LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-522911

Accelerating a random forest classifier: multi-core, GP-GPU, or FPGA?

B. C. Van Essen, C. C. Macaraeg, R. Prenger, M. Gokhale

January 12, 2012

IEEE International Symposium on Field-Programmable Custom Computing Machines
Toronto, Canada
April 29, 2012 through May 1, 2012

Disclaimer

Accelerating a random forest classifier: multi-core, GP-GPU, or FPGA?

Brian Van Essen, Chris Macaraeg, Maya Gokhale and Ryan Prenger
Lawrence Livermore National Laboratory, Livermore, CA 94550
{*vanessen1, macaraeg1, gokhale2, prenger1*}*@llnl.gov*

***Abstract*—Random forest classification is a well known machine learning technique that generates classifiers in the form of an ensemble ("forest") of decision trees. The classification of an input sample is determined by the majority classification by the ensemble. Traditional random forest classifiers can be highly effective, but classification using a random forest is memory bound and not typically suitable for acceleration using FPGAs or GP-GPUs due to the need to traverse large, possibly irregular decision trees. Recent work at Lawrence Livermore National Laboratory has developed several variants of random forest classifiers, including the Compact Random Forest (CRF), that can generate decision trees more suitable for acceleration than traditional decision trees. Our paper compares and contrasts the effectiveness of FPGAs, GP-GPUs, and multi-core CPUs for accelerating classification using models generated by compact random forest machine learning classifiers.**

Taking advantage of training algorithms that can produce compact random forests composed of many, small trees rather than fewer, deep trees, we are able to regularize the forest such that the classification of any sample takes a deterministic amount of time. This optimization then allows us to execute the classifier in a pipelined or single-instruction multiple thread (SIMT) fashion. We show that FPGAs provide the highest performance solution, but require a multi-chip / multi-board system to execute even modest sized forests. GP-GPUs offer a more flexible solution with reasonably high performance that scales with forest size. Finally, multi-threading via OpenMP on a shared memory system was the simplest solution and provided near linear performance that scaled with core count, but was still significantly slower than the GP-GPU and FPGA.

***Keywords*-FPGA; GP-GPU; OpenMP; Machine learning;**

I. Introduction

Random forest classification is a well known machine learning technique [1] in which an ensemble of decision trees is used to assign a label (or classification) to an input sample. Random forest classifiers have been used in myriad application domains ranging from proteomics [2] to ecological studies [3]. In this work we consider hardware acceleration of classification using random forests generated by an off-line machine learning algorithm. Such classification is inherently highly parallelizable since each decision tree processes every sample independently, the only synchronization occurring when the results of all the decision tree are combined to provide a final classification for a sample. However, it is challenging to apply hardware acceleration when the decision trees within the forest vary significantly in terms of shape and depth. This variability makes pipelining and SIMD / SIMT parallelization techniques difficult since the time to process a sample is data dependent. Additionally, this irregularity in tree size and shape makes it difficult to provide deterministic memory access into the tree. Furthermore, the presence of very deep trees within the forest makes it prohibitively expensive to apply techniques that improve regularity, such as fully populating all trees so that the processing time for each sample is identical.

Researchers at LLNL have previously developed an efficient training algorithm that minimizes tree depth to produced a *compact* ensemble of decision trees. The development of a compact random forest (CRF) classifier provides the opportunity to fully populate all decision trees in the forest without dramatically increasing the number of nodes in the forest. This optimization ensures that each classification takes a fixed amount of time and follows a regular execution path. As a result, we are able accelerate sample classification using hardware platforms, such as FPGAs and GP-GPUs, employing tradition pipelining and single-instruction multiple thread (SIMT) techniques, respectively. In this work we will demonstrate that a *compact* random forest makes it possible to accelerate classification with an ensemble of decision trees on an FPGA and GP-GPU. Furthermore, we compare the performance, power-, and cost-efficiency of both FPGA and GP-GPU solutions, enumerating when each provides a reasonable solution. Finally, we identify some of the key parameters in the size and shape of the classification task and relate them to resource trade-offs and hardware constraints on the accelerators.

II. Challenges and opportunities

Classification using decision trees is not compute intensive and is thus not typically considered a good candidate for hardware acceleration. The processing of a single sample through a decision tree within the forest requires one comparison and one data-dependent lookup for each level of the tree. Therefore, the computation to communication ratio is poor. Hardware acceleration platforms are not typically well suited for such pointer chasing algorithms, but they do provide massive bandwidth to a fixed, relatively small, memory. The development of compact random forests (see Figure 1 for an example) makes it possible to fit an entire forest in the memory attached to one or more accelerators and to tap this internal memory bandwidth.

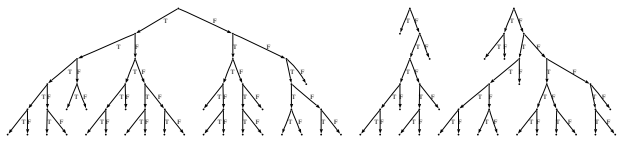

Figure 1. Snippet of a compact random forest with a maximum tree depth of 6 - 3 decision trees shown.

