Towards Machine Learning on the Automata Processor

Tommy Tracy II[1⋆], Yao Fu[2⋆], Indranil Roy[3], Eric Jonas[4], and Paul Glendenning[2]

[1] University of Virginia, Charlottesville, VA, USA
tjt7a@virginia.edu
[2] Micron Technology, Inc., Milpitas, CA, USA
alfu@micron.com
pglendenning@micron.com
[3] Micron Technology, Inc., Boise, ID, USA
iroy@micron.com
[4] University of California, Berkeley, CA, USA
jonas@ericjonas.com

Abstract. A variety of applications employ *ensemble learning models*, using a collection of decision trees, to quickly and accurately classify an input based on its vector of features. In this paper, we discuss the implementation of such a method, namely Random Forests, as the first machine learning algorithm to be executed on the Automata Processor (AP). The AP is an upcoming reconfigurable co-processor accelerator which supports the execution of numerous automata in parallel against a single input data-flow. Owing to this execution model, our approach is fundamentally different, translating Random Forest models from existing memory-bound tree-traversal algorithms to pipelined designs that use multiple automata to check all of the required thresholds independently and in parallel. We also describe techniques to handle floating-point feature values which are not supported in the native hardware, pipelining of the execution stages, and compression of automata for the fastest execution times. The net result is a solution which when evaluated using two applications, namely handwritten digit recognition and sentiment analysis, produce up to 63 and 93 times speed-up respectively over single-core state-of-the-art CPU-based solutions. We foresee these algorithmic techniques to be useful not only in the acceleration of other applications employing Random Forests, but also in the implementation of other machine learning methods on this novel architecture.

Keywords: Automata Processor, Machine Learning, Random Forest

1 Introduction

Recent research has shown that tree-based ensemble models, in particular Random Forests [3], are fast and accurate models of classification for a wide range

⋆ Both authors contributed equally to this work.

of applications including bioinformatics [11], computer vision [4], and sentiment analysis [19]. As data rates climb, accelerating the classification rate of these models is critical, but also presents a variety of challenges. While the non-uniform memory access patterns of tree traversal algorithms result in memory-bound CPU-based implementations, execution divergence while traversing different paths in the tree(s) prevents multiple threads on Single Instruction Multiple Data (SIMD) accelerators such as GPGPUs from executing in parallel.

Although parallelization on contemporary Multiple Instruction Multiple Data (MIMD) machines like CPU clusters is possible due to the independent computability of the individual trees, achieving good load balancing remains a challenge owing to different tree depths, deepest of which determines the overall runtime. Additionally, broadcasting a *feature vector* for every input to all the processors often makes the execution communication bound.

In spite of the above mentioned challenges, the classification rate is an important design metric for Random Forest-based applications. As the training and optimization is typically completed offline, the rate of classification determines the actual runtime performance of the algorithm. For example, the web search engine described by Asadi et al. [2] uses Random Forests in its innermost loop. Therefore, accelerating this loop for a search engine that processes billions of queries per day has a significant effect on the perceived latency experienced by the user. Similarly, a more efficient implementation leads to reduced power, infrastructure and cooling costs for the service providers.

The Automata Processor (AP) is a new non-Von Neumann processor based on the Multiple Instruction, Single Data (MISD) architectural taxonomy. It can compute thousands of user-defined Nondeterministic Finite Automata (NFAs) against a single data stream, in hardware and in parallel. We assert that this is an ideal architecture for accelerating Random Forests, because the values from a single feature vector (representing an input sample), need to be evaluated against the threshold conditions captured by the root-to-leaf paths in the decision trees. By creating a separate automaton for all possible root-to-leaf paths, and by executing thousands of such automata in parallel on the AP, significant acceleration may be achieved.

Nonetheless, executing Random Forest models on the AP required overcoming some fundamental challenges hitherto not addressed by previous AP work [12, 13, 17, 20]. First, Random Forest feature values are often expressed as floating point numbers. Unfortunately, neither floating point numbers nor operations are supported on the AP. To address this limitation, we developed a labeling technique to represent floating point numbers using the symbol-space of the AP, and to operate on the same using the supported instruction set. Second, since all automata on the AP consume bytes from the input in the same order, each automaton was designed to process the feature values in a predefined ordered sequence. This deviates from the current Random Forest implementations, wherein the order of access to the feature values is determined by the tree traversal leading to non-uniform memory accesses.

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

Finally, in order to fit all of the automata required for large Random Forest models onto a single AP board, we adopted a compression technique called Automata Folding, which combines the address spaces of multiple features into as few State Transition Elements (STEs) as possible, reducing the automaton's size.

Having overcome the above mentioned challenges, we have expanded the use of this new processor to accelerate applications employing decision tree-based ensemble classifiers. As exemplars, we used these techniques to accelerate two applications: 1) the classification of handwritten digits and 2) characterization of a poster's sentiment behind a *Tweet*, a 140-character long message on the popular online social networking service *Twitter*. We hope that the techniques described in this paper catalyze further research on the acceleration of other applications using Random Forests, as well as other machine learning techniques on the Automata Processor.

The rest of the paper is arranged as follows. In section 2, we briefly review decision trees and Random Forests, as well as the Automata Processor. Then in Section 3, we introduce our techniques to represent Random Forests as a set of automata that can be executed on the AP. In Section 4, we present our evaluation results, and finally we conclude with a discussion on avenues of future research in Section 5.

