVSKY, A., AND HINTON, G. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto* (2009).

[11] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25* (2012), pp. 1106–1114.

[12] KUFRIN, R. Decision trees on parallel processors. *Machine Intelligence and Pattern Recognition 20* (1997), 279–306.

[13] MEHTA, M., AGRAWAL, R., AND RISSANEN, J. Sliq: A fast scalable classifier for data mining. In *Advances in Database TechnologyEDBT'96*. Springer, 1996, pp. 18–32.

[14] MITCHELL, L., SLOAN, T. M., MEWISSEN, M., GHAZAL, P., FORSTER, T., PIOTROWSKI, M., AND TREW, A. S. A parallel random forest classifier for R. In *Proceedings of the second international workshop on Emerging computational methods for the life sciences* (2011), ACM, pp. 1–6.

[15] NASRIDINOV, A., LEE, Y., AND PARK, Y.-H. Decision tree construction on GPU: ubiquitous parallel computing approach. *Computing* (2013), 1–11.

[16] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., ET AL. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research 12* (2011), 2825–2830.

[17] REISS, A., AND STRICKER, D. Introducing a new benchmarked dataset for activity monitoring. In *Wearable Computers (ISWC), 2012 16th International Symposium on* (2012), pp. 108–109.

[18] RICHARDS, J. W., EADS, D., BLOOM, J. S., BRINK, H., AND STARR, D. Wiserftm: A fast and scalable random forest. [Online; accessed 25-October-2013].

[19] RUBINSTEYN, A., HIELSCHER, E., WEINMAN, N., AND SHASHA, D. Parakeet: A just-in-time parallel accelerator for Python. In *Proceedings of the 4th USENIX conference on Hot Topics in Parallelism* (2012), USENIX Association, pp. 14–14.

[20] SCHWARZ, D. F., KÖNIG, I. R., AND ZIEGLER, A. On safari to random jungle: a fast implementation of random forests for high-dimensional data. *Bioinformatics 26*, 14 (2010), 1752–1758.

[21] SHARP, T. Implementing decision trees and forests on a GPU. In *Computer Vision–ECCV 2008*. Springer, 2008, pp. 595–608.

[22] SRIVASTAVA, A., HAN, E.-H., KUMAR, V., AND SINGH, V. *Parallel formulations of decision-tree classification algorithms*. Springer, 2002.

[23] STROBL, C., MALLEY, J., AND TUTZ, G. An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests. *Psychological methods 14*, 4 (2009), 323.

[24] TAVALLAEE, M., BAGHERI, E., LU, W., AND GHORBANI, A.-A. A detailed analysis of the KDD CUP 99 data set. In *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009* (2009).

[25] YAN, F., XU, N., AND QI, Y. Parallel inference for latent dirichlet allocation on graphics processing units. In *Advances in Neural Information Processing Systems* (2009), pp. 2134–2142.