Training Random Forests on the GPU: Tree Unrolling

Mark Seligman

Suiji

Introduction

The Random Forest algorithm is a popular tool for predictive analytics. The algorithm is both stable and robust, and plays the role of a Swiss Army Knife for data mining and machine learning.

The Arborist

The Arborist is an open-source implementation of the Random Forest algorithm designed for scability and extensibility.

A common code base allows spins of the Arborist to be created for multiple front ends, such as **R** and **Python** as well as various hardware targets, such as multicore and GPU.

The algorithm, in a nutshell

A "forest" of decision trees is built, each tree predicting an expected outcome (*response*) for a set of observations. Regression outcomes are averaged; categorical are voted on.

A tree is built by sampling the response and performing successive bi-partitions: nodes can be identified with sets of row indices. The process terminates when partitioning exhausts. Two *argmax* criteria determine the bipartition:

- Where, among the sample indices, does a given predictor maximize information content?
- Which predictor holds the overall maximum?

Information content is evaluated at each index in the sample set, walking in *predictor* order.

The "winning" predictor determines a predicate, mapping indices to the left or right subnode.

2 forest types x 2 predictor types = 4 cases

Forests are regression or categorical. Both types support numerical, factor and mixed predictors.

Predictor *argmax* depends on both, so presents four distinct cases:

	Predictor	
Response	Numerical	Factor
Regression	*Index walk*	*Index walk* *Block sort* *Block walk*
Categorical	*Index walk*	*Index walk* *Subset walk*

The four types of index walk are similar, but not identical. Each maintains distinct state, and factors maintain runs as blocks.

Restaging: data locality

Efficient bookkeeping dictates maintaining separate index sets for each predictor at each bipartition.

In *restaging*, sample indices are dispatched to either of two subsets, according to the value of a predicate, while preserving order.

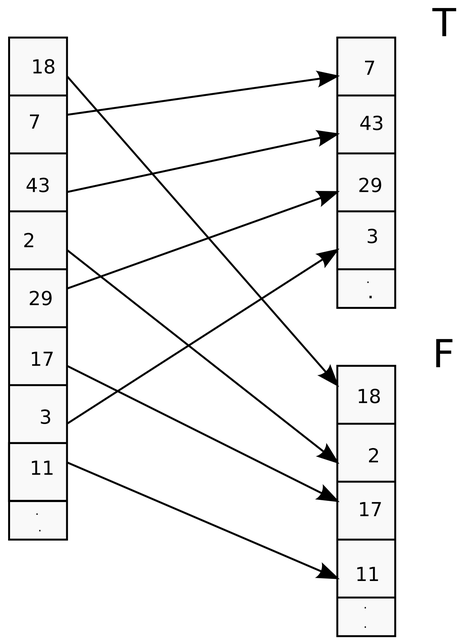
Data locality improves with bipartitioning, as index ranges progressively shrink.

Restaging is independent of both forest and predictor type: a single implementation suffices.

Restaging: concurrency

Restaging is an example of a *stable partition*, which the GPU can parallelize using *scan*.

This occurs for each predictor and node at the current level, offering further opportunities for parallel execution.



Concurrency: argmax and interlevel

The predictor *argmax* operations are embarassingly parallel.

Following a transposition, low memory footprint supports high thread-count at one predictor per thread.

A sequential interlevel pass applies the *argmax* values to define each bipartition.

Left/right index subsets are conveyed to the next level. Although sequential, this step profits from data already residing on the GPU.

Factors and thread divergence

Blocks of runs are walked either in order or by subset: $\mathcal{O}(n)$ vs. $\mathcal{O}(2^n)$.

Factors incur blocks of varying length, leading to thread divergence and requiring algorithmic intervention.

Mixed predictors incur widely divergent index walks. Only the special case of purely numerical data appears immune.

Wide genomic data

We introduced these concepts at GTC 2014, as well as an initial implementation.

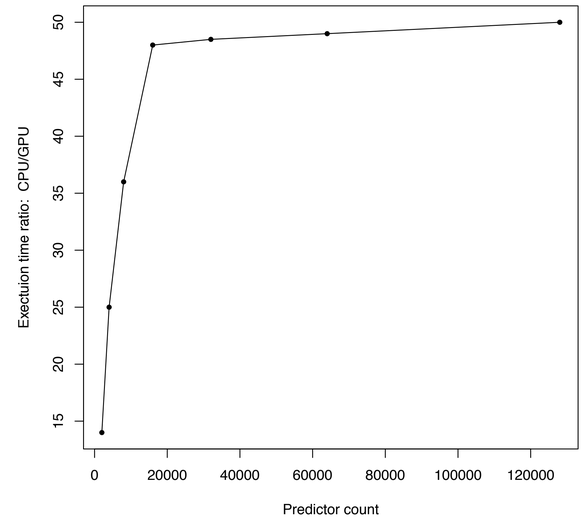
A wide genomic data set was presented, featuring $\sim 10^6$ SNPs from one hundred HIV patients. Factor cardinality was limited to 3, constraining thread divergence.

The implementation coupled the *argmax* and *restaging* phases and yielded minimal acceleration.

The next panel highlights a more recent version, which features *scan*-based *restaging*.

50 x : the case for more predictors

Execution-time ratios between *tuned* multicore and GPU implementations:



With addtional predictors the advantage of the GPU rapidly approaches **50x** for this wide example.

Tall data: parallelizing across trees

Tall data is the more typical use case, often with $10^2 - 10^3$ predictors. Kernels can treat multiple trees to extract more predictor-level parallelism. This is reasonable, as forests usually contain $500 - 1000$ trees.

Implementation

Initial staging data is sent to the GPU for multiple trees, unrolling by a factor suitable for the GPU's memory constraints.

Both *argmax* and *restaging* continue to operate on a per-predictor basis, but with increased parallelism offered by unrolling.

The sequential interlevel pass continues to take place on the GPU, but now reconciles an entire block of tree levels, with workload growing by a similar factor.

This approach introduces some thread divergence toward completion, as tree construction does not terminate uniformly.

Conclusions

Optimal training of a Random Forest varies with the data. For example, a purely numerical/regression implementation can perform *argmax* and *restaging* together as a single pass.

But we seek a general framework to treat all four cases. For this broader goal, the ability to expose more predictors is quite helpful.

Future work

- Continue to pare thread divergence.
- Extend to off-GPU and off-memory sizes.
- Multi-GPU and multi-node implementations.
- General dispatch mapping to appropriate implementation.