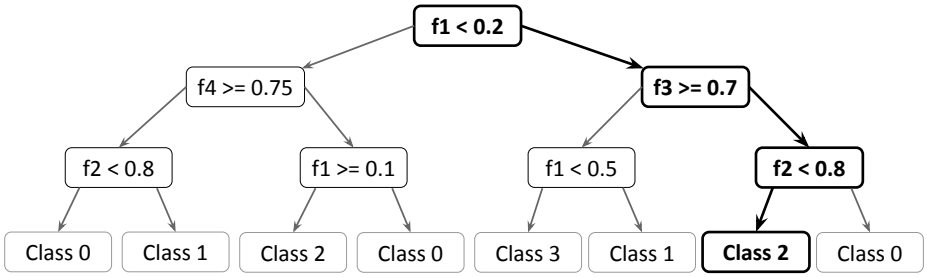
2 Background

2.1 Random Forest

The Random Forest [3] is a supervised classification algorithm. It is an ensemble technique, composed of multiple binary decision trees. Each tree is trained independently by using a random subset, with replacement, of the available training samples. A tree is built by iteratively choosing a *split feature* from a random subset of the feature space, and determining the best threshold value to maximize the entropy reduction per split. This threshold comparison for the split feature is captured as a *split node* in the tree, whose left child corresponds to the next state if the threshold qualification is met, and the right to the contrary. This learning process continues until a maximum depth or minimum error threshold has been met. Each leaf node in a tree represents a classification result. For example, the decision tree shown in Fig.1a can be used to classify an input sample into one of the four classes: $Class$ 0 - $Class$ 3 based on the values of features $f1$-$f4$.

At runtime, a classification of the input sample, represented by a feature vector, is calculated for each tree. Starting at the root node, a root-to-leaf path is traversed based on the values of the features of the input sample. Since each of the split outcomes is mutually exclusive, there is only one root-to-leaf path per tree which can be traversed for any input feature vector. For example, the root-to-leaf path traversed in the decision tree in Fig. 1a for the input feature vector shown in Fig. 1b is highlighted in bold. The sample is therefore classified

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11



(a) A decision tree.

(b) feature-vector

Fig. 1: Classification using a decision tree in the Random Forest.

as belonging to *Class* 2 by this tree. The net classification of the Random Forest is the *mode* of the results from all trees.

Like most machine learning algorithms, Random Forests are trained offline and then optimized for fast runtime classification. Current state of the art implementations run in super-linear time with decision tree depth. This non-linearity arises from the limited locality of the memory access pattern. Computation at each node requires non-uniform access to both the feature vector and the Random Forest model. This is because the split node features are unpredictable, and so are the traversals of the root-to-leaf path for the trees. This unpredictability makes current Random Forest implementations memory-bound, hampering the scalability of the models.

Previous Work The decision trees in Random Forest models are often non-uniform in shape and significant in depth. This makes it impossible to fit an entire decision tree as well as feature vector in the cache memory of modern processors. Therefore, optimizing the traversal algorithm to maximize the spatial and temporal locality has been widely studied.

Essen et al. [16] compare multi-core CPU, GPGPU and FPGA Random Forest implementations for the highest classification rate, performance-to-power and performance-to-cost ratios. They augmented the Compact Random Forest (CRF) design [10] to improve the pipelinability of Random Forests on CPUs, GPUS and FPGAs, and reported improvements in the classification rate by clumping similar trees. Their results show that CRFs computed on FPGAs offer the highest level of performance per watt, GP-GPUs to offer the best performance per dollar, and CPUs to offer the simplest, but lowest performing solution.

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

Researchers have also used ideas from modern compiler and database design to maximize the efficiency of Random Forest models. For example, Asadi et. al [2] use *predication* and *vectorization* to improve the net locality of decision tree traversal to maximize runtime performance. Predication is a technique originating from compiler design to convert control dependencies into data dependencies. Vectorization is a technique originating from database research and batches decision tree computation to mask the cache misses that a sequential algorithm would incur. Although these techniques have shown considerable improvement over existing tree-based solutions, they are only incremental improvements on an algorithm that fundamentally lacks the data-locality necessary for high performance throughput on a von-Neumann architecture.

In [7], Lucchese et. al use an entirely different approach to accelerating additive ensembles of regression trees or a learn-to-rank model by representing tree traversals using bit vectors. Their algorithm, QuickScorer, uses the commutative property of the boolean AND operation to compute out-of-order tree traversals. We use a similar approach, ordering all feature thresholds to be used for simultaneous comparisons, but we pipeline the thresholding, effectively reducing the size of the resulting model, and simplifying the memory footprint. The authors report the fastest run-times to date by reducing the rates of control hazards and branch mispredictions over the previous state-of-the-art VPRED implementation [2].

2.2 Automata Processor

Micron's Automata Processor (AP) [5] is a re-configurable fabric of automata elements. The AP contains *State Transition Elements* (STEs) and *boolean elements* that can be configured to compute a set of Nondeterministic Finite Automata (NFA) in hardware. The AP also contains *counter elements* that give it more power than that available from pure NFAs. The programmer designs their application as automata, which are then compiled and loaded onto the processor.

Automata representation NFAs are represented as homogeneous automata on the AP with STEs and activation edges. STEs represent *states* and their corresponding state transition conditions; activation edges describe activation (transition-enabling) relationships between STEs. STEs with incoming edges from the *start state* are marked as *Start STEs*, and STEs with *final states* are marked as *Reporting STEs*. Start STEs can be configured as *start-of-data* STEs which process only the first symbol of the input data stream, or *all-input-start* STEs which process every symbol in the input data stream.

At runtime, all of the automata are loaded onto the processor, and the input data is streamed in as a *data flow*. This data flow broadcasts one symbol per cycle to all of the AP-chips in an AP *Rank*. On the first clock cycle, only the Start STEs are active which then match the input symbol against the character class of those STEs. If a match occurs, the matched STE activates all STEs connected to its outgoing connections. This process continues on the next cycle.

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

The counter elements and boolean elements may be used to provide additional logic to these activation signals. If in a cycle one or more reporting STEs are matched, then an output is reported identifying the reporting STE(s) and the offset in the data flow where the match(es) occurred.

Programming resources and throughput A single AP chip contains 49,152 STEs, 2304 boolean elements and 768 counter elements. An AP board contains 32 such chips, arranged in 4 ranks of 8 chips each. This cumulatively amounts to over 1.57 million STEs, 73,728 boolean elements and 24,576 counter elements. All of the chips in one rank can receive a broadcast from a single data flow or can be organized into two *logical cores* of 4 chips each. Each logical core processes the data flow at up to 1 Gbps, allowing a maximum data processing rate of 8 Gbps per board. Currently, ongoing work is continuing to increase this throughput to 16 Gbps by allowing logical cores of 2 chips each.

