Improving Programming Support for Hardware Accelerators Through Automata Processing Abstractions

Kevin Angstadt
angstadt@umich.edu

10. March 2020
By 2020, there will be 40x more bytes of data than there are stars in the observable universe.

DOMO, “Data Never Sleeps 7.0”. 2019
Physical Limits Spark Creativity

Hardware accelerators are seen as a viable path forward for tackling increasing compute demands.

New Kinds of Processors
New Kinds of Processors

Google makes Cloud TPU Pods publicly available in beta
Composed of racks of Google’s custom silicon chips, TPU Pods can take just minutes to complete ML workloads that would take days on other systems, Google said.

Everything you need to know about Apple’s AI chip
By Dave Gershgorn • September 12, 2017

All you need to know about Automata, Micron's revolutionary processor
By Desire Athow • March 28, 2014 • World of tech
Lack of [Good] Programming Models

• Akin to “assembly-level” programming on CPU architectures
  • HDLs are not an emphasis of CS curricula
• Require low-level knowledge of architectural design to produce performant code
• Difficult to debug and maintain: oscilloscopes and logic analyzers
• Many efforts to improve
  • OpenCL, Xilinx SDAccel, etc.
  • High-level language + annotations + decent performance
  • Rarely compiles out of the box
  • Non-intuitive impact of high-level implementation on performance
Successful Programming Models

• **Performance and Scalability**: minimize overhead introduced by high-level programming models and tools.

• **Ease of Use**: provide familiar abstractions and a shallow learning curve.

• **Expressive Power**: support the applications that developers wish to accelerate with dedicated hardware.

• **Legacy Support**: support the adaptation of existing software to execute efficiently on hardware accelerators while placing a minimal burden on developers.
Hardware/Software Co-Design
Finite automata provide a suitable abstraction for bridging the gap between high-level programming models and maintenance tools familiar to developers and the low-level representations that execute efficiently on hardware accelerators.
Automata Processing in the Big Data World

- Detecting Intrusion Attempts in Network Packets
- Learning Association Rules with an *a priori* approach
- Detecting incorrect POS tags in NLP
- Looking for Virus Signatures in Binary Data
- Detecting Higgs Events in Particle Collider Data
- Aligning DNA Fragments to the Human Genome
Finite Automata: 10,000ft View

Key

Active Searches (Automata)

Target Pattern

Incoming Data

Matching patterns trigger reports

ATCGA

CGGCAT

Homogeneous Finite Automata

- Finite set of states with transitions operating over a finite alphabet
- Input data processed by repeatedly applying transition rules
- **Non-determinism**: multiple transitions on single input
- **Homogeneity**: all incoming transitions occur on the same input character
Homogeneous Finite Automata

State Transition Element (STE): a state in a homogeneous NFA

• **Non-determinism**: multiple transitions on single input
• **Homogeneity**: all incoming transitions occur on the same input character
Automata/RegEx Processing Platforms
Automata/RegEx Processing Platforms

- **FPGA-Based**
  - REAPR
  - VASim
  - HyperScan
  - Becchi, et al.
  - PCRE

- **CPU-Based**
  - PAP
  - Micron AP
  - Cache Automaton

- **GPU-Based**
  - DFAGE
  - iNFAnt2
  - IBM PowerEN

- **Custom ASIC**
  - UAP
  - HARE

Dissertation Overview

• Research Contributions
  • Acceleration of Legacy Code
  • High-Level Programming Language: RAPID
  • High-Speed, Interactive Debugger for Hardware Accelerators
  • Hardware Support for New Application Domains:
    • In-Cache Accelerator for Parsing
    • In-Cache Hardware Unit for Detecting Security Attacks

• Broader Impact and Mentorship
• Conclusions / Discussion
Acceleration of Legacy Code (String Functions)

ASPLOS 2020
Legacy Code in the Age of Hardware Accelerators

- Legacy code typically cannot be directly compiled for accelerators
- Learning a new programming model is costly and slows rate of adoption of new accelerators
- May want to “try out” new hardware with existing software
  - No training on new hardware
  - Limited time or resources to allocate
AutomataSynth at a Glance

• Framework for executing code (legacy software) on FPGAs and other hardware accelerators

• Dynamically observe and statically analyze program behavior to synthesize a functionally-equivalent hardware design

• Novel combination of model learning (learning theory), software model checking (software engineering), string decision procedures (PL/theory), and high-performance automata architectures (hardware)
Problem Statement

• Input: function \( \text{kernel} : \text{string} \rightarrow \text{bool} \)

• Assumptions:
  • Function decides a regular language
  • Source code for function is available

• Output: finite automaton with the same behavior on “all” inputs as kernel
Angluin-Style Learning (L*)

Learner \( M \)

Teacher

Oracle

Membership Query

\( s \in L(\text{Kernel}) \)

Termination Query

\( L(M) \overset{?}{=} L(\text{Kernel}) \)

Membership Query

Yes/No Answer

Yes or Counterexample
Angluin-Style Learning ($L^*$)

Learner

Teacher

Oracle

$M$

Minimally Adequate Teacher
Membership Queries are Direct

\[ s \in L(Kernel) \]

- Check if kernel accepts input by **running the code**
- Return value of the kernel is the answer from the MAT
- Caution: take care with ASCII encoding and null terminators (not all functions assume C-style strings)
Understanding Termination Queries

\[ L(M) \overset{?}{=} L(Kernel) \]