Random forest classifiers are data parallel within the forest and between data samples. Within the forest each decision tree classifies the sample independently and only synchronizes when all of the decision trees are finished processing the sample. Additionally, each sample is independently processed in the forest.

Classification algorithms are optimized according to the number of decision trees in the forest, the maximum depth of any tree in the forest, and the number of features per sample. In this work we will compare three approaches to accelerating classification: using a multi-core chip multiprocessor (CMP), an FPGA, and a GP-GPU. On the CMP we will use the OpenMP library framework for modest thread-level parallelism, *i.e.* one thread per core. On the FPGA, the processing of each decision tree can be executed in parallel by independent hardware and the processing of each tree can be pipelined. The GP-GPU will use massive thread-level parallelism with many threads per streaming multiprocessor.

III. Training compact random forest (CRF) classifiers

The key difference between compact random forests and traditional random forests is in the size (number of trees) and shape (maximum depth of any tree in the forest) of the forests. The CRF training algorithm accepts as a parameter the maximum tree depth, and will generate trees up to that depth, but no deeper. Using this feature in conjunction with hardware accelerators allows us to design processing pipelines of fixed depth and with fixed memory requirements.

Describing the details of the training algorithm used to generate compact random forests is beyond the scope of this paper, as we focus on accelerating the classification phase. Generally, the CRF training algorithm [4] is derived from LogitBoost [5], with a couple of heuristic optimizations to

- improve accuracy by using a linear function of weights associated with leaf nodes of the decision trees, and
- penalize generation of complex trees.

Tables I(a) and I(b) (from [4]) summarize the characteristics of forests generated by traditional (gini) random forest and compact random forest training algorithms respectively. The input training data sets are the two-class classification data sets from the UCI Machine Learning Repository [6].

The "URL Reputation" data set [7] is a set of features derived from URLs that are labeled as either malicious (responsible for spam, phishing, exploits, and so on) or benign. The data set consists of 121 days of data. We used

Table I
Classifier Performance on URL Reputation Data

Max. Depth	# Trees	Accuracy
∞	200	86.7%
60	200	78.3%
50	200	74.0%
40	200	73.0%
30	200	72.1%
20	200	68.2%

(a) Random Forest

Max. Depth	# Trees	Accuracy
∞	18	86.5%
15	11	86.4%
10	16	85.7%
9	17	85.7%
8	18	86.2%
7	24	85.7%
6	32	85.7%
5	47	86.2%

(b) Compact Random Forest

days 0 through 59 as training set and days 60 through 120 as a testing set. This results in 1,176,300 training samples and 1,200,000 testing samples. Both the training and the test set have roughly 32% malicious and 68% benign samples. The data set has a very large number of sparse features, but here we are interested in problems that require a complex model rather than one capable of dealing with a large number of features, so we used only the 64 continuous real features from the data set. The leaves of the tree use a linear combination of weights of the feature used at the parent node and use a logistic function to map that number to a score for the ensemble.

The RF model was run with a split dimension of eight, the square root of the number of features, which is the heuristic suggested for random forests [1]. This means that the training algorithm evaluated eight random features (without replacement) at each interior node of the decision tree, and selected one to be used as the split criterion at that node. The RF model was run with 200 trees to ensure performance saturation. The CRF model determines the number of trees from the data. Both the RF and CRF models can be constrained to different depths.

The CRF results show that the prediction accuracies of the models do not drop consistently as the maximum depth is decreased, and remain nearly as good as the best RF model. As the maximum depth of the trees decreases below eight, the number of trees added by the CRF algorithm increases, as expected. As the individual trees become simpler and less able to reduce training error, the number of trees necessary to reduce the error increases.

To summarize, the innovation of the CRF algorithm is that the training algorithm produces forests with more trees on average, but with smaller average and maximum depth. Furthermore, these trade-offs in size and shape still produce models with modest memory requirements and incur no significant loss in classification accuracy over traditional RF.

Given the opportunity to constrain random forests into a size and shape that is amenable to hardware accelerators, we next describe and evaluate the algorithms used to map CRF classification onto multi-core systems with OpenMP, GP-GPUs, and FPGAs.

IV. Using OpenMP on a shared-memory multiprocessor

In software, the classification task can be implemented as a doubly nested loop that iterates over samples and trees in the forest. The following code snippet shows the kernel of the classification routine that will be used as the basis for the FPGA and GP-GPU designs. Note that this implementation has been designed to work with full trees and does not test for early termination. For performance testing of the multi-core CPU we ran a version that used sparse, irregular trees, which terminated as early as possible. We have found that the most effective way to parallelize this code with OpenMP is to exploit the data parallelism between samples and process each sample independently. To execute the outer `for` loop in parallel we add an OpenMP pragma on first line of the code snippet.

```
#pragma omp parallel for shared(responses)
for (int j=0; j<data->NumVectors(); j++) {
  real response=0;
  for(int i = 0; i < num_trees; i++) { // Forest
    int node_id=1; // Node ID
    real local_result = 0;
    for(int lvl = 0; lvl < max_depth; lvl++) { // Tree
      int node_idx = (i*num_nodes_per_tree)+node_id;
      real threshold = nodes[node_idx].thresholds;
      int feature_idx = nodes[node_idx].indicies;
      real feature = data->FeatureColumn(feature_idx)[j];
      if(lvl < max_depth - 1) { // Split node
        node_id = (node_id << 1) + (feature < threshold);
      }else { // Leaf node
        local_result = nodes[node_idx].weight
                * feature + nodes[node_idx].offset;
      }
    }
    response+=local_result;
  }
  response=1/(1+exp(-response));
  responses[j] = response;
}
```