Current Status The AP hardware is accompanied by a Software Development Kit (SDK) [1] which includes design tools to define, visualize, simulate, compile, load, and run user-defined NFAs on the AP. Using these tools, previous work including biological motif search [13], modeling Markov Chains [15], association rule mining [17], and Brill tagging [20] were developed. Although, these works inspired our research, we report results on actual hardware for the first time. In fact, the application for sentiment analysis has been showcased on hardware at the International Supercomputing Conference 2015 (ISC-15) and the Supercomputing Conference 2015 (SC-15), albeit with restrictions on prototype hardware which is currently in the validation phase.

3 Methodology

3.1 Overview

Fig.2 shows an overview of the execution pipeline. The classification process consists of three pipelined stages: labeling, model execution, and voting. In the first *labeling* stage, the floating point *feature vector* is converted into 8-bit labels. The labels corresponding to the feature values are concatenated to form a *label vector* delimited by the # symbol. The label vectors of the inputs serve as the input data flow.

In the second *model execution* stage, the automata loaded on the AP process this data flow in parallel to identify classifications for each tree in the model. Finally, in the *voting* stage, the classifications from all of the trees are combined to generate the final classification using a simple majority-consensus model for each input sample.

Currently, the labeling and voting stages are computed on the CPU and contribute to the overall runtimes. In the future, these will be computed on the FPGA present on the AP board. After porting labeling and voting to the FPGA, the *tree-classification* stage is estimated to become the rate-determining stage of the pipeline, hiding the runtimes of the other stages.

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

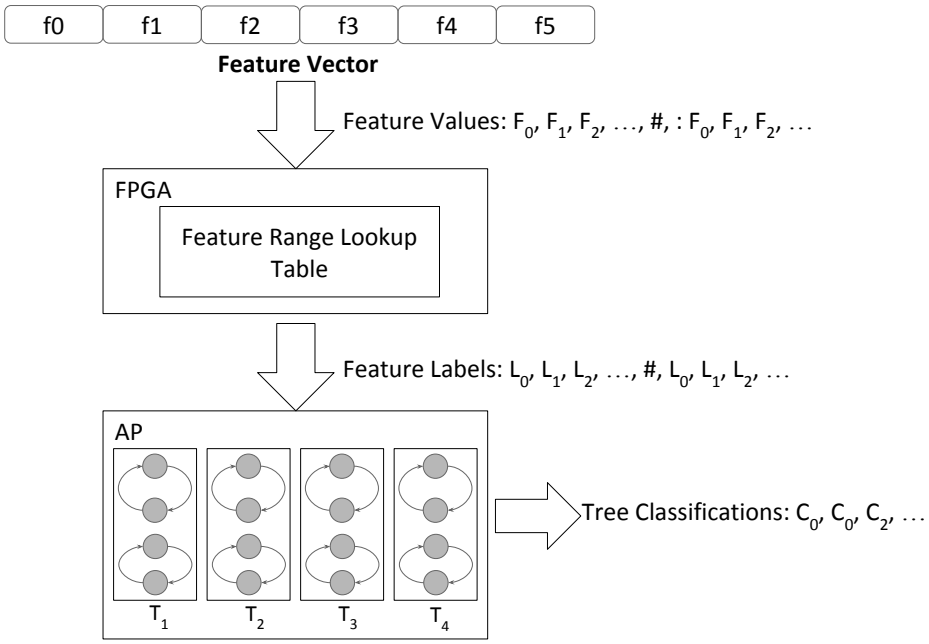


Fig. 2: Three stage execution pipeline

3.2 Automata Design

Given the AP's execution model, the most expeditious implementation of the Random Forest algorithm is obtained by representing each decision tree with one or more automata processing the same data flow in parallel. In order to achieve this we represent each root-to-leaf path in every decision tree in the Random Forest as a chain automaton, and execute all of the chain automata concurrently. In this section, we describe our techniques to generate such automata by overcoming three fundamental challenges. First, the feature vector values across all of the automata must be accessed on the same clock cycle(s). Secondly, a method to handle floating point numbers for feature values and split thresholds must be devised as no native support is present in the hardware. And thirdly, a compression technique must be adopted to arrest the expanded representation of all root-to-leaf paths in the trees. Throughout the rest of the section, We have used the decision tree shown in Fig.1a as our running example to illustrate our techniques.

3.3 Enabling Parallel Execution of Decision Trees

We represent each root-to-leaf path in a decision tree as a chain of feature *evaluationnodes*. Each evaluation node represents one side of the decision tree's split node, and all possible paths are translated into chains. Because the evaluations are complete and exclusive, any feature vector will result in a single

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

chain being traversed from top to bottom, and that one chain will return its associated classification. In this way, we've translated a tree-traversal to a set of exact-match automata.

Note that the order of the nodes in a chain does not affect the outcome of the computation, as the boolean AND operation is commutative; for this reason, we are free to re-order them. All automata on the AP must process feature values in the same order. Therefore, as the second step, the nodes in all of the chains are re-arranged in ascending order by feature id as shown in Fig.3.

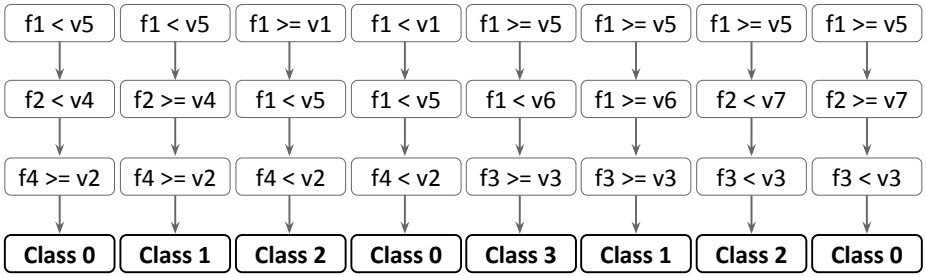


Fig. 3: Reordered chains representation of decision tree shown in Fig.1a.