• Don’t have held-out automaton for comparison
• Test inputs generally do not suffice
  • Coverage, generation, etc. difficult challenges
• Constraint over string inputs
  • No inputs that are accepted by the kernel are rejected by the candidate machine (and vice versa)
  • “The symmetric difference is empty”
  • Allows for formulation as a software verification query
Equality Checking as Software Verification

- Explores control flow graph looking for property violations
  - Success finding variety of bugs (e.g., double-free, locking violations, etc.)
  - Used in industry for driver verification
- Bounded Model Checking suitable for this domain
  - Verifies that property holds for all program execution up to length k (i.e., fixed number of loop unrollings)
  - Incremental unrolling to check longer and longer executions
  - Use theorem prover to identify executions that violate property
- Wrapper program to encode the “symmetric difference” property
- Add in string solver to generate counterexamples
AutomataSynth System Architecture

- **L* Learner**
- **Membership Query**: \( s \in L(Kernel) \)
- **Termination Query**: \( L(M) \models L(Kernel) \)
- **Learned Automaton** \( M \)
- **Synthesis**
- **FPGA**
- **Software Verifier**
- **SMT Solver**
- **String Solver**
- **Kernel**
- **Mapper**

Membership Query: True or False
Termination Query: True or Counterexample

Caveats

Theorem provers are relatively complete.

- Software verification will occasionally return an unknown result.
- No counterexample is produced, so L* cannot continue.
- Implication: resulting automaton is approximate, but correct for all inputs shorter than some fixed bound.

BMC with incremental unrolling is a semi-algorithm.

- Unrolling of program with infinite loops could continue indefinitely.
- Termination query might never terminate.
- For regular languages finite unrolling suffices (See §3.2.4).
- Implication: BMC+string solver will terminate and satisfies requirements for Termination Queries.
Guiding Research Questions

• How many real-world string kernels can AutomataSynth correctly learn? With approximation?

• Does AutomataSynth learn automata that fit within the design constraints of modern, automata-derived, reconfigurable architectures?
Experimental Methodology