V. FPGA Implementation

To accelerate the classification on FPGAs, we have transformed the code snippet shown in Section IV into a hardware implementation. Figure 2 shows the general mapping of the algorithm to the FPGA and the general structure that will be partitioned into discrete pipelines both within and between FPGAs. Finally, Figures 3 and 4 shows the implementation of the decision tree's interior split nodes and leaf nodes.

To date, not much work has explored the efficacy of FPGAs for accelerating machine learning classifiers. One notable exception was work by Jiang et al. [8], which used decision trees for classifying network traffic. Due to the nature of network classification rules, they were able to balance the size of the tree by pushing rules up in the tree hierarchy (rule overlap reduction) and to use precise address range cutting. The rule overlap reduction approach was not applicable to compact random forests, which used a linear function of the sample's features at the leaf nodes. The second optimization, precise address range cutting, was a technique used for training the forest and thus was not applicable to the efforts of accelerating classification with a random forest. Becker et al. [9] presented a very small random forest classifier that used 2-bit Binary Patterns for object tracking. It leveraged the limited set of feature patterns to minimize the fanout and depth of the decision trees, which led to a very small FPGA implementation. Another effort related to decision trees was performed by Narayanan et al. [10], but it focused on improving the training rather than the classification.

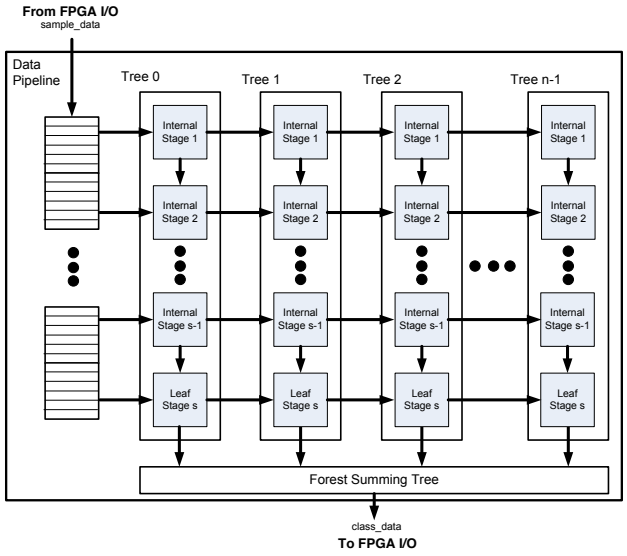

Figure 2. CRF on a FPGA

A. Algorithm Description

Two key parameters in our FPGA algorithm are tree depth and sample width. The tree depth affects FPGA resources because each level of the tree has 2^D nodes at depth D. Data sample width directly affects flip flop usage, especially in a highly pipelined design – flip flop usage in the data pipeline is directly proportional to the data sample width and the depth of the pipeline. In addition, very wide samples can be difficult to multiplex, taxing FPGA routing resources. After a preliminary evaluation of FPGA resources, we settled on a maximum depth of 6, which for this training set required 32 trees. Furthermore, the data set has a sample width of 2048 bits (64 32-bit fields).

For FPGA hardware, we targeted a Hitech Global HTG-V6-PCIE-L240-1 board with a XC6VLX240T-1FFG1759 Virtex 6. To evaluate hardware trade-offs, the design was highly parameterized to allow easy modification of critical CRF core dimensions.

1) Compact Random Forest: Figure 2 illustrates a basic Compact Random Forest implementation in an FPGA. Communication uses an existing, in-house gigabit ethernet core. On startup, forest data is loaded into the tree's pipelines. Once configured, sample data is streamed to the FPGA, where it enters the data pipeline in the CRF core. The core aligns the data with the stages in each tree. The forest's

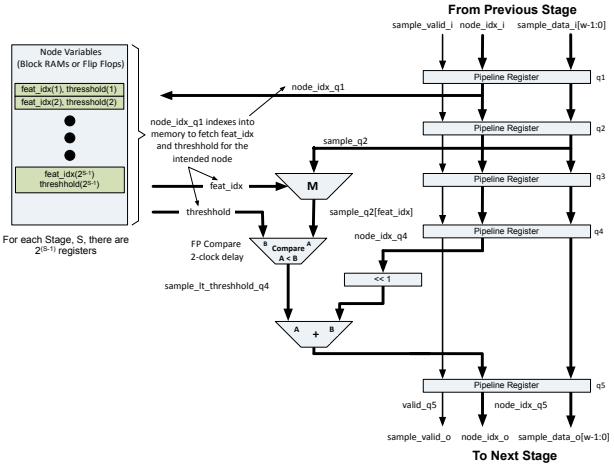

Figure 3. CRF internal stage on a FPGA (implements "split" nodes)

