Compact Architecture for High-Throughput Regular Expression Matching on FPGA

Yi-Hua E. Yang, Weirong Jiang and Viktor K. Prasanna
EE, USC

Kyle Wang, NSLAB, Tsinghua University
Outline

Introduction

Background

Basic Architecture on FPGA

Architectural Optimizations

Performance Evaluation

Conclusion and Future Work
Introduction

- **Basic regular expression example (What)**
  - $b^*c\cdot(a|b)^*[ac]\#$

- **RE-NFA & RE-DFA (How)**
  - regular expression matching (REM) architecture based on (Non-)Deterministic Finite Automaton

- **Bottleneck for DPI in IDS (Why)**
  - large number of patterns to scan for and increasing bandwidth of network traffic

- **An improved RE-NFA approach on FPGA**
  - $n$-state $m$-character input REME (REM Engine) with $O(n \times \log_2 m)$ constructing time and $O(n \times m)$ logic units
Challenges to perform REM on hardware

- processing large numbers of patterns in parallel
- obtaining high *concurrent throughput*
  - throughput of the input stream processed concurrently by \( N \) REMEs
  - RE size/complexity, amount of resources, achieved clock frequency

Prior work upon implementation of RE-NFA on hardware

- “Fast Regular Expression Matching Using FPGAs” *(Sidhu et al, FCCM 2001, Cited by 272)*
- optimizations (e.g. utilizing BRAM)……
Parse the regular expression into a tree structure.

The parse tree for regular expression $b^*c(a|b)^*[ac]#$
Basic Architecture on FPGA
From Regular Expression to NFA(2/3)

- Modified McNaughton-Yamada construction

![Diagram](Image)
Basic Architecture on FPGA
From Regular Expression to NFA (3/3)

- Constructing the “modular” NFA using the modified McNaughton-Yamada rules

The RE-NFA for regular expression $b^*c(a|b)^*[ac]#$
Basic Architecture on FPGA
From RE-NFA to HDL (1/2)

- Mapping state directly into logic module
Basic Architecture on FPGA
From RE-NFA to HDL (2/2)

- Mapping NFA directly to HDL

Matching circuit for regular expression $b^*c(a|b)^*[ac]#$
Character classification unit
- 256 bits for specifying all 8-bit characters
  - e.g. character a can be classified by $\overline{000\cdots0100\cdots0000}$
  - max $256 \times n$ bits BRAM usage for $n$-state RE-NFA
  - Less if two states use the same character
- cons.
  - redundancy in BRAM
  - max $2^{256}$ possible character classes (e.g. simple [ac])
  - optimization is necessary
Basic Architecture on FPGA
BRAM-based Character Classification (2/2)

- Dual-4bit selector
  - 2*16 bits
  - only ‘and gate’ delay
  - why not use?
    - logic slices vs. BRAM
    - clock frequency influence

- For character ‘a’
  - 0110 0001
  - if ‘.’?
Architectural Optimizations

- Three available optimizations to improve the basic design above
  - Multi-Character Input Matching
  - Centralized Character Classification
  - Staging and Pipelining

- These techniques are unique to our design and take advantage of the modularity of the proposed architecture.
  - Scalable pattern matching for high speed networks (multi-character)
  - Regular Expression Matching for Reconfigurable Packet Inspection (centralized)
  - Assisting Network Intrusion Detection with Reconfigurable Hardware (pipeline)
Architectural Optimizations
Multi-Character Input Matching (1/2)

- 2-character input (2-Stride Accelerating)

Multiple outputs may be at ‘and gate’ or ‘state register’
Architectural Optimizations
Multi-Character Input Matching (2/2)

- \(O(n \times \log_2 m)\) time Alg. for \(m\)-character input

2-input matching circuit for regular expression \(b^*c(a|b)^*[ac]\#\)
Call function to examine and compare each state’s character class to the character class entries (CCE) in BRAM so far

- **if new:** a new 256-bit entry is added into BRAM
- **if old:** proper connection is made from the BRAM output of the previous CCE to the input of the current state
- \( O(n \times \omega) \) time complexity
  - \( \omega \) is the number of distinct character classes among the n state, in the worst case, \( \omega \) could be linear in \( n \)
Architectural Optimizations
Centralized Character Classification (2/2)

- BRAM storage
- How about the fan-out of the selector?
Against the decline in achievable clock frequency with larger numbers of REMEs

- Architectural Optimizations
  - Staging and Pipelining

- **fixed delay:**
  - $p$ (pipeline) + $s$ (stage)
  - clock cycles
Performance Evaluation
Hardware Complexity of Regular Expression

- Metrics to quantify the complexity of RE
  - area requirement
    - State count: Total number of states needed by REME
  - logic complexity
    - State fan-in: maximum number of states that can immediately transition to any state in the REME
    - State fan-out: maximum number of states to which any state in the REME can immediately transition
  - routing complexity
    - Loop size: total number of transitions within a loop of state transitions
    - Branch-size delta: difference in number of transitions between two state transition paths with the same first and final states
Performance Evaluation

Implemented Regular Expressions (1/3)

