RAPID Programming of Pattern-Recognition Processors

Kevin Angstadt  Westley Weimer  Kevin Skadron
Department of Computer Science
University of Virginia
{angstadt, weimer, skadron}@cs.virginia.edu
19. August 2016
Finding Needles in a Haystack

• Researchers and companies are collecting increasing amounts of data
• 44x data production in 2020 than in 2009†
• Demand for real-time analysis of collected data‡

† Computer Sciences Corporation. Big data universe beginning to explode. 2012
‡ Capgemini. Big & fast data: The rise of insight-driven business. 2015.
What is the common theme?

Locate the most probable location for a DNA fragment in the human genome

Find products that are most commonly purchased together

Parse English text to identify historical records that are duplicates

Identify consumer sentiment based off social media posts

Search for Higgs events based off on paths of subatomic particles

Pattern Search Problems
Parallel searches

Key

Active Searches

Target Pattern

Incoming Data

CGGCAT

ATCGA
Parallel searches

Key

Incoming Data

Active Searches

Target Pattern

…

CGGGCGCTATAG

0...

ATCGA ×

CGGCAT ×
Parallel searches

Key

Active Searches

Target Pattern

Incoming Data

AGGCTAT....

ATCGA

CGGCAT
Parallel searches

Key

Active Searches

Target Pattern

Incoming Data

ATCGA

CGGCAT
Parallel searches

Key
= Active Searches

Incoming Data

ATGCAT

Target Pattern

A

CGGCAT

ATCGA

Active Searches

TRCTAGGC

Center for Automata Processing
Parallel searches

Key

Active Searches

Target Pattern

Incoming Data

ATCGA

CGGCAT
Parallel searches

Key

Active Searches

Target Pattern

Incoming Data

ATCGA

CGGCAT

C C A T A C G G G

C T A T G

C C A T A C G G G

C
Parallel searches

Key
Active Searches
Target Pattern

Incoming Data

ACCATATCGG

ATCGA

CGCAT

G

CGGCAT
Parallel searches

Key

Active Searches

Target Pattern

Incoming Data

CGG

ATCGA

CGGCAT

G

A

C

C

A

T

A

C

G

G

C

T

A

...
Parallel searches

Key

Active Searches

Target Pattern

Incoming Data

T G A C C A T A

ATCGA

CGGC

TCGA

CGGCAT

Active Searches = Target Pattern

Active Searches

Active Searches

Active Searches

Active Searches

Active Searches

Active Searches
Parallel searches

Key

\[ \text{Active Searches} = \text{Target Pattern} \]

Incoming Data

\[ \ldots C \ T \ G \ A \ C \ C \ A \ T \ A \ C \ G \ G \ C \ \ldots \]

\[ \text{ATCGA} \]

\[ \text{CGGCAT} \]
Parallel searches

Key
= Active Searches

Target Pattern

Incoming Data

G C T G A C C A T

ATCGA

CGGCAT

T

Center for Automata Processing
Parallel Searches: Goals

• Fast processing
• Concise, maintainable representation
• Efficient compilation
  – High throughput
  – Low compilation time

Specialized Hardware

RAPID Programming Language
A researcher should spend his or her time designing an algorithm to find the important data, not building a machine that will obey said algorithm.
The Remainder of this Talk

• Automata Processor
  – Architectural Overview
  – Current Programming Models
• RAPID Programming Language
  – Language Overview
  – AP Code Generation and Optimizations
• Experimental Evaluation
• Conclusions and Future Directions
The Remainder of this Talk

• Automata Processor
  – Architectural Overview
  – Current Programming Models

• RAPID Programming Language
  – Language Overview
  – AP Code Generation and Optimizations

• Experimental Evaluation

• Conclusions and Future Directions
Micron’s Automata Processor

• Accelerates identification of patterns in input data stream using massive parallelism
• Hardware implementation of non-deterministic finite automata
• 1 gbps data processing
• MISD architecture
Micron’s Automata Processor

- Implements homogeneous NFAs
  - All incoming edges to state have same symbol(s)
  - State Transition Element (STE)
- Memory-derived architecture
  - Memory as a computational medium
  - State consists of a column in DRAM array
  - Connections made with reconfigurable routing matrix partitioned into blocks
- 1.5 million states on development board
- Saturating Up Counter, Boolean Logic

Dd o n u t
u g h

Start STE
Reporting STE

*[Dd](o|ough)nut
Micron’s Automata Processor

Figure courtesy of Micron
Programming Workflow

Source: www.micronautomata.com
Current Programming Models

**ANML**
- Automata Network Markup Language
- Directly specify homogeneous NFA design
- High-level programming language bindings for generation

**RegEx**
- Support for a list of regular expressions
- Support for PCRE modifiers
- Compiled directly to binary
Programming Challenges

• ANML development akin to assembly programming
  – Requires knowledge of automata theory and hardware properties
  – Tedious and error-prone development process
• Regular expressions challenging to implement
  – Often exhaustive enumerations
  – Similarly error-prone
Programming Challenges

• Implement **single instance** of a problem
  – Each instance of a problem requires a brand new design
  – Need for meta-programs to generate final design