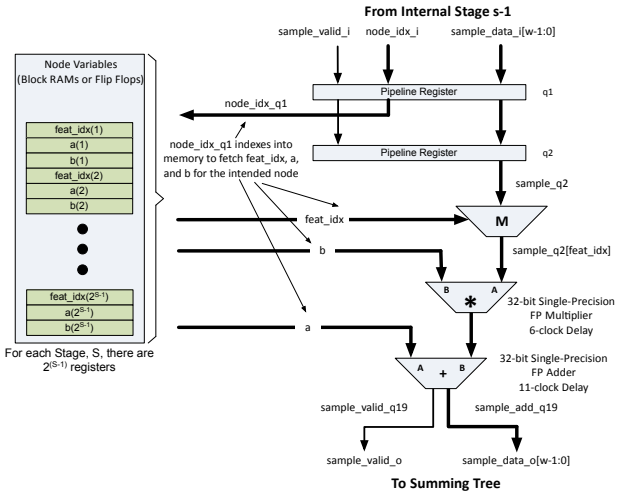

Figure 4. CRF leaf stage on a FPGA (implements leaf nodes)

dimensions are n trees by s stages, where a stage represents all the nodes on a level and will be discussed in more detail below. The outputs of all the trees are summed together, and the result (a 32-bit single precision floating point word) is transmitted back to the host for final analysis. Figure 2 shows an instance of the FPGA design with a single data pipeline shared amongst all the trees. Note that the horizontal arrows represent a point at which the pipeline is tapped so that every tree in the FPGA can use the sample data in parallel.

2) Sample pipeline: using clumps: Within an FPGA a wide data pipeline distributed to multiple destinations could easily create routing issues; in the CRF, this problem is enhanced due to the extreme width of the data pipeline - 2048 bits. With a large CRF, the issue is readily apparent - more trees mean each distribution point in the pipeline (where a stage taps off the pipeline to analyze the sample) must be routed to more logic, while taller trees mean more points of distribution, both of which compete for routing resources with the remaining logic on the FPGA.

To better control the routing of the pipelines, and routing of the samples from the pipelines to the trees, the concept of a clump is introduced. A clump is a subset of the trees in the CRF which share a unique sample data pipeline; a clump's architecture is identical to the basic CRF architecture shown in Figure 2, except that a CRF could comprise multiple clumps, and an external summing tree would be required to sum the outputs of each clump. To create a complete CRF, one clump containing all the trees in the CRF could be instantiated, or multiple clumps each containing as few as one tree could be instantiated. Changing the number of clumps allows precise control over the number of pipelines in the FPGA, and thus the fanout of each clump's sample pipeline. Varying the number of clumps changes the demands on the flip flops, LUTs, and routing resources and will be explored in Section V-B.

3) Decision Tree: The most direct implementation of a decision tree on an FPGA would create a specialized block of logic for every node within the tree. While minimizing memory requirements, this would create a great deal of routing logic. Instead, our design uses a single block of logic at each level (*i.e.* stage) in the tree to implement the functionality of any node at that level. To "customize" a stage to behave like a particular node, a node index from the previous stage is used to address local, unshared memory and load from it node variables that describe the behavior of the current stage. The tradeoff is that additional pipelining for the sample data is required to allow non-blocking access to the local memory. The architecture of the internal and leaf stages are shown in Figures 3 and 4, respectively.

4) Decision Tree Node: The tradeoff between using block and distributed storage for node variables was more complex than the design of the decision tree stages. Specifically, low-level stages (early in the tree) require significantly less storage than higher-level stages, as the node count doubles at each subsequent level. Internal stages require a feature index and threshold value for each node, while leaf stages require a feature index, multiplication factor, and offset value for each node. In order to meet the performance metric of one sample per clock cycle, a stage cannot block on a memory access and thus does not time-multiplex storage structures. While BRAMs are efficient, they are a scarce resource and underutilized by low-level stages. Conversely, flip-flops are plentiful, but inefficient for the large fan-in (-out) at higher stages. We found that the best split was to use BRAMs for stages with 32 or more nodes and logic for the other stages.

In addition to BRAM, DSP slices were used to compute the final weighted summation. The selected FPGA (XC6VLX240T) contains 768 DSP48E1 digital signal processing slices which can be used to implement floating point functions. For each floating point adder, two DSP48E1s were used; for each multiplier, three were used. Comparisons within the interior split nodes used a comparator core that didn't require DSP48E1 slices.