Next, the nodes representing identical features are combined, and new *satisfy-on-any-value* nodes are introduced for features that are not considered in a chain. The resultant chains are shown in Fig.4. The satisfy-on-any-value nodes are depicted with a $*$ symbol. Notice that, in the resultant chains, all of the features are checked serially and in the same order, and hence these chains can be converted into automata executed in parallel on the AP. However, evaluating the thresholds for floating point numbers still remains a challenge; we discuss our solution next.

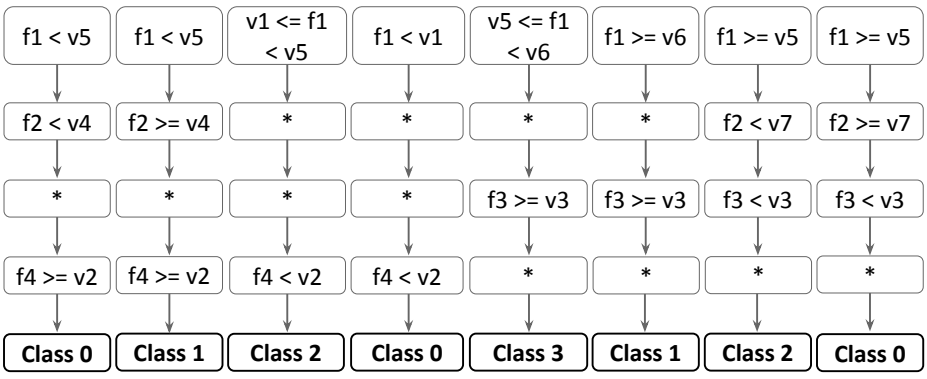


Fig. 4: Completed chains representation of decision tree shown in Fig.1a.

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

3.4 Handling Floating Point Features and Threshold Values

Floating point feature values cannot be directly expressed on the AP, as there is no native support. Scaling down these feature values to the 1-byte symbol space of an STE (or a few STEs) is easy to achieve, but may lead to an unacceptable loss of precision. We formulated an alternate approach by observing that the feature values are only used by the Random Forest to determine which side of a split threshold that value lies. Therefore, it is only necessary to know between which two of the Forest's thresholds a feature value resides. For each feature, we express the address space of floating point numbers as a set of intervals demarcated by split thresholds used for the feature in *all the trees* in the forest. Each interval is then assigned a label. In our case studies, the number of intervals for each feature was always less than 255, and hence a 1-byte label could be used. In cases where this is not true, multi-byte labeling is utilized.

This labeling technique is easy to compute, and leads to a simple automaton design without any loss of precision. Fig.5 shows the labels selected for features $f1 - f4$ of our running example. The address space of feature $f1$ is split using the values $v1, v5$ and $v6$. Therefore, the intervals $(-\infty, v1)$, $[v1, v5)$, $[v5, v6)$ and $[v6, \infty)$ are labeled using 1-byte labels `0x0`, `0x1`, `0x2` and `0x3`, respectively. Similarly, the address-space for feature $f2$ can be labeled as `0x4`, `0x5`, `0x6`; $f3$ as `0x7`, `0x8`; and $f4$ as `0x9`, `0xa`. A later section on Automata Folding will clarify the need for disjoint feature address spaces. Care is taken to avoid labeling an interval with the delimiter symbol `0xff`. Notice that, given a feature value, its corresponding label can be computed in logarithmic time of the number of intervals associated with that feature. This binary-search look up operation is to be implemented on the on-board FPGA in the future.

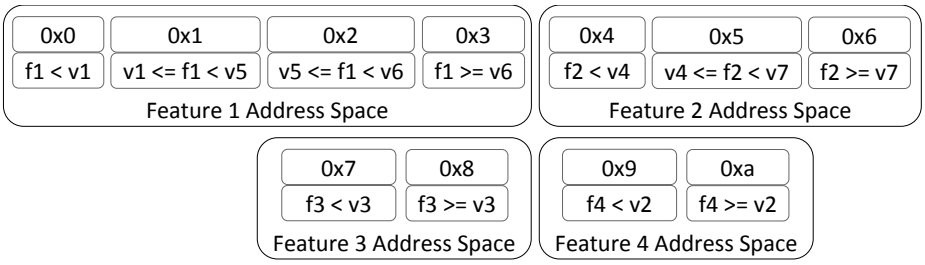


Fig. 5: Feature Address Space

Finally, these chains are ready to be converted into automata that can be executed concurrently on the AP. The resultant automata are shown in Fig.6. The STEs in the automata are depicted using circles with labels placed inside. A Start STE is demarcated using a solid triangle on the top-left corner. The ∞ sign inside the triangle marks the STE as an all-input-start STE. The reporting STEs are outlined using double lines. STEs representing *satisfy-on-any-value*

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

nodes are labeled to match any label for that feature. For example, for feature $f2$, the STEs are labeled as `0x4-0x6`, $f3$ as `0x7-0x8` and $f4$ as `0x7-0x8`.

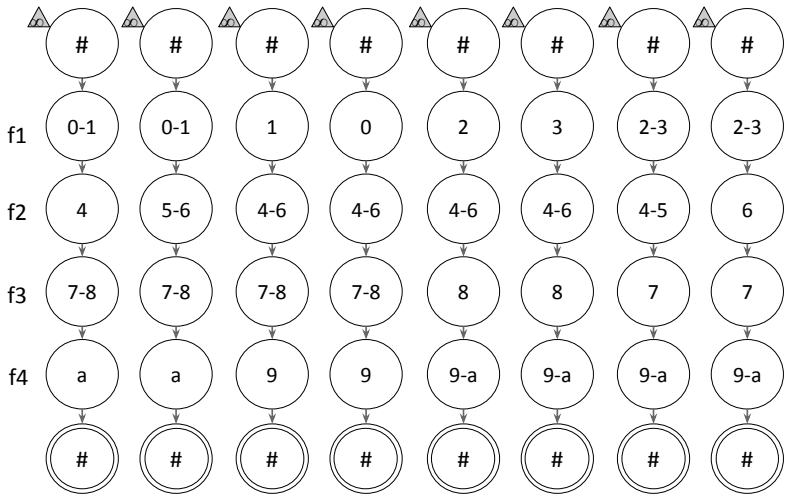


Fig. 6: Chains represented as automata executable on the Automata Processor

For any input sample, the processing of all automata begins at the top Start STE. Because this STE is an all-input-start STE which is active on every clock cycle, it processes the end-delimiter `0xff`, shown as #, at the end of the label vector of the previous input sample and activates the second STE to check the value of feature $f1$ in the next cycle. If the check is successful, the next STE is activated to check the value of feature $f2$, and so on. If the checks for all feature values are successful, then a report is generated by the reporting STE on encountering the delimiter at the end of the label vector.