• Mine GitHub for string functions in top C repositories
• Use Cil framework to iteratively parse each source file and extract all string functions
• Filter for duplicates and manual analysis to filter on Boolean return type
• Considered 26 repositories, 973 separate string functions, 18 meaningfully-distinct real-world benchmarks
• AutomataSynth did not support 3 due to functionality of underlying string solver (e.g., no math on characters)
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Project</th>
<th>LOC</th>
<th>Member Queries</th>
<th>Term. Queries</th>
<th>States</th>
<th>Runtime (min)</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>git_offset_1st_component</td>
<td>Git: Revision control</td>
<td>6</td>
<td>4,090</td>
<td>2</td>
<td>2</td>
<td>0.12</td>
<td>✔️</td>
</tr>
<tr>
<td>checkererrmsg</td>
<td>jq: Command-line JSON processor</td>
<td>4</td>
<td>32,664</td>
<td>2</td>
<td>15</td>
<td>1436.58</td>
<td>✔️*</td>
</tr>
<tr>
<td>checkfail</td>
<td>jq: Command-line JSON processor</td>
<td>14</td>
<td>189,013</td>
<td>3</td>
<td>35</td>
<td>1438.47</td>
<td>✔️*</td>
</tr>
<tr>
<td>skipline</td>
<td>jq: Command-line JSON processor</td>
<td>17</td>
<td>7,663</td>
<td>3</td>
<td>3</td>
<td>4.90</td>
<td>✔️</td>
</tr>
<tr>
<td>end_line</td>
<td>Linux: OS kernel</td>
<td>11</td>
<td>510,623</td>
<td>4</td>
<td>44</td>
<td>491.88</td>
<td>✔️</td>
</tr>
<tr>
<td>start_line</td>
<td>Linux: OS kernel</td>
<td>11</td>
<td>206,613</td>
<td>2</td>
<td>46</td>
<td>80.22</td>
<td>Approx.</td>
</tr>
<tr>
<td>is_mcounted_section_name</td>
<td>MASSCAN: IP port scanner</td>
<td>54</td>
<td>672,041</td>
<td>7</td>
<td>57</td>
<td>1439.98</td>
<td>Approx.</td>
</tr>
<tr>
<td>is_numeric_index</td>
<td>MASSCAN: IP port scanner</td>
<td>17</td>
<td>10,727</td>
<td>3</td>
<td>4</td>
<td>4.95</td>
<td>✔️</td>
</tr>
<tr>
<td>is_comment</td>
<td>MASSCAN: IP port scanner</td>
<td>11</td>
<td>4,090</td>
<td>2</td>
<td>2</td>
<td>0.23</td>
<td>✔️</td>
</tr>
<tr>
<td>AMF_DecodeBoolean</td>
<td>OBS Studio: Live streaming and recording software</td>
<td>2</td>
<td>2,557</td>
<td>2</td>
<td>2</td>
<td>0.07</td>
<td>✔️</td>
</tr>
<tr>
<td>cf_is_comment</td>
<td>OBS Studio: Live streaming and recording software</td>
<td>28</td>
<td>4,599</td>
<td>2</td>
<td>4</td>
<td>5.00</td>
<td>✔️</td>
</tr>
<tr>
<td>cf_is_splice</td>
<td>OBS Studio: Live streaming and recording software</td>
<td>22</td>
<td>1,913</td>
<td>2</td>
<td>4</td>
<td>0.05</td>
<td>✔️</td>
</tr>
<tr>
<td>is_reserved_name</td>
<td>OBS Studio: Live streaming and recording software</td>
<td>39</td>
<td>240,705</td>
<td>8</td>
<td>42</td>
<td>1424.48</td>
<td>✔️</td>
</tr>
<tr>
<td>has_start_code</td>
<td>Openpilot: Open-source driving agent</td>
<td>18</td>
<td>10,213</td>
<td>2</td>
<td>7</td>
<td>0.08</td>
<td>✔️</td>
</tr>
<tr>
<td>stbtt_isfont</td>
<td>Openpilot: Open-source driving agent</td>
<td>24</td>
<td>79,598</td>
<td>5</td>
<td>19</td>
<td>0.22</td>
<td>✔️</td>
</tr>
<tr>
<td>Benchmark</td>
<td>Project</td>
<td>LOC</td>
<td>Member Queries</td>
<td>Term. Queries</td>
<td>States</td>
<td>Runtime (min)</td>
<td>Correct</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------------------------------------------</td>
<td>-----</td>
<td>----------------</td>
<td>---------------</td>
<td>--------</td>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td>git_offset_1st_component</td>
<td>Git: Revision control</td>
<td>6</td>
<td>4,090</td>
<td>2</td>
<td>2</td>
<td>0.12</td>
<td>✔</td>
</tr>
<tr>
<td>checkererrormsg</td>
<td>jq: Command-line JSON processor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>checkfail</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>skipline</td>
<td>Linux: OS kernel</td>
<td>4.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>end_line</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>start_line</td>
<td></td>
<td>80.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>is_mc0unted_section_name</td>
<td>MASSCAN: IP port scanner</td>
<td>17</td>
<td>10,727</td>
<td>3</td>
<td>4</td>
<td>4.95</td>
<td>✔️</td>
</tr>
<tr>
<td>is_numeric_index</td>
<td>MASSCAN: IP port scanner</td>
<td>11</td>
<td>4,090</td>
<td>2</td>
<td>2</td>
<td>0.23</td>
<td>✔️</td>
</tr>
<tr>
<td>is_comment</td>
<td>AMF_DecodeBoolean</td>
<td>2</td>
<td>2,557</td>
<td>2</td>
<td>2</td>
<td>0.07</td>
<td>✔️</td>
</tr>
<tr>
<td>AMF_DecodeBoolean</td>
<td>OBS Studio: Live streaming and recording software</td>
<td>28</td>
<td>4,599</td>
<td>2</td>
<td>4</td>
<td>5.00</td>
<td>✔️</td>
</tr>
<tr>
<td>cf_is_comment</td>
<td>cf_is_splice</td>
<td>22</td>
<td>1,913</td>
<td>2</td>
<td>4</td>
<td>0.05</td>
<td>✔️</td>
</tr>
<tr>
<td>cf_is_splice</td>
<td></td>
<td>28</td>
<td>4,599</td>
<td>2</td>
<td>4</td>
<td>5.00</td>
<td>✔️</td>
</tr>
<tr>
<td>is_reserved_name</td>
<td></td>
<td>39</td>
<td>240,705</td>
<td>8</td>
<td>42</td>
<td>1424.48</td>
<td>✔️</td>
</tr>
<tr>
<td>has_start_code</td>
<td></td>
<td>18</td>
<td>10,213</td>
<td>2</td>
<td>7</td>
<td>0.09</td>
<td>✔️</td>
</tr>
<tr>
<td>stbtt__isfont</td>
<td>Openpilot: Open-source driving agent</td>
<td>24</td>
<td>79,598</td>
<td>5</td>
<td>19</td>
<td>0.29</td>
<td>✔️</td>
</tr>
</tbody>
</table>

AutomataSynth learns 13/18 kernels correctly and a further 2 approximately
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Project</th>
<th>LOC</th>
<th>Member Queries</th>
<th>Term. Queries</th>
<th>States</th>
<th>Runtime (min)</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>git_offset_1st_component</td>
<td>Git: Revision control</td>
<td>6</td>
<td>4,090</td>
<td>2</td>
<td>2</td>
<td>0.12</td>
<td>✔</td>
</tr>
<tr>
<td>checkerErrorMsg</td>
<td>jq: Command-line JSON processor</td>
<td>4</td>
<td>32,664</td>
<td>2</td>
<td>15</td>
<td>1436.58</td>
<td>✓*</td>
</tr>
<tr>
<td>checkfail</td>
<td></td>
<td>14</td>
<td>189,013</td>
<td>3</td>
<td>35</td>
<td>1438.47</td>
<td>✓*</td>
</tr>
<tr>
<td>skyline</td>
<td></td>
<td>17</td>
<td>7,663</td>
<td>3</td>
<td>3</td>
<td>4.90</td>
<td>✔</td>
</tr>
<tr>
<td>end_line</td>
<td></td>
<td>11</td>
<td>510,623</td>
<td>4</td>
<td>44</td>
<td>491.88</td>
<td>✔</td>
</tr>
<tr>
<td>start_line</td>
<td>Linux: OS kernel</td>
<td>11</td>
<td>206,613</td>
<td>2</td>
<td>46</td>
<td>80.22</td>
<td>Approx.</td>
</tr>
<tr>
<td>is_mcounted_section_name</td>
<td>MASSCAN: IP port scanner</td>
<td>57</td>
<td>4,62,041</td>
<td>7</td>
<td>57</td>
<td>1439.98</td>
<td>Approx.</td>
</tr>
<tr>
<td>is_numeric_index</td>
<td></td>
<td>4</td>
<td>4,950</td>
<td>2</td>
<td>4</td>
<td>4.95</td>
<td>✔</td>
</tr>
<tr>
<td>is_comment</td>
<td></td>
<td>2</td>
<td>0.23</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>AMF_DecodeBoolean</td>
<td></td>
<td>2</td>
<td>0.07</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>cf_is_comment</td>
<td></td>
<td>2</td>
<td>5.00</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>cf_is_splice</td>
<td></td>
<td>4</td>
<td>0.05</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>is_reserved_name</td>
<td></td>
<td>39</td>
<td>240,705</td>
<td>8</td>
<td>42</td>
<td>1424.48</td>
<td>✔</td>
</tr>
<tr>
<td>has_start_code</td>
<td></td>
<td>18</td>
<td>10,213</td>
<td>2</td>
<td>7</td>
<td>0.08</td>
<td>✔</td>
</tr>
<tr>
<td>stbtt__isfont</td>
<td>Openpilot: Open-source driving agent</td>
<td>24</td>
<td>79,598</td>
<td>5</td>
<td>19</td>
<td>0.22</td>
<td>✔</td>
</tr>
</tbody>
</table>

