Random Forests as Automata on the Automata Processor

TOMMY TRACY II
(UNIVERSITY OF VIRGINIA)

AL FU, INDRANIL ROY, PAUL GLENDENNING
(MICRON TECHNOLOGIES INC.)

ERIC JONAS
(UC BERKELEY)

ISC-HPC
FRANKFURT, GERMANY
21 JUNE 2016
Research Contributions

- Accelerated the Random Forest (RF) algorithm on the Automata Processor (AP)

- Increase the scope of the AP beyond string and graph matching to feature-based classification by:
  - Representing floating point values with labels
  - Efficiently representing large RF models in few states with automata folding
The Automata Processor (AP)

- Memory-based Non-Von Neumann architecture

- Computes multiple Non-deterministic Finite Automata (NFAs) in parallel (MISD)

- NFAs are represented with connected State Transition Elements (STEs)

- The input symbols are 8-bits, and STEs can recognize any subset of the symbol space (bit vector).

- Each automaton has a **start** STE and a **reporting** STE.
  - If a reporting STE is activated, a report is generated.
1. Automata are configured onto the AP, like a bitstream on an FPGA.
2. The input data is streamed to the board, one 8-bit symbol at a time.
3. Any matching final-state automata report to the CPU.
What the AP is Good At

- Can **simultaneously** search for 10s of 1000s of patterns in linear time with the input.
  - A von-Neumann architecture would require sequential search of the patterns.

- AP particularly well-suited for:
  - Regular Expression computing
  - Network traffic analysis
  - Bioinformatics

- Up until now, the AP was used for string pattern and graph pattern matching
  - Feature-based Classification
Random Forest: A General Purpose Model

- A general-purpose, supervised machine learning algorithm.

- Composed of an ensemble of decision trees.
  - The **majority vote** of the constituent weak classifiers serves as the resulting classification.

- Used in many big-data applications including:
  - Image processing
  - Natural Language Processing
  - Bioinformatics
Memory-Bound Decision Trees

- A decision tree prediction is accomplished with a root-to-leaf traversal.
  - At each split node, check a feature value against the node’s threshold.

- This traversal requires non-uniform memory access of both the model and the feature vector.
  - This non-uniformity is exacerbated by multiple trees.
Need for Speed!

- Current implementations coerce the non-uniform memory access into the deep von-Neumann memory hierarchies.
  - The memory bottleneck fundamentally limits the execution time of RF.

- Our approach: **Pipelining**
  - **Labeling**: Reduce the granularity of the input feature vector.
  - **Automata**: Represent tree paths with Non-deterministic Finite Automata (NFAs) and explore all Decision Tree paths simultaneously!
State of the Art

- CPUs execute pointer-chasing algorithms poorly with their hierarchical memory architecture
- GPUs do poorly with conditional jumps
  - High rate of thread divergence
- FPGAs offer highest performance per Watt, but are limited by space
  - Requires multi-chip/multi-board solution to fit a modestly sized forest
  - Also has a steep developer curve
- AP similar performance per Watt as the FPGA, but can fit larger Random Forests
  - Also much easier to develop for

Restructure each Decision Tree into chains
- Each chain represents a path through each tree in the forest.
- Do this for ALL trees in the forest.
Restructuring Decision Trees

- The input is streamed one symbol at a time, and all STEs see the input at the same clock cycle.
  - Need to make sure all chains are ordered
- Features may be used multiple times, or not at all in some chains
Feature Labeling

- Feature values are often floating point values. We need to make a decision at each split node. Is \( f_{v_i} \) value \( \leq \) or \( > \) the split node’s threshold value?

- How do we represent threshold floats with 8-bit symbols?
  - Linear transform fails with high quantization error; instead use `binning`.

- How do we represent ranges of values in the limited STE address space?