Table II
EIGHT-TREE AND SIXTEEN-TREE FORESTS, 6 STAGES PER TREE, 2048-BIT SAMPLES, ON XC6VLX240T-1 & XC6VLX550T-2, RESPECTIVELY

	XC6VLX240T-1 (Target Frequency = 100 MHz) (Max. Registers = 301,440 Max. LUTs = 150,720)			XC6VLX550T-2 (Target Frequency = 100 MHz) (Max. Registers = 687,360 Max. LUTs = 343,680)		
FPGA Usage & Stats	1 clump x 8 trees/clump	2 clumps x 4 trees/clump	4 clumps x 2 trees/clump	1 clump x 16 trees/clump	2 clumps x 8 trees/clump	4 clumps x 4 trees/clump
Slice Registers	48,642	73,218	122,370	62,660	87,236	136,388
Slice LUTs (total)	76,731	98,275	139,348	122,328	143,527	185,904
P&R Stage Completed	bitgen	bitgen	mapper	mapper	bitgen	bitgen
Mapped Freq. (MHz)	64	69	N/A	N/A	57	79
CAD P&R Time (min)	775	398	232	155	848	1049

B. Exploratory Place and Routes

Given the demands on the routing resources, flip flops, and floating point cores, it was apparent that that even a modestly sized CRF would not fit on a single FPGA. Using the modular clump design, we targeted placing 8 trees (max depth 6) of a CRF on a single LX240 FPGA that we had in the lab and 16 trees (depth 6) on a LX550T. The LX550T-2 was the largest device available in the Virtex 6 family that would not require design modifications. The number of clumps was varied from 1, 2, or 4 (the number of clumps can only be configured as a power of 2), and the number of trees per clump was 8, 4, or 2 respectively. To find the upper limit of the design the CRF core was mapped to the FPGA with a target frequency of 100MHz, even though it would be I/O bound at that frequency. The results are shown in Table II. Increasing the number of clumps is a design tradeoff which reduces the amount of routing required from each pipeline to its associated trees, at the cost of increasing the number of flip flops utilized (due to the duplicated pipelines). The overall impact of introducing clumps to the design typically allows the tools to achieve a higher clock rate at the expense of using more flip flops and LUTs. With 8 trees on the LX240, too many clumps prevented the design from routing, but with 16 trees on the LX550 adding clumps allowed the design to route. The impact of clumps on overall CAD place and route time was erratic, in some cases it helped, others it hurt.

Unlike many floating point FPGA designs, the design was constrained by routing resources for the input sample and memory resources for tree depth, rather than DSP slices. On the LX240, the 4 clump configuration uses a large amount of hard FPGA resources (Slice LUT usage exceeded 92%). Coupled with the 2048-bit sample width, which taxes routing resources in any configuration, PAR did not complete, with a large number of signals left unrouted. Conversely, fewer clumps results in fewer pipelines and fewer utilized hard resources to compete for routing resources. Therefore, the 1 and 2 clump configurations were able to complete their routes, but were unable to meet the requested timing of 100MHz. The target board used for testing has a LX240T-1 FPGA, which is the slowest speed grade available for that device; when the fastest speed grade of the same device (-3) is used, the timing is easily met for the 1-clump and 2-clump configurations. On the LX550T, the best results occurred with 4 clumps; having 4 pipelines did not tax the registers or LUTs, but the routing resources were still stressed - the processing time was exceptionally long, and timing did not meet the 100MHz target frequency, instead reaching 79MHz.

C. CRF Performance on FPGA

The performance of the CRF core is highly deterministic with respect to the number of samples it can process per clock cycle. This characteristic was achieved by using a high-bandwidth design that is synchronous, non-blocking, and highly-pipelined. The performance of the CRF core was tested using Modelsim, while the entire CRF FPGA infrastructure was validated using Xilinx ISim. With both environments, it was verified that the CRF core can process samples at the rate of one sample per clock cycle indefinitely.

VI. GP-GPU ALGORITHM

Setting up the classification task for thread level parallelism is the first step for accelerating on the GP-GPU. Using the code snippet from Section IV as a starting point, we identify the key challenges for executing the code efficiently on a GP-GPU as 1) extracting enough thread parallelism, and 2) selecting the best location in the storage hierarchy to hold the input data (samples) and the CRF model (forest). Figure 5 shows both the memory hierarchy of the GP-GPU and the distribution of data and computation for the CRF classification. Note that texture caches and L1 caches are both private to each streaming multiprocessor; furthermore, the texture caches are optimized for unaligned single word accesses, and the L1 cache is optimized for coalesced multi-word accesses.

Unlike the multi-core kernel segment in Section IV, using just a single thread per sample is insufficient to exercise the capabilities of a modern GP-GPU. To increase the amount of available work, each thread processes a sample on a portion of the compact random forest. Partial results are locally aggregated and then combined by the host processor. To improve data locality, threads are assigned to streaming multiprocessors (SM) such that each SM on the GP-GPU only traverses a subset of the decision trees in the forest. During steady state execution, each SM uses independent threads to process a small number of samples in parallel on a portion of the compact random forest. For example, Figure 5, shows SMs 0 and 1 evaluating samples 0 and 1 on trees 0 and 1 and 2 and 3, respectively.