The report contains the id of the reporting STE, which has an associated classification value. The report also contains the offset in the data flow where the report was generated which is used to determine the input sample associated with the classification. A simple majority of the classifications from all of the trees for an input sample is then declared as its final classification. These simple post-processing steps are scheduled to be migrated to the on-board FPGA.

3.5 Optimizing Automata for higher parallelism

The use of one STE per feature per automaton leads to significant resource requirements, even for moderately sized Random Forests. By realizing that the symbol space of an STE is typically much larger than the number of intervals associated with a feature, we used a compression technique called $AutomataFolding$ to effectively combine features in a single STE. We did this by $folding$ a chain into a loop.

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

Fig 7 shows the folded automata for our running example. The features in a typical Random Forest model have differing numbers of intervals associated with them. In general, the more relevant a feature is to determining the boundary between classifications, the more split nodes for that feature will exist in the forest, and the more intervals assigned to that feature. Assuming that the maximum number of thresholds used by a single feature is less than 255, it is possible to represent multiple features with a single STE!

Automata Folding works by solving the following optimization problem:

$$\min n : \forall i \in [1, n], \sum_{j=0}^{\lfloor m/n \rfloor} f_{nj+i} \leq C \tag{1}$$

Where n is the number of STEs used in the automaton, i is the index of the current STE, f_{nj+i} is the number of intervals assigned to feature $nj+i$, m is the total number of features, and C is the capacity of the STE, 255. This optimization function returns the minimum number of STEs required to represent m features, where the STEs are chained to form a loop. In a simple case where two STEs are required, STE_1 checks feature 1. STE_2 then checks feature 2. STE_1 checks feature 3, STE_2 checks feature 4, and so forth.

Since the total number of labels for all of the features is less than 255[1], we need a single STE to check the labels of all of the features. This STE checks the first symbol of the label vector against the possible labels for feature $f1$. If a match occurs, it activates itself to check the second symbol in the label vector against the possible labels for $f2$ and so on. This is possible because the labels for different features are processed on separate clock cycles and the labels assigned to each feature are disjoint.

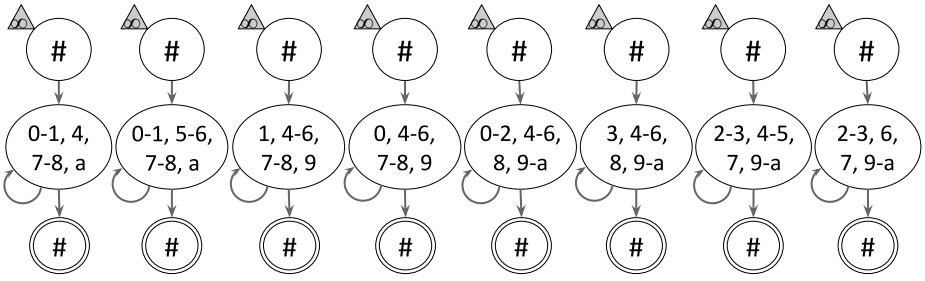


Fig. 7: Combining features into STEs.

[1] the symbol space of an STE minus one symbol reserved for the delimiter

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

4 Experimental Analysis

We implemented our automata-based Random Forest design on the AP as a proof of concept. We then compared our AP implementation against a state of the art Random Forest CPU implementation using two different models trained with differing data sets. The first data set, the MNIST handwritten digits database [6], contains labeled images of handwritten digits, where the classification for each image reflects the representative digit's value from 0 to 9. Each sample is represented by a 28x28 pixel 2-d array representing the image after being centered and scaled. Although we were able to successfully run our design on the AP hardware, because this is a prototype version, we were not able achieve the max performance expected from the hardware.

The second data set, the Sanders Twitter Sentiment Corpus [14], contains one large data set of Twitter messages with their associated sentiments. Three sentiments were considered in the set. The positive, neutral and negative classifications indicate the author's sentiment, while the irrelevant classification is reserved for Tweets in a different language, or those that have no meaning.

These two application data sets were used to train diverse Random Forest models using version 0.16.1 of the Scikit-Learn [9] machine learning framework, with differing tree counts, tree depths, and feature counts. We then took the generated models and converted them into our pipelined automata design to run on the AP. We chose to naively represent the handwritten digits with a 784-wide feature vector, one value per pixel. For the Tweet data, we used TF-IDF (Term Frequency, Inverse Document Frequency) vectorization with an experimentally-determined 1600 feature size.

The Random Forest models from both application were converted into space-efficient chain models that we loaded onto the AP. Knowing the number of STEs per chain, the feature vector size, and the number of trees in the ensemble, we could calculate the throughput of our models on future releases of the AP development board. There are 16 rows of STEs per block, 192 blocks per AP chip, 8 chips per rank, and 4 ranks per AP development board. The input symbol rate is 133 MegaSymbols per second. If the Random Forest model fits on a single rank, we can use inter-rank multiplexing to increase our throughput to 4x that. If the model is small enough to fit into two of chips in a rank, we can use rank logic core multiplexing and achieve an addition 4x speedup, with an effective throughput of 2.128 GigaSymbols per second!