• Current programming models place unnecessary burden on developer
Goals: Current Approaches Fail

• Fast processing ✓
• Concise, maintainable representation ✗
• Efficient compilation
  – High throughput ✓
  – Low compilation time !
The Remainder of this Talk

- Automata Processor
  - Architectural Overview
  - Current Programming Models
- RAPID Programming Language
  - Language Overview
  - AP Code Generation and Optimizations
- Experimental Evaluation
- Conclusions and Future Directions
RAPID at a Glance

• Provides concise, maintainable, and efficient representations for pattern-identification algorithms
• Conventional, C-style language with domain-specific parallel control structures
• Excels in applications where patterns are best represented as a combination of text and computation
• Compilation strategy balances synthesis time with device utilization
Program Structure

• **Macro**
  – Basic unit of computation
  – Sequential control flow
  – Boolean expressions as statements for terminating threads of computation

• **Network**
  – High-level pattern matching
  – Parallel control flow
  – Parameters to set run-time values
This program structure also exposes optimizations...
Program Structure

```
network ( ... ) {
    ...
}
macro foo (...) { ... }
macro bar (...) { ... }
macro baz (...) { ... }
macro qux (...) {
    ...
}
```

Network
Data in RAPID

• Input data stream as special function
  – Stream of characters
  – \texttt{input()}
    • Calls to \texttt{input()} are synchronized across all active macros
    • All active macros receive the same input character
Counting and Reporting

• **Counter**: Abstract representation of saturating up counters
  – Count and Reset operations
  – Can compare against threshold

• **RAPID programs can report**
  – Triggers creation of report event
  – Captures offset of input stream and current macro
Parallel Control Structures

• Concise specification of multiple, simultaneous comparisons against a single data stream
• Support MISD computational model
• Static and dynamic thread spawning for massive parallelism support
• Explicit support for sliding window computations
# Parallel Control Structures

<table>
<thead>
<tr>
<th>Sequential Structure</th>
<th>Parallel Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>if…else</td>
<td>either…orelse</td>
</tr>
<tr>
<td>foreach</td>
<td>some</td>
</tr>
<tr>
<td>while</td>
<td>whenever</td>
</tr>
</tbody>
</table>
Either/Orelse Statements

either {
  hamming_distance(s,d); //hamming distance
  'y' == input(); //next input is 'y'
  report; //report candidate
} orelse {
  while('y' != input()); //consume until 'y'
}

• Perform parallel exploration of input data
• Static number of parallel operations
Some Statements

network (String[] comparisons) {
    some(String s : comparisons)
        hamming_distance(s,5);
}

• Parallel exploration may depend on candidate patterns
• Iterates over items, dynamically spawn computation

```java
macro hamming_distance (String s, int d) {
    Counter cnt;
    foreach (char c : s)
        if(c != input()) cnt.count();
    if(cnt <= d)
        report;
}
```
Whenever Statements

```java
whenever( ALL_INPUT == input() ) {
    foreach(char c : "rapid")
        c == input();
    report;
}
```

- Body triggered whenever guard becomes true
- ALL_INPUT: any symbol in the input stream
Example RAPID Program

Association Rule Mining
Identify items from a database that frequently occur together
Example RAPID Program

If all symbols in item set match, increment counter

Spawn parallel computation for each item set

Sliding window search calls `frequent` on every input

Trigger `report` if threshold reached

```plaintext
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;
    }
}
```
System Overview

Input

RAPID Program
Annotations

RAPID Compiler

ANML
apcompile

Driver Code
AP Binary

Output
Code Generation

• Recursive transformation of RAPID program
  – Input Stream $\rightarrow$ STEs
  – Counters $\rightarrow$ 1 or more physical counter(s)

• Similar to RegEx $\rightarrow$ NFA transformation
Challenge: Synthesis

• Placement and routing are resource-intensive
• Large AP designs often fail outright
• **Goal:** technique to reduce AP design such that synthesis tools succeed
Tessellation Optimization

• Automata Processor designs are often **repetitive**
• Programmatically **extract** repetition, and compile once
• Load **dynamically** at runtime
Auto-Tuning Optimization

Block

Block

Block
Tessellation Advantages

• Reduces overall compilation time
  – Smaller design requires less time to place and route
  – Shorter debug cycle increases productivity

• Improved device utilization
  – Reduced search space size for place and route
  – Able to find “better” solution
The Remainder of this Talk

• Automata Processor
  – Architectural Overview
  – Current Programming Models
• RAPID Programming Language
  – Language Overview
  – AP Code Generation and Optimizations
• Experimental Evaluation
• Conclusions and Future Directions
Reminder: Goals

• Fast processing ✔
• Concise, maintainable representation
• Efficient compilation
  – High throughput
  – Low compilation time
Research Questions

1. Do RAPID constructs generalize to pattern search problems across multiple problem domains?

2. (Conciseness) Do RAPID programs require fewer lines of code than a functionally equivalent ANML program to represent a given pattern search problem?

3. (Maintainability) Does a RAPID program require fewer modifications than an equivalent ANML program to alter functionality?