Learning took an average of 7 hours. More than half take fewer than 5 minutes.
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Project</th>
<th>LOC</th>
<th>Member Queries</th>
<th>Term. Queries</th>
<th>States</th>
<th>Runtime (min)</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>git_offset_1st_component</td>
<td>Git: Revision control</td>
<td>6</td>
<td>4,090</td>
<td>2</td>
<td>2</td>
<td>0.12</td>
<td>✔</td>
</tr>
<tr>
<td>checkerrormsg</td>
<td>jq: Command-line JSON processor</td>
<td>4</td>
<td>32,664</td>
<td>2</td>
<td>15</td>
<td>1436.58</td>
<td>✔</td>
</tr>
<tr>
<td>checkfail</td>
<td>jq: Command-line JSON processor</td>
<td>14</td>
<td>189,013</td>
<td>3</td>
<td>35</td>
<td>1438.47</td>
<td>✔</td>
</tr>
<tr>
<td>skipline</td>
<td>jq: Command-line JSON processor</td>
<td>17</td>
<td>7,663</td>
<td>3</td>
<td>3</td>
<td>4.90</td>
<td>✔</td>
</tr>
<tr>
<td>end_line</td>
<td>jq: Command-line JSON processor</td>
<td>11</td>
<td>510,623</td>
<td>4</td>
<td>44</td>
<td>491.88</td>
<td>✔</td>
</tr>
<tr>
<td>start_line</td>
<td>Linux: OS kernel</td>
<td>11</td>
<td>206,613</td>
<td>2</td>
<td>46</td>
<td>80.22</td>
<td>Approx.</td>
</tr>
<tr>
<td>is_mcounted_section_name</td>
<td>MASSCAN: IP port scanner</td>
<td>54</td>
<td>672,041</td>
<td>3</td>
<td>57</td>
<td>1439.98</td>
<td>Approx.</td>
</tr>
<tr>
<td>is_numeric_index</td>
<td>MASSCAN: IP port scanner</td>
<td>17</td>
<td>10,727</td>
<td>3</td>
<td>4</td>
<td>4.95</td>
<td>✔</td>
</tr>
<tr>
<td>is_comment</td>
<td>MASSCAN: IP port scanner</td>
<td>11</td>
<td>4,090</td>
<td>2</td>
<td>2</td>
<td>0.23</td>
<td>✔</td>
</tr>
<tr>
<td>AMF_DecodeBoolean</td>
<td>OBS Studio: Live streaming and recording software</td>
<td>2</td>
<td>2,557</td>
<td>2</td>
<td>2</td>
<td>0.07</td>
<td>✔</td>
</tr>
<tr>
<td>cf_is_comment</td>
<td>OBS Studio: Live streaming and recording software</td>
<td>28</td>
<td>4,599</td>
<td>2</td>
<td>4</td>
<td>5.00</td>
<td>✔</td>
</tr>
<tr>
<td>cf_is_splice</td>
<td>OBS Studio: Live streaming and recording software</td>
<td>22</td>
<td>1,913</td>
<td>4</td>
<td>4</td>
<td>0.05</td>
<td>✔</td>
</tr>
<tr>
<td>is_reserved_name</td>
<td>OBS Studio: Live streaming and recording software</td>
<td>39</td>
<td>240,705</td>
<td>8</td>
<td>42</td>
<td>1424.48</td>
<td>✔</td>
</tr>
<tr>
<td>has_start_code</td>
<td>stbtt: Openpilot: Open-source driving agent</td>
<td>18</td>
<td>10,213</td>
<td>2</td>
<td>7</td>
<td>0.08</td>
<td>✔</td>
</tr>
<tr>
<td>stbtt__isfont</td>
<td>stbtt: Openpilot: Open-source driving agent</td>
<td>24</td>
<td>79,598</td>
<td>5</td>
<td>19</td>
<td>0.22</td>
<td>✔</td>
</tr>
</tbody>
</table>

