GPU-acceleration for Large-scale Tree Boosting

Huan Zhang
UC Davis
Davis, CA
ecezhang@ucdavis.edu

Si Si
Google Research
Mountain View, CA
sisidaisy@google.com

Cho-Jui Hsieh
UC Davis
Davis, CA
chohsieh@ucdavis.edu

1 INTRODUCTION

Decision tree has become one of the most successful nonlinear learning algorithms in many machine learning and data mining tasks. Many algorithms are proposed based on decision trees and tree ensemble methods, such as random forest [1–3], gradient boosting decision trees (GBDT) [4], and regularized greedy forest [5]. These algorithms have shown superb performance in regression, classification, and ranking [6] tasks.

Among these tree ensemble methods, GBDT has gained lots of attention recently due to its superb performance and its flexibility of incorporating different loss functions. However, it is non-trivial to have an implementation that performs well in practice, leading to a need of developing efficient software packages. XGBoost [7] is the most widely used package for training GBDT, which uses an optimized sort-and-scan algorithm to find the exact best split on each leaf. Recently, LightGBM [8] proposes to use histogram-building approach to speed up the leaf split procedure. Although the leaf splits are approximate, it is more efficient than the exact-split method.

Recent works [9, 10] proposed several different approaches for parallelizing decision tree building on distributed systems; however, as an important parallel computing resource, GPU is rarely exploited for this problem. Among these packages, only XGBoost utilizes GPU for acceleration[1], but the speedup is not very significant, e.g., training on a top-tier Titan X GPU is only 20% faster than a 24-core CPU[2]. There are also some other early attempts on building decision trees using GPUs, for instances, CUDATree[11]. All these GPU implementations use a similar strategy (parallel multi-scan and radix sort) to find the best split, which mimics the exact-split method on CPU. They require a lot of irregular memory access and the computation pattern does not fit into GPU's parallelization model well. Thus, they can hardly compete with optimized multicore implementations on modern server CPUs.

In this paper, we present a novel massively parallel algorithm for accelerating the decision tree building procedure on GPUs, which is a crucial step in GBDT and random forests training. Previous GPU based tree building algorithms are based on parallel multi-scan or radix sort to find the exact split and thus suffer from scalability and performance issues. We show that using a histogram based algorithm to approximately find the best split is more efficient and scalable on GPU. We develop a fast feature histogram building kernel on GPU with carefully designed computational and memory access sequence to reduce atomic update conflict and maximize GPU utilization. Our algorithm can be used as a drop-in replacement for histogram construction in popular tree boosting systems to improve their scalability. Our results are highlighted as follows:

- We show that histogram based methods for decision tree construction on GPU is more efficient than existing approaches. We design a very efficient algorithm for building feature histograms on GPU and integrate it into a popular GBDT learning system, LightGBM.
- We show significant speedup on large-scale experiments. For epsilon dataset, XGBoost (with exact-split tree builder) takes over 4,100 seconds on a 28-core machine and we only need 165 seconds to achieve the same accuracy using a \$500 GPU, or 300 seconds with a \$239 GPU.
- Our algorithm has superior scalability. The exact-split based GPU implementation in XGBoost fails due to insufficient memory on 4 out of 6 datasets we used, while our learning system can handle datasets over 25 times larger than Higgs on a single GPU, and can be trivially extended to multi-GPUs.

[1]This paper describes the GPU algorithm used in LightGBM; *full version available at* https://arxiv.org/pdf/1706.08359.pdf
[2]http://dmlc.ml/2016/12/14/GPU-accelerated-xgboost.html

2 PROPOSED ALGORITHM

Approximate Split Finding Using Feature Histograms. GBDT constructs a bunch of decision trees in a boosting fashion and the main computation cost is to build each decision tree. When splitting a node in a decision tree, we need to choose a feature and find a good threshold, to split the data examples the left or right child leaves based on their feature values. Finding the exact best split for a feature requires going through all feature values and evaluating the loss function for each of them. For large datasets, it is unnecessary and repetitious to check every possible position to find the exact split location; instead, an approximately best split often works quite well. One way to find the approximate best split is to test only k split positions, and this can be done efficiently using feature histograms. We first convert continuous feature values into k discrete bins, and then construct a histogram with k bins for each feature. To find the split, we can evaluate the loss function only at these k points. Because building histograms is a rather straight-forward process, it is easy to implement efficiently on hardware.

