RAPID Programming of Pattern-Recognition Processors

Kevin Angstadt
Qualifying Exam Depth Presentation
MCS Project Presentation
22. April 2016
Publication: ASPLOS 2016

RAPID Programming of Pattern-Recognition Processors

ASPLOS 2016
April 2–6
Atlanta, GA

Acceptance Rate: 23%
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?

Pattern Search Problems

- 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 on social media posts
- Search for Higgs events based off on paths of subatomic particles
Parallel searches

Key

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

\[ \ldots \]

\[ \begin{array}{cccccc}
G & G & C & T & A & T \end{array} \]

Incoming Data

\[ \begin{array}{cccccc}
G & T & C & T & G & G \end{array} \]

ATCGA

CGGCAT
Parallel searches

Key

Active Searches

Incoming Data

Target Pattern

CGGCAT

ATCGA

Active Searches = Target Pattern
Parallel searches

Key

Active Searches

Target Pattern

Incoming Data

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

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

\[ \text{Incoming Data} \]

\[ \text{ATCGA} \quad \times \]

\[ \text{CGGCAT} \quad \times \]
Parallel searches

Key

Active Searches

Target Pattern

TATACTG

Incoming Data

ATCGA

CGGCAT

Active Searches = Target Pattern
Parallel searches

Key
... = Active Searches

Incoming Data

ATCGA

CGGCAT
Parallel searches

Key

Active Searches

Target Pattern

Incoming Data

... C A T A C G G G C T T T T T ...

ATCGA

Target Pattern

CGGCAT
Parallel searches

Key

\[
\text{Active Searches} = \begin{array}{c}
0 \\
\vdots \\
9 \\
\end{array}
\]

Target Pattern

Incoming Data

\begin{align*}
&\text{CGGCAT} \\
&\text{ATCGA} \\
\end{align*}
Parallel searches

Key

= Active Searches

Incoming Data

ACTCATACG

...
Parallel searches

Key
= Active Searches

Incoming Data

Target Pattern

ATCGA

Active Searches

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G

G

CGG

G
Parallel searches

Key

Active Searches

Target Pattern

T G A C C C A T A

Incoming Data

T G A C C C A T A

C G G C T

ATCGA

...
Parallel searches

Key

\[
\vdots \quad \vdots \quad \vdots
\]

Active Searches

Target Pattern

\[
\begin{array}{ccccccccccc}
C & T & G & A & C & C & A & T & C & G & G & C & \ldots
\end{array}
\]

Incoming Data

\[
\begin{array}{ccccccccccc}
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{array}
\]

\[
\begin{array}{ccccccccccc}
A & C & G & G & C & A & \ldots
\end{array}
\]

\[
\begin{array}{ccccccccccc}
A & C & G & G & C & A & \ldots
\end{array}
\]

ATCGA

CGCCAT

Active Searches = Target Pattern
Parallel searches

Key

Active Searches

Target Pattern

Incoming Data

CGGCAT

ATCGA

CGGCAT

…
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
Micron’s Automata Processor

- Altera Stratix IV GX230 FPGA
- JTAG header
- AS header
- SODIMM socket M4_0
- SODIMM socket M4_1
- PCIe Gen2 x8 Connector
- 2GB DDR3 M1
- UDIMM socket M2
- Reset button
- 2x4 Power Connector
- SODIMM socket M3_0
- SODIMM socket M3_1

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
Thinking ahead...
This program structure also exposes optimizations
Program Structure

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

• Input data stream as special function
  – Stream of characters
  – `input()`
    • Calls to `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
Whenever Statements

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

```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;
    }
}
```

- 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
System Overview

Input

RAPID Program
Annotations

Output

Driver Code

AP Binary

ANML

apcompile

RAPID Compiler
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
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

<table>
<thead>
<tr>
<th></th>
<th>Percent Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>80%</td>
</tr>
<tr>
<td>Brill</td>
<td>40%</td>
</tr>
<tr>
<td>Exact</td>
<td>60%</td>
</tr>
<tr>
<td>Gappy</td>
<td>100%</td>
</tr>
<tr>
<td>MOTOMATA</td>
<td>100%</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 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

Small footprint increases throughput

Incoming Data

ATCGA

CGGCAT

Target Pattern

Active Searches
Generated STEs

<table>
<thead>
<tr>
<th>Percent Reduction</th>
<th>ARM</th>
<th>Brill</th>
<th>Exact</th>
<th>Gappy</th>
<th>MOTOMATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>80%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-20%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-40%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Compilation Time

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM*</td>
<td></td>
</tr>
<tr>
<td>Brill</td>
<td></td>
</tr>
<tr>
<td>Exact*</td>
<td></td>
</tr>
<tr>
<td>Gappy*</td>
<td></td>
</tr>
<tr>
<td>MOTOMATA*</td>
<td></td>
</tr>
</tbody>
</table>

- **Handcrafted** - blue bars
- **RAPID** - orange bars

* 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
Debugging Support

• Spurious reports in large data stream
• Can we quickly “sweep” to problematic region and inspect?
• Replay debugging
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 acceleration using the Automata Processor