4. (Efficiency) Are RAPID programs no less efficient at runtime and during synthesis than hand-optimized ANML programs?
Research Questions

1. Do RAPID constructs generalize to pattern search problems across multiple problem domains?

2. (Conciseness) Do RAPID programs require fewer lines of code than a functionally equivalent ANML program to represent a given pattern search problem?

3. (Maintainability) Does a RAPID program require fewer modifications than an equivalent ANML program to alter functionality?

4. (Efficiency) Are RAPID programs no less efficient at runtime and during synthesis than hand-optimized ANML programs?
## Description of Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
<th>Domain</th>
<th>Baseline Generation Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>Association Rule Mining</td>
<td>ML</td>
<td>Meta Program</td>
</tr>
<tr>
<td>Brill</td>
<td>Brill Part of Speech Tagging</td>
<td>NLP</td>
<td>Meta Program</td>
</tr>
<tr>
<td>Exact</td>
<td>Exact DNA Alignment</td>
<td>Bioinformatics</td>
<td>ANML</td>
</tr>
<tr>
<td>Gappy</td>
<td>DNA Alignment with Gaps</td>
<td>Bioinformatics</td>
<td>ANML</td>
</tr>
<tr>
<td>MOTOMATA</td>
<td>Planted Motif Search</td>
<td>Bioinformatics</td>
<td>ANML</td>
</tr>
</tbody>
</table>
Research Questions

1. Do RAPID constructs generalize to pattern search problems across multiple problem domains?

2. (Conciseness) Do RAPID programs require fewer lines of code than a functionally equivalent ANML program to represent a given pattern search problem?

3. (Maintainability) Does a RAPID program require fewer modifications than an equivalent ANML program to alter functionality?

4. (Efficiency) Are RAPID programs no less efficient at runtime and during synthesis than hand-optimized ANML programs?
RAPID Lines of Code

Percent Reduction

ARM | Brill | Exact | Gappy | MOTOMATA

0% | 20% | 40% | 60% | 80% | 100%
Research Questions

1. Do RAPID constructs generalize to pattern search problems across multiple problem domains?

2. (Conciseness) Do RAPID programs require fewer lines of code than a functionally equivalent ANML program to represent a given pattern search problem?

3. (Maintainability) Does a RAPID program require fewer modifications than an equivalent ANML program to alter functionality?

4. (Efficiency) Are RAPID programs no less efficient at runtime and during synthesis than hand-optimized ANML programs?
RAPID is Maintainable

Task: Convert Hamming distance comparison of length 5 to length 12
Research Questions

1. Do RAPID constructs generalize to pattern search problems across multiple problem domains?

2. (Conciseness) Do RAPID programs require fewer lines of code than a functionally equivalent ANML program to represent a given pattern search problem?

3. (Maintainability) Does a RAPID program require fewer modifications than an equivalent ANML program to alter functionality?

4. (Efficiency) Are RAPID programs no less efficient at runtime and during synthesis than hand-optimized ANML programs?
Parallel searches

Maximize number of parallel active searches by reducing STE usage

Key

Active Searches

Target Pattern

Incoming Data

ATCGA

Small footprint increases throughput

CGGCAT
Generated STEs

- Percent Reduction

- ARM: 60%
- Brill: 0%
- Exact: -20%
- Gappy: 40%
- MOTOMATA: 80%
Compilation Time

- ARM*
- Brill
- Exact*
- Gappy*
- MOTOMATA*

**Time (seconds)**

- Handcrafted
- RAPID

* RAPID Tessellation
Research Questions

1. Do RAPID constructs generalize to pattern search problems across multiple problem domains? **YES**

2. **(Conciseness)** Do RAPID programs require fewer lines of code than a functionally equivalent ANML program to represent a given pattern search problem? **YES**

3. **(Maintainability)** Does a RAPID program require fewer modifications than an equivalent ANML program to alter functionality? **YES**

4. **(Efficiency)** Are RAPID programs no less efficient at runtime and during synthesis than hand-optimized ANML programs? **OFTEN** (**YES**)
The Remainder of this Talk

• Automata Processor
  – Architectural Overview
  – Current Programming Models
• RAPID Programming Language
  – Language Overview
  – AP Code Generation and Optimizations
• Experimental Evaluation
• Conclusions and Future Directions
Architectural Targets

RAPID Program
Multi-Layer Automata Research Framework

Frontend
- High-level representation
- Analyses and Transformations

Middle Layer
- ANML
- FSM optimizations

Backend
- AP
- FPGA
- GPU
- CPU
- Experimental architectures

RAPID

VASim
Debugging Support

- Spurious reports in large data stream
- Can we quickly “sweep” to problematic region and inspect?
- Replay debugging
Open Source Release

• Prototype compiler will be available on GitHub
• BSD-style license
• Available in the coming weeks
Conclusions

• RAPID is a **concise, maintainable, and efficient** high-level language for pattern-search algorithms

• Achieved with domain-specific **parallel control structures**, and **suitable data representations**

• Prototype compiler allows for execution using the Automata Processor, FPGA, CPU
QUESTIONS

This work was supported in part by the Center for Future Architectures Research (C-FAR), one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.