Building Feature Histograms on GPU. Although building a histogram is trivial using a single thread, problems arise when there are a large number of threads computing one histogram. One way to build histogram in parallel is that each thread builds its private histogram using part of the data, and in the end all threads reduce their private histograms into a single final histogram. When the number of threads is large, the reduction step will incur a large overhead. Thus, we want the number of private histograms to be much smaller than the total number of threads. However, if two or more threads update the bin counters of the same histogram, their updates must be atomic. When utilizing hardware atomic instructions, it is important to reduce conflicts for best performance.

Data Structure and Memory Allocation. To avoid the inefficient non-sequential scattered access to the feature array in global memory, we bundle every 4 binned features (one byte each) into a 4-feature tuple (4-byte) and store feature tuples in GPU memory. Each GPU thread will work on 4 features of one sample at once. This strategy also requires that each workgroup maintains 4 set of histograms in local memory, and each set of histogram consists of 3 statistics: gradient, hessian and a counter. Each value takes 4 bytes

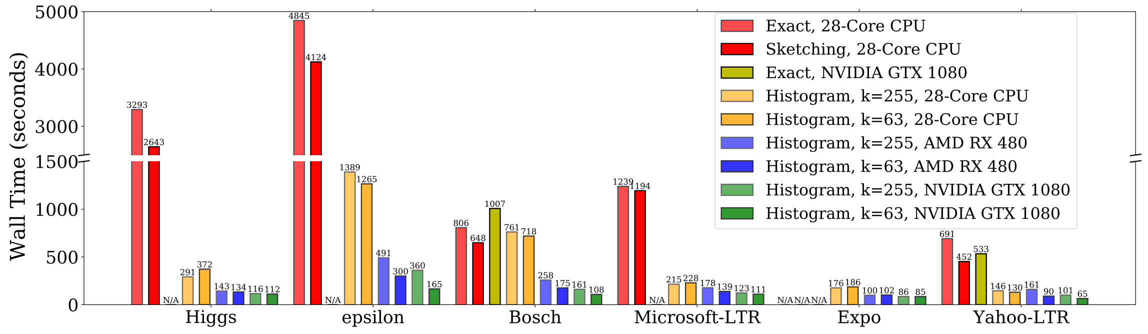

Figure 1: Performance comparison between Histogram **with 63 and 255 bins on CPU and GPU,** Exact **on CPU and GPU, and** Sketching **on CPU.** Exact **on GPU runs out of memory except for** Bosch **and** Yahoo-LTR

(assuming single precision is used), so the total local memory requirement is $4 \times 3 \times 4 \times k$ bytes. When $k = 256$, we need 12 KB local memory per workgroup. This allows 5 workgroups per compute unit of GPU, which is an acceptable occupancy.

Reduce Atomic Update Conflicts. In our GPU algorithm, each thread processes one sample of the 4-feature tuple, and a total of m (m is the size of workgroup) samples are being processed at once. When all m (usually 256) threads update a single histogram with k bins simultaneously, it is very likely that some threads have to write to the same bin because they encounter the same feature value, and the atomic operation becomes a bottleneck since hardware must resolve this conflict by serializing the access. To reduce the chance of conflicting updates, we exploit a special structure that occurs in our feature histogram problem but not in traditional image histogram problem—we construct multiple histograms simultaneously instead of just one. We want m threads to update 8 distinct histograms in each step in a interleaving manner. In this case, m threads are updating $8k$ histogram bins at each step, greatly reduce the chance that two threads write to the same bin.

Use of Small Bin Size. A major benefit of using GPU is that we can use a less than 256 bin size to further speedup training, potentially without losing accuracy. On CPU it is not very beneficial to reduce the bin size below 256, as at least one byte of storage is needed for each feature value. However, in our GPU algorithm, using a smaller bin size k, for example, 64, allows us to either add more private histograms per workgroup to reduce conflict writes in atomic operations, or reduce local memory usage so that more workgroups can be scheduled to the GPU, which helps to hide the expensive memory access latency. Further more, using a smaller bin size can reduce the size of histograms and data transfer overhead between CPU and GPU.

3 EXPERIMENTAL RESULTS

We compare the following algorithms on CPU and GPU:

Histogram: We will compare our proposed histogram-based algorithm on GPU[3] with other methods. LightGBM is a representative CPU implementation of this algorithm and we use it as the reference.

Exact: the traditional method to learn a decision tree which enumerates all possible leaf split points. We use the "exact" tree learner in XGBoost for its CPU implementation, and the "grow_gpu" learner [12] in XGBoost as a GPU reference implementation.