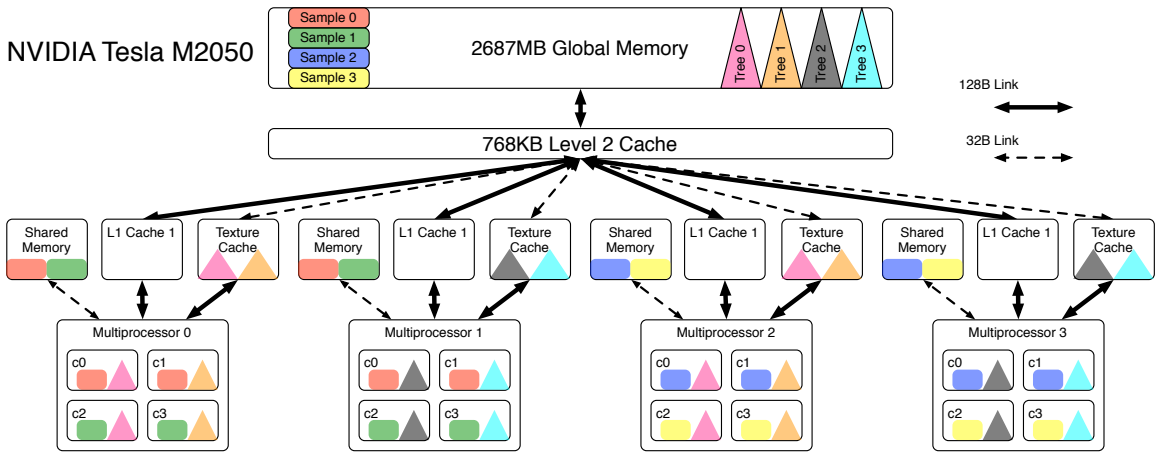

Figure 5. CRF on a GP-GPU.

When optimizing the data movement within the GP-GPU we have two classes of data to consider, forest data and sample data. As noted previously, the forest data for CRFs is actually fairly small, can be loaded once for the lifetime of classification, and is reused for every sample. The sample data is asynchronously input as a blocked stream, it has many features per sample and is only reused when processing each tree in the forest.

Maximizing the reuse and locality of data the CRF trees is challenging because processing each sample down a decision tree is inherently data dependent. However, the CRF trees are not too large and require only a small amount of data per level. Therefore we are able to take advantage of the texture fetch engines and the texture caches. We found that allowing each multiprocessor to evaluate the entire forest thrashes the cache and reduces performance. Therefore we block the design so that each multiprocessor only evaluates four decision trees of the CRF. Figure 5 illustrates an example where trees 0 and 1 processed on SMs 0 and 2 and trees 2 and 3 are processed on SMs 1 and 3. (Partial summations are combined on host.) In this design the private (per SM) level 1 cache is not used since the texture cache provides better access latency and bandwidth for random, uncoalesced reads. This approach of fully populating decision trees, packing the forest data into texture memory, and converting control-flow into data-dependent memory accesses is similar to Sharp's work in [11]. Our implementation extends these concepts to work with larger forests by blocking the design to improve the effectiveness of texture caching.

The sample data is asynchronously moved in batches from the host memory into the GP-GPU's global memory. To process each sample a data-dependent subset of the features within the sample will be required for comparison to the branches within the decision trees. To minimize the cost of moving sample data we load each sample from global memory into shared memory using wide, coalesced loads to pull samples in 128 bit chunks (Figure 5). While this will typically pull in unused features for each sample, it minimizes the number of memory requests and makes the data dependent lookup target shared memory, rather than global memory. Note that for classification problems with a very small number of features per sample (*i.e.* $\leq$ 12) using private registers in the SM is better than the shared memory. However, when there are a reasonable number of features per sample using shared memory, rather than private registers, improves thread occupancy. With multiple threads all concurrently evaluating multiple samples, the data organization in the SM's shared memory is optimized for parallel 32 bit access, *e.g.* one feature per sample, per thread, per bank.

VII. Evaluation

To evaluate the performance of our accelerators we used the URL dataset published by Ma et al. [7], as noted in Section III. The hardware platform for both the multi-core and GP-GPU tests was a 2-socket Intel X5660 Westmere system with 12 cores running at 2.8 GHz, 96 GB DRAM, and an attached NVIDIA Tesla M2050 that had 3 GB GDDR5. The FPGA system was a Hitech Global HTG-V6-PCIE-L240-1 as described in Section V. As noted earlier, we used the CRF training algorithm as a black box, and we focused solely on the classification task. We selected a maximum tree depth of 6 since that produced a forest of a reasonable size that fit well into our FPGA framework.

Reported performance measurement for the CPU and GP-GPU platforms is based on elapsed time with the following caveats. To exclude disk seek access times, the performance for the multi-core system was measured for the classification loops once the CRF and sample data files were loaded within the program. For the GP-GPU we also allowed the program to open the files and load them into memory, but performance did include the time required to transfer all of the data over the PCIe-bus and back.