The CPU throughput values were experimentally determined by using Scikit-Learn's Random Forest implementation (Scikit-Learn version 0.16.1) using the same Random Forest models we discussed above.

While benchmarking CPU performance using multi-cores, we found that the performance varies depending on the hardware configuration and the algorithm's parallel efficiency. In our analysis, Scikit-Learns performance did not scale well with core count. For example, with 16 cores, a speedup of no more than 3x was observed. Therefore, in order to provide a more reliable and stable comparison, we chose to use a single thread of the Intel Xeon CPU E5-2630 v3 @ 2.40 GHz processor for benchmarking CPU performance.

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

4.1 Results and Discussion

Random Forests Model Parameters and Accuracy Before comparing the AP and CPU implementations for the Twitter data set, we did a parameter exploration on the number of trees and number of leaves per tree and their impact on the sentiment model's accuracy. The goal was to find the elbow in the graph that maximized accuracy, while reducing the number of trees in the ensemble, and the number of leaves, which relates to the number of split nodes per feature. The experimental results on Twitter data show that the classification accuracy increases as the leaves per tree increases. We found that the maximum accuracy for our model saturated at 72% with 800 leaves per tree. We also found the classification accuracy to increase from 5 to 40 trees, but no more significant increase of accuracy with more than 40 trees.

We performed the same parameter space exploration for the MNIST data set models. Our results show that increasing the number of leaves per tree in the ensemble has a similar effect as with the Twitter data, and we found our elbow with around 1000 leaf nodes per tree. Unlike the other data set, Twitter data models had a significant increase in accuracy when increasing the number of trees in the ensemble from 5 to 160 trees. Our highest experimental accuracy was calculated to be 97.1% and was determined with 160 trees and 4500 leaves per tree (Figure 8) .

Throughput vs. Accuracy Generally, there is a trade-off between throughput and model accuracy for classification models. Random Forest models with fewer and shallower trees can achieve higher throughput, but at lower accuracy. Adding additional trees and training them to be deeper increases the accuracy to the model's saturation; adding any additional resources beyond this point just reduces the efficiency of the model and can yield over-fitting. The goal of model optimization is to find the trade-off between these parameters that maximizes throughput, while still achieving the required level of accuracy. This is often accomplished with design space exploration.

AP vs. CPU Figure 9 and Figure 10 show that throughput is significantly affected by the number of trees in the Random Forest ensemble. As we discussed above, we saturate our accuracy with 40 decision trees, and therefore adding any more is unnecessary. With the same number of trees per model, the AP consistently performs with higher throughput than a single-threaded CPU on Twitter data. Figure 10 shows that, on MNIST data, the AP outperforms the CPU in most of the cases. With the number of trees greater than 20, and a large leaf number per tree (over 4000 leaves per tree), the throughput matches.

The AP architecture allows up to 16x multiplexing if the Random Forest model fits into two chips. As the model size increases, this multiplexing factor is reduced by factors of two (Table 1, Table 2) . The steps in the graph indicate the model dimensions where the hardware cannot sustain the multiplexing factor. Future generations of the hardware will be able to fit larger models, therefore flattening the throughput curve.

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

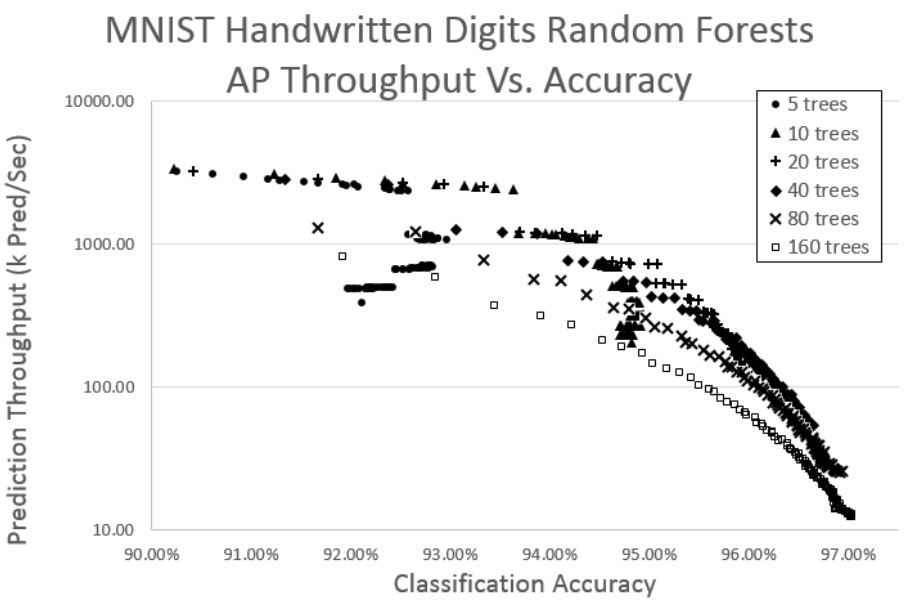


Fig. 8: MNIST Handwritten Digits Random Forests Throughput on AP vs. Accuracy

Table 1: Key data points of Twitter Results

Trees	**Leaves**	**Accuracy**	**AP Throughput (k Pred/Sec)**	**CPU Throughput (k Pred/Sec)**	**AP Speed Up**
5	40	66.9%	14400	154	93
10	40	67.5%	8130	129	63
20	40	67.7%	5360	93.4	57
40	40	68.0%	3750	58.5	64
5	600	70.4%	2010	118	17
10	600	71.4%	1530	86.4	18
20	700	71.7%	385	51.5	7
40	700	71.9%	194	32.4	6

For the Twitter models, the Random Forest implementations on the AP achieve from 2 times to 93 times the prediction throughput of a single CPU. For MNIST, the AP can achieve up to 63 times speed up over the CPU. The speed ups achievable using the AP are more significant with models that have fewer leaves per trees and fewer trees per forest.