Sketching: the "approx" tree learner in XGBoost, which also uses histogram for approximately finding the split, however features are re-binned after each split using sketching. No GPU implementation is available for this algorithm due to its complexity.

Experiment Setup. We use the following six datasets in our experiments: Higgs, epsilon, Bosch, Yahoo-LTR, Microsoft-LTR, and Expo, representing quite different data characteristics. We follow a publicly available benchmark instruction[4] for setting training parameters, so that our results are comparable to public results. Bin size k is set to 255[5] and 63. The GPU implementation of Exact algorithm in XGBoost only works on Bosch and Yahoo-LTR datasets; other datasets do not fit into the 8 GB GPU memory. In all our experiments, we use two representative, main-stream GPUs from the latest production line of AMD and NVIDIA: Radeon RX 480 ($239 MSRP) and GTX 1080 ($499 MSRP). The two GPUs are installed to a dual-socket 28-core Xeon E5-2683 v3 ($3692 MSPR) server with 192 GB memory. For all CPU results, we run 28 threads. Note that the GPUs we used are not the best ones in the market, and our results can be further improved by using a more expensive GPU. We hope that even a budget GPU can show significant speedup in training, making GPU a cost-effective solution.

Training Performance. We find that training with a bin size of 63 does not affect training performance metrics (AUC or NDCG) on both CPU and GPU. Also, our Histogram based method on GPU can get very similar AUC and NDCG with the one on CPU despite using single precision. Figure 1 shows the training time using different algorithms. On dataset epsilon and Bosch, our speedup is most significant: Using the GTX 1080 GPU and 63 bins, we are 7-8 times faster than Histogram algorithm on CPU, and up to 25 times faster than Exact on CPU. On Higgs, Expo and Microsoft-LTR, we also have about 2-3 times speedup. Even using a low-cost RX 480 GPU (less than half of the price of GTX 1080), we can still gain significant amount of speed up, as it is only 30% to 50% slower than GTX 1080. We should reemphasize that this comparison is made between a powerful 28-core server, and a budget or main-stream (not the best) GPU. Also, Exact on GPU cannot even beat 28 CPU cores on Yahoo-LTR and Bosch, and for all other datasets it runs out of memory. Thus, we believe that the Exact decision tree construction algorithm using parallel multi-scan and radix sort on GPU does not scale well. Our histogram based approach can utilize the computation power of GPU much better.

[3]Our GPU algorithm was first released at https://github.com/huanzhang12/lightgbm-gpu on Feb 28, 2017, and has been merged into LightGBM repository on April 9, 2017

[4]https://github.com/Microsoft/LightGBM/wiki/Experiments

[5]LightGBM uses one bin as sentinel, thus a byte can only represent 255 bins. Similarly, only 63 bins are used when using a 6-bit bin value representation.

REFERENCES

[1] Tin Kam Ho. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR '95, pages 278–, Washington, DC, USA, 1995. IEEE Computer Society.

[2] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[3] Andy Liaw and Matthew Wiener. Classification and regression by random forest. *R News*, 2(3):18–22, 2002.

[4] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

[5] Rie Johnson and Tong Zhang. Learning nonlinear functions using regularized greedy forest. *IEEE transactions on pattern analysis and machine intelligence*, 36(5):942–954, 2014.

[6] Ping Li, Christopher J. C. Burges, and Qiang Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. In *NIPS*, 2007.

[7] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *KDD*, pages 785–794. ACM, 2016.

[8] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3149–3157, 2017.

[9] Firas Abuzaid, Joseph K. Bradley, Feynman T. Liang, Andrew Feng, Lee Yang, Matei Zaharia, and Ameet S. Talwalkar. Yggdrasil: An optimized system for training deep decision trees at scale. In *NIPS*, 2016.

[10] Qi Meng, Guolin Ke, Taifeng Wang, Wei Chen, Qiwei Ye, Zhi-Ming Ma, and Tie-Yan Liu. A communication-efficient parallel algorithm for decision tree. In *NIPS*, 2016.

[11] Yisheng Liao, Alex Rubinsteyn, Russell Power, and Jinyang Li. Learning random forests on the GPU. *New York University, Department of Computer Science*, 2013.

[12] Rory Mitchell and Eibe Frank. Accelerating the XGBoost algorithm using GPU computing. *PeerJ Preprints*, 5:e2911v1, 2017.