Given that we were unable to assemble the 4 PCIe boards required for an entire FPGA-based classifier we did not use

end-to-end wall clock time to calculate performance. Instead we tried to tease out the performance of the classification task itself, including system delays as was reasonable. We first ran an end-to-end test on a partial forest, using a single FPGA board and an existing in-house Gigabit Ethernet core to validate correctness. The overall performance was I/O bound as expected, and throughput for the test matched expectations given the relatively low bandwidth of a GigE core. For the purposes of comparison we evaluated the FPGA CRF core assuming that it was matched with a PCIe-2.0 x16 core that could deliver 8GB of data bandwidth and thus a sample rate of 31.25 Msamples/s. The assumed I/O interface places the FPGA design on a competitive level with the NVIDIA M2050. The final performance and power characterization of the FPGA-based systems was computed using a 31.25 MHz system frequency to keep up with the I/O rate.

A. Performance

To test the multi-core and GP-GPU systems, we wanted a large enough sample set to amortize any pipeline fill and drain issues, so we created a 4.7M sample set by looping the original 1.2M input samples. Using this set of 4.7M samples on a CRF with 32 trees and 6 levels per tree we were able to classify samples at the following rates, reported in Kilo-Samples per second (KSps):

- CPU: 9,291 KSps (12 threads) & 884 KSps (1 thread)
- GPU: 20,398 KSps (14 SMs w/ 1536 threads per SM)
- FPGA: 31,250 KSps (w/ 4 LX240s)

B. Power and Cost

To provide a first-order comparison of the power efficiency of each accelerator we computed the ratio of classification rate to power consumed for the FPGAs, the GP-GPU plus GDDR5 DRAM, and the CPUs. The GDDR5 DRAM was included since the GP-GPU actively uses it for the classification, but we chose to exclude the system DRAM since it was common to all designs. We were unable to use a power meter to directly instrument the multi-core and GP-GPU compute node in our production environment, or fully assemble an FPGA-based system. Therefore, we relied upon the data sheet specifications for the NVIDIA GP-GPU and Intel CPUs, and a power estimator for the FPGAs. The power envelope of the Tesla M2050 is listed as $\leq 225W$ for the entire card. The Intel Westmere-EP X5660 processor is listed as having a TDP of $95W$ (note that hyperthreading was administratively disabled).

Xilinx' XPower Estimator (XPE) 13.2 was used to determine the power consumed by the various CRF FPGA configurations listed in Table II that successfully completed place and route. Table III shows the results of the XPower calculations at a clock frequency of 31.25 MHz, all signals at a toggle rate of 50%, and the optimization field was set to "Timing Performance".

Table III
CRF COST AND POWER CONSUMPTION @ 31.25MHZ, FROM XPOWER ESTIMATOR 13.2 (FPGA PRICES ARE FROM DIGIKEY WEBSITE FOR 1759-BALL COMMERCIAL PACKAGES 01/10/12)

FPGA[1]	No. Clumps/ Total No. Trees Per FPGA	Power Consumption of One FPGA (Watts)	Cost ($)
XC6VLX240T-1	1/8	2.88	1941.25
XC6VLX240T-1	2/8	3.20	1941.25
XC6VLX240T-3	1/8	2.88	1941.25
XC6VLX240T-3	2/8	3.20	1941.25
XC6VLX550T-2	2/16	5.45	5444.40
XC6VLX550T-2	4/16	6.10	5444.40

To put these results in perspective, the performance cost, in terms of both power and dollars, of a 32-tree Compact Random Forest was calculated as shown in the last two columns of Table IV and discussed in Section VII-D. Note that these tables contain MSRP prices for the GP-GPU and CPU, and list price of the component cost for the FPGAs required to create a CRF of 32-trees. Furthermore they do not take into other costs associated with the broader FPGA-based system.

C. Scalability

One challenge with machine learning algorithms is that the size of the compact random forest is dependent on the complexity of the input training set and the desired level of classification accuracy. We know that the FPGA based system can be scaled out to support moderately large forests by adding hardware. To test how the performance of the GP-GPU kernel would scale we forced the CRF training algorithm to dramatically increase the number of trees in the ensemble. This resulted in a CRF with 234 trees and 6 levels per tree that was able to classify samples at the following rates, reported in Kilo-Samples per second (KSps):

- CPU: 1,044 KSps (12 threads) & 93 KSps (1 thread)
- GPU: 5,381 KSps (14 SMs w/ 1536 threads per SM)
- FPGA: 31,250 KSps (w/ 8 LX240s)

We see that performance of the GP-GPU decreases with the increased work, but not by the seven-fold increase in number of trees in the forest. Furthermore, the GP-GPU kernel provided this scalable level of performance without any change to parameters or implementation. Finally, the advantage of the GP-GPU versus multi-core CPU with OpenMP was greater with the larger forest than with the smaller one.