Learned automata fall within resource constraints of FPGA-based architectures
AutomataSynth Summary

- Framework for accelerating legacy Boolean string kernel functions using FPGAs
- Constructs a behaviorally-equivalent automaton that can be accelerated with an FPGA
- Novel combination of Angluin-style learning with software model checking and string solvers
- Successfully construct equivalent (or near equivalent) FPGA designs for more than 80% of real-world benchmarks mined from GitHub
- Provides **legacy support** and **performance**
RAPID: A High-Level Language for Portable Automata Processing

ASPLOS 2016, TPDS 2019
RAPID at a Glance

• Provides concise, maintainable, and efficient representations for pattern-identification algorithms
• Conventional, C- or Java-style language with domain-specific parallel control structures
• Excels in applications where patterns are best represented as a combination of text and computation
• Compilation to automata supports execution on AP, FPGAs, CPUs, and GPUs
Parallel Control Structures

• Concise specification of multiple, simultaneous comparisons against a single data stream
• Support common pattern search paradigms
• Static and dynamic thread spawning for massive parallelism support
• Explicit support for sliding window computations
Multi-Architecture System Overview

Focus of this contribution

Appendix A

- Hyperscan Compiler -> CPU Engine
- VASim -> GPU Output
- REAPR -> FPGA Engine
- Micron AP Compiler -> AP Binary
- RAPID Program -> RAPID Compiler
- RAPID Compiler -> Automata
- Automata -> VASim
- REAPR -> Xilinx PAR
Experimental Results (Summary)

• Successfully ported benchmarks from real-world applications (expressiveness)
  • Demonstrated RAPID can implement regular expressions

• 2-8x more compact than scripts generating automata and 13-71x more compact than defining automata (scalability)
  • RAPID size remains constant or grows sublinearly with application instance size

• Overheads of compiled RAPID on FPGA and AP are within 15% of hand-optimized automata (performance)

• Automata optimizations supports more stable porting of applications across architectures than OpenCL (performance)
RAPID Summary

• RAPID allows developers to write new pattern-searching programs for hardware accelerators
  • Programs compile to a set of finite automata
  • Domain-specific parallel control structures support common tasks
  • Programs are significantly more concise than hand-crafted automata

• RAPID programs can execute on a wide variety of hardware platforms (FPGA, AP, CPU, GPU, etc.)
  • Portability of automata provides more stable performance across architectures

• Provides scalability and performance
Interactive Debugging for High-Level Languages and Accelerators

ASPLOS 2019
Houston, we have a problem!

- Unexpected output deep in data processing
- Bug in corner case infrequently activated by input

Incoming Data

CPU too slow to debug full application, but may be difficult to extract subset of input
Where do we stop?

• **Breakpoints** annotate expressions/statements to specify locations to pause execution for inspection
  • Traditional notion relies on instruction streams
  • Mechanism does not apply directly to architectures with no instructions (e.g., FPGAs, AP)

• **Key Insight:** Automata computation driven by input
  • Set breakpoints on input data, not instructions
  • Supports use case of stopping computation at abnormal behavior
  • Can also provide abstraction of traditional breakpoints
Capturing State

• Process input data up to breakpoint
• State of automata is **compact**
  • O(n) in the number of states of the NFA
• Repurpose existing hardware to capture
  • AP: State vector cache
  • FPGA: Integrated Logic Analyzers (ILAs) and Virtual I/O pins (VIOs) allow for probing of activation bits
• Cache state vectors to decrease latency
Bridging the Gap

• Developer can set breakpoint in program or data
• Automatically translate this location to automaton states
• Use hardware test equipment to monitor and read state information in the circuit
• Automatically translate HW information back to program source code
Experimental Results Summary

• Using server-class FPGA and standard ANMLZoo benchmarks, we found debugging required 3x logic elements and 6x register elements of baseline designs (scalability)

• Debugging of ANMLZoo benchmarks can occur at ~80% of baseline clock frequencies (performance)

• Human study of 61 undergraduate and graduate programmers found statistically significant increase in fault location accuracy (ease of use)
  • Debugger aided novices and relative experts alike
Debugging Summary

- Developers are now able to debug on FPGAs using high-level languages (e.g., RAPID)
- Bridge the semantic gap by storing mappings between program expressions and low-level hardware resources
- Leverage Virtual I/O on FPGAs to capture state
  - Automata abstraction produces minimal state to capture
- Provides **performance**, **scalability**, and **ease of use**
Architectural Support for New Application Domains

MICRO 2018, (Current Manuscript)
Two New Application Domains

XML Parsing
• Parsing of data central to most (all?) data processing pipelines
• XML is one of the most common formats
• Parsing notoriously difficult to accelerate

Detecting Security Attacks
• Continual cat and mouse game
• Recent discoveries of hardware bugs leave billions of devices vulnerable
• Current fixes are costly and/or ineffectual
XML Nesting

```xml
<course>
  <footnote></footnote>
  <sln>10637</sln>
  <prefix>ACCTG</prefix>
  <crs>230</crs>
  <lab></lab>
  <sect>01</sect>
  <title>INT FIN ACCT</title>
  <credit>3.0</credit>
  <days>TU,TH</days>
  <times>
    <start>7:45</start>
    <end>9</end>
  </times>
  <place>
    <bldg>TODD</bldg>
    <room>230</room>
  </place>
  <instructor>
    B. MCELDOWNEY
  </instructor>
  <limit>0112</limit>
  <enrolled>0108</enrolled>
</course>
```