The AP is a massively parallel device. With smaller Random Forests models, especially for models with lower numbers of leaves per tree, the AP's advantage of massive parallelism can be greatly taken as it can process hundreds of trees simultaneously. For the smaller models, we were able to achieve results with up to

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

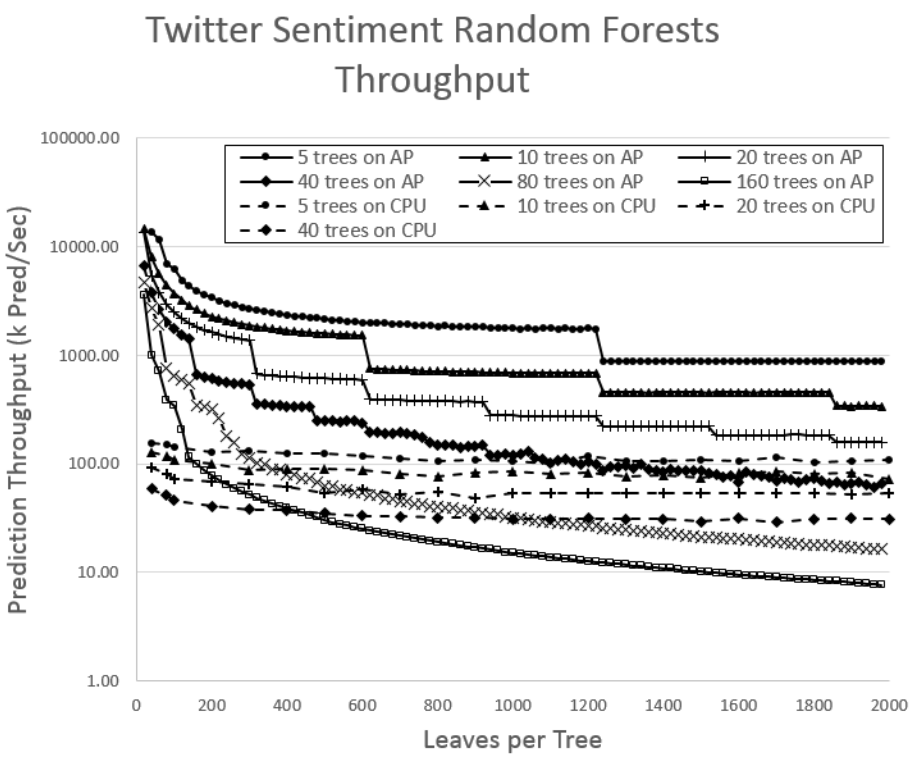


Fig. 9: Twitter Sentiment Random Forests Throughput on AP

Table 2: Key data points of MNIST Results

Trees	Leaves	Accuracy	AP Throughput (k Pred/Sec)	CPU Throughput (k Pred/Sec)	AP Speed Up
5	50	82.2%	13200	337	39
10	50	86.1%	5980	242	25
20	50	87.8%	4170	150	28
40	50	88.7%	3350	86.5	39
80	50	89.2%	2940	46.4	63
160	50	89.6%	1350	25.0	54
10	500	93.3%	2480	205	12
20	500	94.3%	1160	125	9
40	750	95.2%	420	68.0	6
80	1250	96.0%	111	34.3	3
20	4000	96.1%	129	98.9	1.3
40	4750	96.7%	55.0	51.5	1.1
80	5000	96.9%	25.0	26.6	0.9
160	5000	97.1%	12.2	13.5	0.9

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

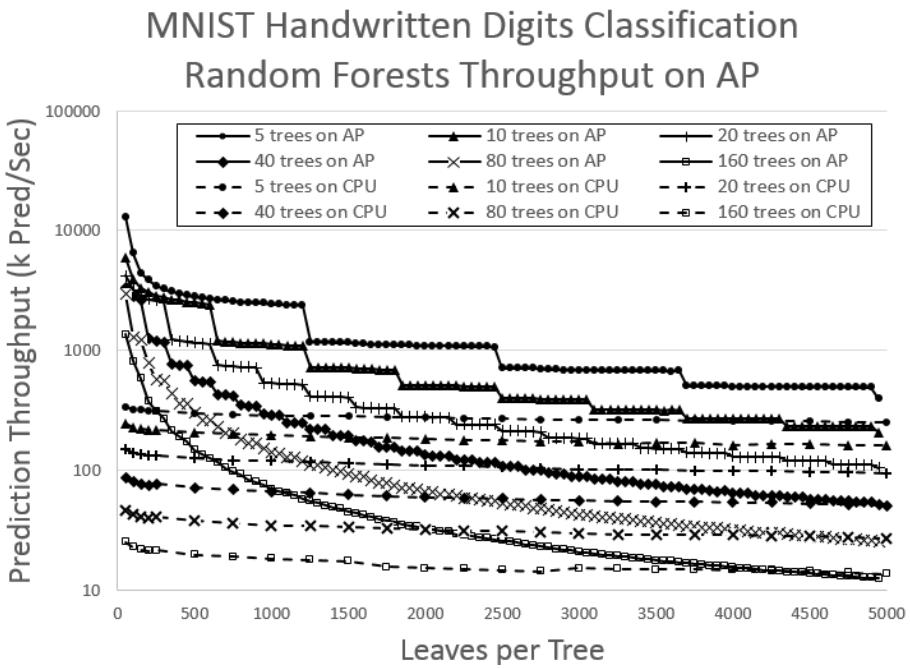


Fig. 10: MNIST Handwritten Digits Random Forests Throughput on AP

93 times speedup against a single CPU thread. The AP's advantage decreases as the number of leaves per tree increase. With significantly decreased parallelism, a higher frequency CPU can reach similar performance as the AP.

With these properties in mind, it's important to focus on compacting ensemble models on the AP to maximize performance. Future applications of Random Forest-like models on the AP should focus on models that require smaller trees, but with large number of trees. As the AP scales with process nodes, we expect the hardware to achieve better scaling.

5 Future Work

5.1 Further Optimizations

Our first generation design addresses many important aspects of computing machine learning applications as automata on specialized hardware. Further optimizations include improving the performance by means of modifying the ensemble models and the automata algorithm. For example, reducing the size of the ensemble algorithms by using Compact Random Forest (CRF) [10] techniques would result in smaller trees utilizing fewer feature values, but achieving similar accuracies. Whereas Essen et al. constrain their CRFs to 6 levels to fit

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

FPGAs, the AP allows us additional flexibility to choose larger tree depths and tree counts that may provide higher accuracy.