- Patterns selected for RE-NFA evaluation
  - avoid patterns too short or too simple
    - can be dealt with pattern-level optimizations
  - count identical patterns in different rules as one
    - should not be used to inflate the REME numbers
  - avoid patterns with long quantified repetitions
    - can be easily dealt with shift-register
  - avoid patterns requiring backreference
    - requires capability beyond regular languages
    - is beyond the scope of this paper
Snort rule categories (partially implemented)

<table>
<thead>
<tr>
<th>Snort rule cat. implemented</th>
<th># of pat.</th>
<th># of states</th>
<th># of states/pat.</th>
</tr>
</thead>
<tbody>
<tr>
<td>backdoor</td>
<td>154</td>
<td>3648</td>
<td>23.7</td>
</tr>
<tr>
<td>chat/ftp</td>
<td>18</td>
<td>163</td>
<td>9.1</td>
</tr>
<tr>
<td>deleted</td>
<td>23</td>
<td>607</td>
<td>26.4</td>
</tr>
<tr>
<td>smtp/pop2/pop3</td>
<td>22</td>
<td>306</td>
<td>13.9</td>
</tr>
<tr>
<td>spyware-put</td>
<td>446</td>
<td>11720</td>
<td>26.3</td>
</tr>
<tr>
<td>web-misc/web-php</td>
<td>51</td>
<td>1082</td>
<td>21.2</td>
</tr>
<tr>
<td>(others)</td>
<td>46</td>
<td>1052</td>
<td>22.9</td>
</tr>
<tr>
<td>760-REME</td>
<td>760</td>
<td>18578</td>
<td>24.4</td>
</tr>
<tr>
<td>523-REME</td>
<td>523</td>
<td>13379</td>
<td>25.6</td>
</tr>
<tr>
<td>267-REME</td>
<td>267</td>
<td>6551</td>
<td>24.5</td>
</tr>
</tbody>
</table>
Performance Evaluation
Implemented Regular Expressions (3/3)

- State fan-in and fan-out values

<table>
<thead>
<tr>
<th>value $v$</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>
</tr>
</thead>
<tbody>
<tr>
<td># of state fan-in = $v$</td>
<td>52</td>
<td>656</td>
<td>23</td>
<td>5</td>
<td>14</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td># of state fan-out = $v$</td>
<td>45</td>
<td>663</td>
<td>31</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

- Although a few states reach 8 or 9, the slowest REME determines the overall clock frequency when implementing in the same clock domain
Performance Evaluation
Implementations Results (1/3)

Throughput scaling of 760 REMEs on Xilinx Virtex 4 LX-100-12
Performance Evaluation
Implementations Results (2/3)

Throughput scaling of 267 REMEs on Xilinx Virtex 4 LX-100-12
Clock rate and LUT usage of 267 REMEs on Xilinx Virtex 4 LX-100-12
Performance Evaluation

Performance Comparison

- Throughput efficiency
  - concurrent throughput of the circuit divided by the number of LUTs the circuit uses per state
  - a metric to evaluate performance in various conditions

<table>
<thead>
<tr>
<th></th>
<th># non-meta char.</th>
<th>Multi-char. m</th>
<th>Tput. (Gbps)</th>
<th># LUT per state</th>
<th>Tput. efficiency (Gbps*state/LUT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>760-REME-2</td>
<td>～20k</td>
<td>2</td>
<td>4.8</td>
<td>1.27</td>
<td>3.8</td>
</tr>
<tr>
<td>523-REME-2</td>
<td>～15k</td>
<td>2</td>
<td>4.88</td>
<td>1.24</td>
<td>3.9</td>
</tr>
<tr>
<td>523-REME-4</td>
<td>～15k</td>
<td>4</td>
<td>7.46</td>
<td>2.2</td>
<td>3.4</td>
</tr>
<tr>
<td>267-REME-4</td>
<td>～8k</td>
<td>4</td>
<td>7.5</td>
<td>2.25</td>
<td>3.3</td>
</tr>
<tr>
<td>Bispo et al. [5]</td>
<td>19580</td>
<td>1</td>
<td>2.9</td>
<td>1.28</td>
<td>2.3</td>
</tr>
<tr>
<td>Clark et al.-1 [6]</td>
<td>17537</td>
<td>1</td>
<td>2.0</td>
<td>1.7</td>
<td>1.9</td>
</tr>
<tr>
<td>Clark et al.-4 [6]</td>
<td>17537</td>
<td>4</td>
<td>7.0</td>
<td>3.1</td>
<td>2.3</td>
</tr>
<tr>
<td>Mitra et al. [13]</td>
<td>N.A.</td>
<td>1</td>
<td>12.8/16</td>
<td>～2.3</td>
<td>～0.35</td>
</tr>
</tbody>
</table>

NOTE: The FPGA platform is not the same, but the throughput efficiency can still illustrate something
Conclusion and Future Work

**Conclusion**
- compact and easy-mapping architecture
- stack character spatially to match multiple input
- utilize block memory (BRAM) available
- 2D staging and pipelining organization

**Future Work**
- group REMEs into stages intelligently (Fang Yu)
  - exploit pattern-level properties such as the common prefix extraction
- separate simple and complex REs into different clock domains
  - supposedly a pipeline of simple REMEs can be clocked higher than complex REMEs
Thanks for your attention!

Let’s Talk.