JSON Encoding

```json
{
  "id": "0t_15l_5r",
  "type": "hState",
  "enable": "onActivateIn",
  "report": true,
  "inputDefs": [
    {
      "width": 1,
      "portId": "i"
    }
  ],
  "outputDefs": [
    {
      "width": 1,
      "activate": false,
      "portId": "o"
    }
  ],
  "attributes": {
    "reportId": 5,
    "latched": false,
    "symbolSet": "[\\xFF]"
  }
}
```

Parsing

```
s → exp →
   exp → term + exp
   | term
   | ( exp )
   | int

term → int * term
   | ( exp )
   | int
```

10. March 2020

K. Angstadt – PhD Dissertation
Pushdown Automata Refresher
Pushdown Automata Refresher

Input Symbol Match

Top of Stack Match

Stack Actions

Finite State Control

Stack Memory
ASPLEMENT Supports Richer Analyses

- Accelerated in-SRAM Pushdown ENgine
- Scalable processing engine that uses LLC slices to accelerate Pushdown Automata computation
- Custom five-stage datapath using SRAM lookups can process up to one byte per cycle
- Optimizing compiler supports existing grammars, packs states efficiently, and reduces the number processing stalls
- Provides additional cache when not in use
Where is ASPEN?

• ASPEN uses 2 arrays per bank
• 240 states per bank
• Full connectivity within bank
• Global switch and stack in CBOX for large DPDA
XML Parsing Experimental Results

- Benchmarks: Parabix, Ximpleware, UW XML
- ASPEN is $13-18x$ faster (on average) than popular CPU Parsers
- Performance did not vary significantly with complexity of XML
- Optimizations and tokenization hide ε-stalls
The New Way Your Computer Can Be Attacked

Spectre and Meltdown explained: What they are, how they work, what's at risk

Spectre and Meltdown are the names given to a trio of variations on a vulnerability that affects nearly every computer chip manufactured in the last 20 years. The flaws are so fundamental and widespread that security researchers are calling them catastrophic.

By Josh Fruhling

Intel Performance Hit 5x Harder Than AMD After Spectre, Meltdown Patches

By Joel Hruska on May 20, 2019 at 1:46 pm | 255 Comments
Processor Designs are Flawed

TECHNOLOGY

The New Spectre: Security Threats to Core Processor Designs are Flawed

Intel Confirms ‘ZombieLoad 2’ Security Threat

Davey Winder Senior Contributor
Cybersecurity
I report and analyse breaking cybersecurity and privacy stories

Intel Performance Hit 5x Harder Than AMD After Spectre, Meltdown Patches

By Joel Hruska on May 20, 2019 at 1:46 pm 255 Comments
Understanding Spectre and Meltdown

- **Conditional branches** (if statements) are slow
  - Time needed to determine next instruction to execute
- Modern processors use **speculation** to guess the next instruction to execute
  - When the processor guesses wrong, it can “undo” running the speculated instruction
- Modern computers use caches to speed up access to data
  - Processors do not “undo” changes made to cache data as a result of speculation
  - Careful programming combines misprediction and caching to steal information
Anomaly Intrusion Detection

• Fixing hardware takes time and has significant cost
• In the meantime, can we detect attacks with minimal modifications?
• **Key Idea**: Run known good programs many times to **learn correct behavior** of the software
• Use this model to categorize other running programs
• What do we monitor?
  • Previous work monitored sequences of **system (OS) calls**
  • Does not capture the speculative behavior we want to monitor
  • Instead, monitor **sequence of memory accesses**
Basic Approach: Sliding Windows

- Record **sequences of memory addresses** accessed by a program during execution
- **Sliding windows** captures order in the sequences
  - Windows from training stored in a **dictionary**
  - Can be encoded using **finite automata**

Training Memory: AABACCBBC
Basic Approach: Sliding Windows

- Record *sequences of memory addresses* accessed by a program during execution
- **Sliding windows** captures order in the sequences
  - Windows from training stored in a *dictionary*
  - Can be encoded using *finite automata*
Basic Approach: Sliding Windows

- Record **sequences of memory addresses** accessed by a program during execution
- **Sliding windows** captures order in the sequences
  - Windows from training stored in a **dictionary**
  - Can be encoded using **finite automata**
Basic Approach: Sliding Windows

• Record **sequences of memory addresses** accessed by a program during execution

• **Sliding windows** captures order in the sequences
  • Windows from training stored in a **dictionary**
  • Can be encoded using **finite automata**

![Diagram](image-url)
Basic Approach: Sliding Windows

• Record **sequences of memory addresses** accessed by a program during execution

• **Sliding windows** captures order in the sequences
  • Windows from training stored in a **dictionary**
  • Can be encoded using **finite automata**
Basic Approach: Sliding Windows

- Record **sequences of memory addresses** accessed by a program during execution
- **Sliding windows** captures order in the sequences
  - Windows from training stored in a dictionary
  - Can be encoded using finite automata

![Training Memory: AABACCBBC]

![Testing Memory: AAECED]

![Diagram showing memory addresses]
Basic Approach: Sliding Windows

- Record **sequences of memory addresses** accessed by a program during execution
- **Sliding windows** captures order in the sequences
  - Windows from training stored in a **dictionary**
  - Can be encoded using **finite automata**