There are also potential algorithmic improvements that can be made. A denser binning technique could reduce the number of STEs used per tree, significantly increasing the size of forests that could be supported by one AP board. By potentially further combining feature address spaces, fewer symbols would need to be streamed per feature, improving the throughput of the system.

5.2 Accelerating Other Models

The techniques that we presented in this paper are not limited to Random Forests. Any decision-tree based ensemble technique can be ported to our automata design with little effort by a similar transformation. Some examples of models that could be accelerated include Boosted Trees [18] and Random Ferns [8]. These models are fundamentally similar in their tree traversal technique, but with emphasis on reducing the depth of trees or applying specialized learning techniques.

5.3 Automata on Other Hardware

Our automata design effectively reduces the run time complexity of the Random Forest algorithm by splitting the algorithm into floating point labeling and model computation. Splitting the critical path allows for the algorithm to be pipelined, accelerating the model.

This design could also work well on other hardware platforms including CPUs, GPGPUs and FPGAs. By considerably reducing the size of the Random Forest model, we could increase the cache utilization on a CPU or GPU. The thresholding operation could be computed in parallel on a subset of the available cores, while the model is executed on the remaining cores. Additional future work would include measuring the power-efficiency of this algorithm, and potentially using automata computation on low-power embedded applications. This work is left open for future research.

6 Conclusions

In this paper we present a technique for converting the Random Forest algorithm from a tree-traversal algorithm to a series of pattern matching automata in a pipelined system. This novel algorithm has effectively introduced a new design space for machine learning researchers. Whereby past research has focused on creating shallower decision trees to reduce the latency for tree traversal, our algorithm runs in linear time with the number of features, regardless of the depth of the decision trees! This potentially opens the door for future research into deeper trees on fewer features. We implemented our algorithm on Micron's AP and evaluated the runtime performance of our Random Forest implementation on AP. The results not only showed a promising avenue of applying tree-based

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

ensemble classification methods on AP, but also provide information on the relationship between model settings and the runtime performance on the AP, which can be used to guide future research and development of more efficient classification models.

References

1. The micron automata processor developer portal (Nov 2014), http://www.micronautomata.com/
2. Asadi, N., Lin, J., de Vries, A.P.: Runtime optimizations for tree-based machine learning models. IEEE Transactions on Knowledge and Data Engineering 26(9), 1–1 (2014)
3. Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (2001), http://dx.doi.org/10.1023/A3A1010933404324
4. Criminisi, A., Shotton, J., Konukoglu, E.: Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. Found. Trends. Comput. Graph. Vis. 7(2–3), 81–227 (Feb 2012), http://dx.doi.org/10.1561/0600000035
5. Dlugosch, P., Brown, D., Glendenning, P., Leventhal, M., Noyes, H.: An efficient and scalable semiconductor architecture for parallel automata processing. Parallel and Distributed Systems, IEEE Transactions on 25(12), 3088–3098 (2014)
6. LeCun, Y., Cortes, C.: Mnist handwritten digit database. AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist (2010)
7. Lucchese, C., Nardini, F.M., Orlando, S., Perego, R., Tonellotto, N., Venturini, R.: Quickscorer: A fast algorithm to rank documents with additive ensembles of regression trees. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 73–82. SIGIR '15, ACM, New York, NY, USA (2015), http://doi.acm.org/10.1145/2766462.2767733
8. Ozuysal, M., Fua, P., Lepetit, V.: Fast keypoint recognition in ten lines of code. In: Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on. pp. 1–8 (June 2007)
9. 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, 2825–2830 (2011)
10. Prenger, R., Chen, B., Marlatt, T., Merl, D.: Fast map search for compact additive tree ensembles (cate). Tech. rep., Lawrence Livermore National Laboratory (LLNL), Livermore, CA (2013)
11. Qi, Y.: Random forest for bioinformatics. In: Zhang, C., Ma, Y. (eds.) Ensemble Machine Learning, pp. 307–323. Springer US (2012), http://dx.doi.org/10.1007/978-1-4419-9326-7_11
12. Roy, I.: Algorithmic Techniques for the Micron Automata Processor. dissertation, Georgia Institute of Technology (2015)
13. Roy, I., Aluru, S.: Finding motifs in biological sequences using the micron automata processor. In: Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium. pp. 415–424. IPDPS '14, IEEE Computer Society, Washington, DC, USA (2014), http://dx.doi.org/10.1109/IPDPS.2014.51
14. Sanders, N.: Twitter sentiment corpus (2011), http://www.sananalytics.com/lab/twitter-sentiment/

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11

15. Stan, J.W.K.W.M., Skadron, K.: Uses for random and stochastic input on microns automata processor. Tech. Rep. CS-2015-06, University of Virginia Department of Computer Science, Charlottesville, Va (September 2015)
16. Van Essen, B., Macaraeg, C., Gokhale, M., 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. pp. 232–239. IEEE (2012)
17. Wang, K., Qi, Y., Fox, J., Stan, M., Skadron, K.: Association rule mining with the micron automata processor. In: IPDPS'15 (May 2015)
18. Windeatt, T., Ardeshir, G.: Boosted tree ensembles for solving multiclass problems. In: Multiple Classifier Systems, pp. 42–51. Springer (2002)
19. Zhang, K., Cheng, Y., Xie, Y., Honbo, D., Agrawal, A., Palsetia, D., Lee, K., keng Liao, W., Choudhary, A.: Ses: Sentiment elicitation system for social media data. In: Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on. pp. 129–136 (Dec 2011)
20. Zhou, K., Fox, J.J., Wang, K., Brown, D.E., Skadron, K.: Brill tagging on the micron automata processor. In: Semantic Computing (ICSC), 2015 IEEE International Conference on. pp. 236–239. IEEE (2015)

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_11