D. Tradeoffs: Analysis

Looking at the performance, estimated power and costs we can compare the effectiveness of both accelerators in terms of samples per second per Watt and samples per second per dollar. However, note that without a full end-to-end FPGA system and more precise power numbers from the GP-GPU and CPU systems, these comparisons are only approximate. Despite those caveats, these approximations

Table IV
CRF PERFORMANCE VS. POWER CONSUMPTION AND COST - CRF WITH 32 TREE FOREST WITH MAX. DEPTH OF 6.

Accelerator	No. Clumps/ Total No. Trees Per FPGA	# FPGAs, GPUs, or CPUs	Classification Rate (Ksamples/s)	Accel. Power (W)	Accel. Cost ($)	CRF Performance/ Power ((Ksamples/s)/Watt)	CRF Performance/Cost ((Ksamples/s)/$)
XC6VLX240T-1	1/8	4	31,250	12	7765	2604	4.0
XC6VLX240T-1	2/8	4	31,250	13	7765	2404	4.0
XC6VLX550T-2	2/16	2	31,250	11	10889	2841	2.9
XC6VLX550T-2	4/16	2	31,250	12	10889	2604	2.9
Tesla M2050	N/A	1	20,398	225	2699	91	7.6
Intel X5660	N/A	2	9,291	190	2440	49	3.8

can provide an effective roadmap for implementing a random forest classification system. Examining the results of Table IV we draw the following conclusions:

- FPGAs offer the highest level of performance and performance per Watt.
- FPGAs are built to support a maximum CRF size and require additional hardware to scale to larger classifiers.
- GP-GPUs offer the best performance per dollar and more than twice the performance of CPUs.
- GP-GPUs performance gracefully degrades with larger classifiers.
- GP-GPUs still have hard resource bounds that are sensitive to classifier or sample size.
- Multi-core CPUs with OpenMP are extremely simple to get scalable, near linear performance.

Overall, real-time classification systems typically have a minimum performance requirement. Once that requirement is met further performance is often unnecessary. The challenge of assembling any accelerator based system is that the exact shape and size of the classifier's random forest is unknown until it is trained, and it can evolve over time as the training is refined. Therefore, while the minimum level of required performance will dictate if an FPGA is required or if a GP-GPU is suitable, the scalability of a GP-GPU based system would provide a more flexible system deployment.

VIII. SUMMARY AND CONCLUSIONS

In this work, we have demonstrated acceleration algorithms to speed up classification using a compact random forest classifier on multi-core CPUs, GP-GPUs, and FPGAs. A key contribution of this work is exploiting the low tree depth of the CRF model to regularize the decision trees and thus be able to create pipelined and SIMT algorithms. Further we have related aspects of memory hierarchy such as size, shared access, alignment, and access pattern to the classification algorithm parameters of forest dimension, tree shape, and sample size. We have presented parameterized designs, and carefully evaluated the space of feasible designs, leading to the notion of clumping groups of trees to improve routability in the FPGA. We have identified alternative mappings to be used for different forest and sample sizes. Finally, we have quantified the performance, power, and cost of each implementation, yielding measurable criteria to be used in selecting a platform for this classification task.

ACKNOWLEDGMENTS

This work was partially performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-522911). Funding partially provided by LDRD 10-SI-013. Portions of experiments were performed at the Livermore Computing facility resources.

REFERENCES

[1] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[2] G. Izmirlian, "Application of the random forest classification algorithm to a seldi-tof proteomics study in the setting of a cancer prevention trial," *Annals of the New York Academy of Sciences*, vol. 1020, no. 1, pp. 154–174, 2004.

[3] D. R. Cutler *et al.*, "Random forests for classification in ecology," *Ecology*, vol. 88, no. 11, pp. 2783–2792, Nov. 2007.

[4] R. J. Prenger, B. Y. Chen, D. M. Merl, T. D. Lemmond, and W. G. Hanley, "Fast map search for compact additive tree ensembles," LLNL, Tech. Rep., 2012, (in progress).

[5] "Java Class LogitBoost," http://weka.sourceforge.net/doc/weka/classifiers/meta/LogitBoost.html.

[6] A. Frank and A. Asuncion, "UCI machine learning repository," 2010, http://archive.ics.uci.edu/ml.

[7] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Identifying suspicious urls: an application of large-scale online learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: ACM, June 2009, pp. 681–688.

[8] W. Jiang and V. K. Prasanna, "Large-scale wire-speed packet classification on fpgas," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. New York, NY, USA: ACM, 2009, pp. 219–228.

[9] T. Becker, Q. Liu, W. Luk, and G. Nebehay, "Hardware-Accelerated Object Tracking," in *Computer Vision on Low-Power Reconfigurable Architectures Workshop, Field Programmable Logic and Applications (FPL)*, 2011.

[10] R. Narayanan, D. Honbo, G. Memik, A. Choudhary, and J. Zambreno, "An fpga implementation of decision tree classification," in *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, April 2007, pp. 1 –6.

[11] T. Sharp, "Implementing Decision Trees and Forests on a GPU," *Computer Vision–ECCV 2008*, vol. 5305, no. Lecture Notes in Computer Science, pp. 595–608, 2008.