![Diagram of sliding windows]

Training Memory: AABACCCB

Testing Memory: AAEC
Basic Approach: Sliding Windows

• Record **sequences of memory addresses** accessed by a program during execution

• **Sliding windows** captures order in the sequences
  • Windows from training stored in a **dictionary**
  • Can be encoded using **finite automata**

---

Training Memory

AABACCCBBC

Testing Memory

AAECD

---

A
B
C
D
E

A
B
C
D
E

Basic Approach: Sliding Windows

• Record **sequences of memory addresses** accessed by a program during execution

• **Sliding windows** captures order in the sequences
  • Windows from training stored in a **dictionary**
  • Can be encoded using **finite automata**

![Diagram showing training and testing memory sequences and a grid with memory addresses]
**MARTINI: Detecting Attacks**

- For real-time monitoring, use a small hardware accelerator that executes automata
  - Embedded in the cache of the processor
- Dictionaries need simplification to embed in processor
  - Improve interaction with other security and system abstractions
  - Fit within hardware resource constraints
- Three primary techniques
Δ-windows: to mitigate overfitting due to ASLR, we store the difference between subsequent memory accesses

Truncation: to mitigate difference between physical and virtual addresses, we truncate all deltas to 8 bits
  • Mask selectable
  • Lower 7 + sign bit worked well in our experiments

Compression: to reduce state space (hardware utilization), we represent windows with unordered sets
  • More permissive but smaller
Where is MARTINI?

Simplified routing because automata are chains of 8 STEs
Experimental Methodology

• Benchmark suite of real-world applications and exploits
  • GNU Coreutils programs
  • PARSEC benchmarks
  • Exploits: Spectre, Meltdown, Objdump CVE, DNSTracer CVE
• Use instrumented QEMU and Pin tools to collect traces
• 2,400 program traces and over 13 billion memory accesses
• Simulate MARTINI with custom version of VASim
  • Simulate address delta, automata core, and trigger arbitration units
Guiding Research Questions

• Do sequences of memory accesses differentiate programs?

• Is MARTINI able to detect malicious inputs to trained programs?

• Can MARTINI detect anomalous programs, including recent hardware attacks?
**MARTINI can Detect Attacks**

- Dictionary trained on 60% of Coreutils traces
- Testing on held out Coreutils traces, Spectre, Meltdown, Objdump CVE, DNSTracer CVE
- AUC = 0.9954
- 100% true positive = 4.4% false positives
Architectural Support Summary

• Extend **expressive power** of current automata-derived architectures to support DPDA (and parsing)
• Custom 5-stage datapath for executing DPDA
  • 18.5x faster for parsing XML than state-of-the-art parsers
• Custom datapath for detecting security attacks
  • Based on sliding windows of abstracted memory accesses
  • Can detect recent hardware exploits (e.g., Spectre, and Meltdown)
• Architectures provide **scalability**, and **performance** for new application domains
Dissertation Overview

• Research Contributions
  • Acceleration of Legacy Code
  • High-Level Programming Language: RAPID
  • High-Speed, Interactive Debugger for Hardware Accelerators
  • Hardware Support for New Application Domains:
    • In-Cache Accelerator for Parsing
    • In-Cache Hardware Unit for Detecting Security Attacks

• Broader Impact and Mentorship
• Conclusions / Discussion
Broader Impact

• **Tools to Promote Adoption**: prototype tools used in this dissertation are (being) released as part of MNCaRT, an end-to-end automata processing ecosystem
  - AutomataSynth, RAPID Compiler, DPDA compiler, MNRL state machine language, VASim (+DPDA)
  - Appendix A

• **Undergraduate Mentorship and Teaching**: training the next generation of researchers and introducing students to the beauty of automata
Teaching and Mentorship

- **Automata-based Instruction**: lecture on string algorithms (regex) in DS+A; lecture on language design in grad/undergrad PL; guest lecture on automata processing in grad architecture; guest lecture on accelerator debugging in grad PL

- **Undergraduate Research Projects**:
  - ‡MARTINI (Yujun Qin, Samuel Gonzalez, Linh Le)
  - †Debugging (Matthew Casias, UVA)
  - ‡Automata-based file carving and disk damage modeling (Ian Bertram, Michael Flanagan, Aniruddh Agarwal)
  - Automata-based surface detection (Emma Fass, Luke Merrick, Joe Tidwell, UVA)
  - ‡Diversity in undergrad CS (Fee Christoph)
  - †Quadcopter Security (Kate Highnam, UVA)

‡Peer-Reviewed Publication
†In-Flight Publication
Proposed Research (2018)

- Four components to improve programming support for hardware accelerators using automata abstractions
  - High-level programming language (RAPID)
  - High-speed, interactive debugging for RAPID on AP and FPGA
  - In-cache accelerators
    - For pushdown automata (ASPEN)
    - For detection of security attacks
  - Adapt legacy kernels for execution on hardware accelerators

- Evaluation w.r.t. **Performance & scalability, ease of use, expressive power, and legacy support**
Publications Supporting Contributions


7. *(Under review)* Yujun Qin, Samuel Gonzalez, **Kevin Angstadt**, Xiaowei Wang, Stephanie Forrest, Reetuparna Das, Kevin Leach, and Westley Weimer. MARTINI: Memory Access Traces to Detect Attacks.

