A Scalable and Efficient in-Memory Interconnect Architecture for Automata Processing

Elaheh Sadredini, Reza Rahimi, Vaibhav Verma, Mircea Stan, Kevin Skadron

University of Virginia
elaheh@virginia.edu
Processor / Memory Performance Gap

Source: David Patterson, UC Berkeley

Sorry, didn’t know that it would be that serious!
Processor / Memory Performance Gap

Source: David Patterson, UC Berkeley

Sorry, didn’t know that it would be that serious!
Scalable and High-Performance Techniques Are Needed for Pattern Processing

- Incoming packet is checked against every single rule of the database
Scalable and High-Performance Techniques Are Needed for Pattern Processing

- Incoming packet is checked against every single rule of the database

**Problem:**
- Increase in the number of rules
- Increase in the network line rate

**Diagram:**
- Rule-set
  - Rule 1
  - Rule 2
  - ...
  - Rule 10000
- Packet
- Network Intrusion Detection System
- Malicious/Non-malicious packet
Pattern Recognition Importance

Network security
<table>
<thead>
<tr>
<th>Pattern Recognition Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network security</td>
</tr>
<tr>
<td>Bioinformatics</td>
</tr>
</tbody>
</table>

Task: 2780.001
Pattern Recognition Importance

Network security  Bioinformatics  Data mining
Pattern Recognition Importance

Network security  Bioinformatics  Data mining  NLP
Pattern Recognition Importance

Network security  Bioinformatics  Data mining  NLP

Patterns are often complex
Pattern Recognition Importance

Network security  Bioinformatics  Data mining  NLP

Patterns are often complex

Thousands of patterns need to be processed in parallel
Pattern Recognition Importance

Network security | Bioinformatics | Data mining | NLP

Patterns are often complex

Thousands of patterns need to be processed in parallel

Regular Expressions \(=\) Finite Automata
Existing Automata Processing Solution

- Custom ASIC
- von Neumann Architectures
- Memory-Centric Architectures
- Reconfigurable SW/HW Engines
Existing Automata Processing Platforms

- Custom ASIC
- von Neumann Architectures
- Memory-Centric Architectures
- GPU-Based
- CPU-Based

Platforms:
- UAP [9]
- HARE [8]
- iNFAnt2 [10]
- DFAGE [11]
- PCRE [14]
- HyperScan [12]
- VASim [13]

Reconfigurable SW/HW Engines
Existing Automata Processing Platforms

**Problem:** von Neumann processors easily become memory bound

- Unpredictable behavior
  - Branch mispredictions
- Irregular access pattern
  - Cache-miss
- Many parallel state transitions
  - Saturate memory bandwidth
Existing Automata Processing Platforms

- **von Neumann Architectures**
  - UAP [9]
  - HARE [8]

- **Memory-Centric Architectures**
  - Micron Automata Processor (AP) [15]
  - Cache Automaton (CA) [16]

- **GPU-Based**
  - iNFAnt2 [10]
  - DFAGE [11]

- **CPU-Based**
  - PCRE [14]
  - HyperScan [12]
  - VASim [13]

- **Reconfigurable SW/HW Engines**
  - REAPR [17]
State Transition

![State Transition Diagram](image-url)
Problems: interconnect inefficiency in the existing memory-centric architectures

- Automata Processor [15]
  - Routing matrix congestion
  - 13% state utilization for applications with complex routing!

- Cache Automaton [16]
  - Full-crossbar is excessive for interconnect
  - On average, only 0.53% of switches are utilized!
Main Contribution:

Designing a low-overhead, yet flexible routing architecture for automata processing and mapping it to a right memory technology.
Full-Crossbar interconnect

An Example Automaton
Full-Crossbar interconnect

BFS Labeling

An Example Automaton
Full-Crossbar interconnect

Connectivity Matrix

<table>
<thead>
<tr>
<th>Source State</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BFS Labeling

An Example Automaton

Task: 2780.001
Solution: Minimizing Full-Crossbar

Connectivity Matrix

BFS Labeling

An Example Automaton
Observation: Union Heatmap of Routing Switches with BFS Labeling

- 17 out of 19 benchmark applications show **diagonal property**
Observation: Union Heatmap of Routing Switches with BFS Labeling

- 17 out of 19 benchmark applications show diagonal property

Only less than 1% of the switch cells are utilized!
Solution: Reduced Crossbar Interconnect

Memory cell as a switch
Solution: Reduced Crossbar Interconnect

An OR operation is needed

Memory cell as a switch

Task: 2780.001
Solution: Reduced Crossbar Interconnect

Our solution requires 7X fewer memory cells!

Memory cell as a switch

An OR operation is needed

Task: 2780.001
Mapping to Memory Technology

- Non-destructive read is necessary to implement OR functionality
- 2T1D cell has lower area overhead than 8T cell

Cache Automaton use 8T SRAM cell

We propose to use 2T1D cell (a type of gain cell)
Summary of Performance Evaluation

• Incorporate both architectural contribution and technology contribution
• eAP_2T1D has **1.7X, 3.3X** and **210X** better throughput per unit area than eAP_8T, CA, and the AP
Thanks for Listening!

Questions?
Please stop by my poster

This work was supported in part by Semiconductor Research Corporation (SRC).