- Split nodes only care about what side of a threshold a feature value is. **Represent all ranges between thresholds with labels.**

<table>
<thead>
<tr>
<th>Threshold Values</th>
<th>1.0</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ranges</td>
<td>X ( \leq ) 1.0</td>
<td>1.0 &lt; X ( \leq ) 2.0</td>
</tr>
<tr>
<td>Range Symbols</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Feature i**
Labeling the Input

- Encode feature values on the FPGA
- Parallel log(n) binary search
- STEs now accept ranges of label symbols

Input Symbol Stream: #,1,3,7,8,#,2,3,6,8,#

<table>
<thead>
<tr>
<th>Input Feature Matrix</th>
<th>Output Range Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>X≤1.0</td>
<td>1</td>
</tr>
<tr>
<td>1.0&lt;X≤2.0</td>
<td>2</td>
</tr>
<tr>
<td>2.0&lt;X</td>
<td>3</td>
</tr>
<tr>
<td>X≤1</td>
<td>3</td>
</tr>
<tr>
<td>1&lt;X≤5</td>
<td>6</td>
</tr>
<tr>
<td>5&lt;X</td>
<td>7</td>
</tr>
<tr>
<td>X≤8</td>
<td>8</td>
</tr>
<tr>
<td>8.0&lt;X</td>
<td>8</td>
</tr>
<tr>
<td>X≤10</td>
<td></td>
</tr>
<tr>
<td>10&lt;X</td>
<td></td>
</tr>
</tbody>
</table>

F1 ≤ 2.0
F2 > 5

0,1
5
One Chain per Feature Scales Poorly

**Problem:** One STE per feature scales very poorly for high-dimensional problems!
- If only a few split values are considered, why waste an entire STE’s address space (256 values)?
As long as features have disjoint address spaces, combine features into as few STEs as required.

- Use delimiter (0xFF) to terminate and restart prediction.
  - Only chains with valid feature values will remain active until delimiter

- If there are too many features / thresholds, use Automata Folding to add additional STEs.
Automata Folding to Multiple STEs

<table>
<thead>
<tr>
<th>Features</th>
<th>Sorted Thresholds</th>
<th>Number of Thresholds + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0</td>
<td>1, 1.5, 2, 4, 5, 6</td>
<td>7</td>
</tr>
<tr>
<td>F1</td>
<td>1, 2, 3, 5, 7, 9</td>
<td>7</td>
</tr>
<tr>
<td>F2</td>
<td>4, 5, 6, 7, 8</td>
<td>6</td>
</tr>
<tr>
<td>F3</td>
<td>1, 2, 3, 5, 10</td>
<td>6</td>
</tr>
</tbody>
</table>

![Diagram showing thresholds for F0, F2, F4, F1, F3]
System Overview

- Pipeline labeling with automata execution
  - Use FPGA (onboard) to transform feature values into labels
  - Use AP to compute all valid tree paths in parallel
  - Majority vote is computed on CPU
Experiment Setup

- We tested our implementation with MNIST and Twitter Sentiment analysis using Scikit-Learn.
  - We used one dedicated CPU thread for comparisons.

- MNIST: hand-written numeral recognition
  - Trivially used the 784 pixel values as feature values

- Twitter: determine sentiment of Tweets
  - Used 1600 TF-IDF features

https://twitter.com/twitter
http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html
Experimental Results

- **Twitter**: The AP achieved a max 93x speedup over CPU
- **MNIST**: The AP achieved a max 63x speedup over CPU
Summary

- The AP is more than just a string-matching architecture.
- It can efficiently compute ensemble algorithms
  - Represent floating-point values with a labeling technique
  - Use automata folding to compress automata

Contact me at: tjt7a@virginia.edu
Future Work

- Apply similar transformations to other ensemble techniques including:
  - Boosted Trees
  - Random Ferns
  - Regression Trees
  - Etc.

- Fit larger trees onto the AP with CRFs

- Accelerate Learning of ensemble techniques
Experimental Results

- Graphs above show prediction throughput on CPU and AP
- AP outperforms CPU significantly with smaller RFs
Feature Labeling

Do we really need to compare feature values against feature thresholds at every split node?

We could just figure out all of the threshold ranges we care about, and then represent features with their range values.

\[
\begin{array}{c|c|c|c|c}
F_1 \leq 10 & F_1 \leq 3 & F_1 \leq 1 & F_1 \leq 5 \\
\hline
F_1 \leq 1 & 1 \leq F_1 < 3 & 3 \leq F_1 < 5 & 5 \leq F_1 < 10 & 10 \leq F_1 \\
0 & 1 & 2 & 3 & 4 \\
\end{array}
\]

\[F_1 = 3.8 \quad R_1 = ?\]