Undergraduate collaborators are underlined
Additional Publications


Invited Papers and Tech Reports


*Undergraduate collaborators are underlined*
Dissertation Summary

• Hardware accelerators more commonplace—need for programming models and maintenance tools

• Using finite automata as an abstraction, we developed a programming model that provides performance & scalability, ease of use, expressive power, and legacy support
  • AutomataSynth: porting legacy code to FPGAs
  • RAPID: writing new pattern-searching programs for hardware accelerators
  • High-speed, FPGA-based debugger for RAPID programs
  • Two in-cache accelerators for new applications
    • Parsing of XML and detecting security attacks
Bonus Slides

(Excerpts)
What is a Program?

**Task:** Blink a lightbulb

**CPU (Python)**

```python
import gpiozero
import time

led = gpiozero.LED(14)

while True:
    led.on()
    time.sleep(1)
    led.off()
    time.sleep(1)
```

**FPGA (Circuit)**

Generally written in Verilog or VHDL language
Why Automata(Synth)?

- FPGA designs are often described in terms of state machines
- Automata a versatile and broadly-applicable
- Can build on significant research effort accelerating state machine execution
- Other high-level approaches (cf. HLS) generally fail to abstract low-level architectural details
- Our approach **decouples** high-level program and low-level implementation
Reasoning About Strings

String solver should support the following:

- Unbounded string length
- Regular expression-based constraints over strings
- Access to individual characters of strings
- Comparison of individual characters and strings
- Reasoning about the length of strings
- Comparison between strings and bitvectors (treat characters as numbers)***
- Ability to generate strings that satisfy a set of constraints

***Not currently supported
Example RAPID Program

**macro frequent** (String set, Counter cnt) {
    foreach(char c : set) {
        while(input() != c);
    }
    cnt.count();
}

**network** (String[] set) {
    some(String s : set) {
        Counter cnt;
        whenever(START_OF_INPUT == input())
            frequent(s,cnt);
        if (cnt > 128)
            report;
    }
}
Putting it all together

Standard Program Execution

Accelerator processes data

Abnormal behavior observed

Debugging Execution

Accelerator processes data

System-calculated breakpoint

User-defined breakpoint

Accelerator state vector

Simulator processes data

Simulator state vector

Mapping
Traditional Breakpoints

RAPID Program

```c
macro helloWorld() {
    whenever( ALL_INPUT == input() ) {
        foreach(char c : "Hello") {
            c == input();
        }
        input() == ' ';
        foreach(char c : "world") {
            c == input();
        }
        report;
    }
}

network() {
    helloWorld();
}
```

Accelerator processes data with Machine A

Accelerator processes data with Machine B

Reports occur when line is executed

Input breakpoints inserted at reports
Five Steps of DPDA Execution Per Cycle

1. Input Match
2. Stack Match
3. Action Lookup
4. Stack Update
5. State Transition

01010c01010
ASPEN Datapath — 240 States in 2 SRAM Arrays

1. Input Match
2. Stack Match
3. Action Lookup
4. Stack Update
5. State Transition

- 1. Input Matching — One Column per State
- 2. Stack Matching — One Column per State
- 3. Stack Actions — One Row per State
- 4. Local Stack — One Row per Entry
- 5. Reconfigurable Transition Matrix
Optimizations

Epsilon Merging

Goal: Reduce the number of stalls while processing input

Multipop

• **Average of 65% reduction** in epsilon states

[A-Z]
- Pop 0
- No Push

ε*
- Pop 1
- Push ‘a’

[A-Z]
- Pop 1
- Push ‘a’

ε*
- Pop 0
- No Push

ε*
- Pop 1
- No Push

ε*
- Pop 1
- No Push

ε*
- Pop 1
- No Push

ε*
- Pop 4
- No Push

10. March 2020

K. Angstadt – PhD Dissertation

114
Differentiating Programs

Each point represents a sequence of three addresses accessed by the program.

- `cal` — displays a calendar
- `dmesg` — displays system message buffer
Dictionary contains: \(\{c,a,t\}, \{c,o,t\}, \{c,o,w\}, \text{and } \{d,o,g\}\)

- Sacrifices accuracy for space
  - Now accepts \(\{c,a,w\}\)
  - 9 to 6 states
  - In limit, reduces \(2^{64}\) states to 2,048 states
MARTINI can Differentiate Programs
MARTINI can Detect Anomalous Inputs

Pass rate vs. threshold (Interval size: 20000, decay: 0.50)

- **objdump**
- **CVE-2018-6323**
Both ASPEN and MARTINI consume a portion of LLC

What does this do to total system performance?

**Platform:** Ubuntu 16.04, 192 GB RAM, 2 Intel Xeon Platinum 8275CL CPUs (36 cores, each) @ 3 GHz, 36 MB LLC per processor, subdivided into 11 ways

**Experiment:** execute PARSEC benchmarks 40 times and measure wall clock time (20 with full cache, 20 with reduced cache)
  - Reduce by 1 way to simulate MARTINI footprint
Runtime Comparison with Reduced Cache

- Runtime (ms)
- Full Cache
- Reduced Cache

- blackscholes
- bodytrack
- cannel
- dedup
- facesim
- ferret
- fluidanimate
- freqmine
- netdedup
- netferret
- netstreamcluster
- raytrace
- streamcluster
- swaptions
- vips
- x264

Runtime Comparison with Reduced Cache

Runtime was reduced by 1.42% in the worst case. The differences were not statistically significant.