Fast and Smoothed Packet Classification

Recent advancement of research partnership with V. Prasanna and S.-H. Teng

Jun Li & Yibo Xue
with contributions from our students Yaxuan Qi, Jeffrey Fong, Xiaoqi Ren, et al.
Outline

• Background
• HyperSplit Algorithm
• Architecture for FPGA Implementation
• Evaluation with Smoothed Analysis
• Future Work
Outline

• Background
• HyperSplit Algorithm
• Architecture for FPGA Implementation
• Evaluation with Smoothed Analysis
• Future Work
Packet Classification Problem

- To identify and associate each packet to a specific rule
- May match multiple rules
- Used for:
  - Routing
  - FW, IDS/IPS, & AV
  - LB & TE
  - OpenFlow & SDN
**Theoretical Complexity**

- **Point location problem**

- **Very high complexity**

<table>
<thead>
<tr>
<th></th>
<th>H-Trie</th>
<th>H-Tree</th>
<th>S-Trie</th>
<th>S-Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>d=1</td>
<td>time</td>
<td>$\Theta(W)$</td>
<td>$\Theta(\log n)$</td>
<td>$\Theta(W)$</td>
</tr>
<tr>
<td></td>
<td>space</td>
<td>$\Theta(n \cdot W)$</td>
<td>$\Theta(n \cdot \log n)$</td>
<td>$\Theta(n \cdot W)$</td>
</tr>
<tr>
<td>d&gt;1</td>
<td>time</td>
<td>$\Theta(W^{d-1})$</td>
<td>$\Theta(\log^{d-1} n)$</td>
<td>$\Theta(d \cdot W)$</td>
</tr>
<tr>
<td></td>
<td>space</td>
<td>$\Theta(n \cdot W^{d-1})$</td>
<td>$\Theta(n \cdot \log^{d-1} n)$</td>
<td>$\Theta(n^d \cdot dW)$</td>
</tr>
</tbody>
</table>

Diagram showing point location problem with regions $r_1$, $r_2$, $r_4$, and $r_5$. The point $p(3, 3)$ is located in region $r_4$.
**Performance in Practice**

### Theoretical vs. Practical Complexity for Real-Life Rules

<table>
<thead>
<tr>
<th>Rule Sets</th>
<th># rules</th>
<th># non-over ranges in each field (theoretical)</th>
<th># non-over ranges in sIP (practical)</th>
<th># non-over ranges in dIP (practical)</th>
<th># non-over ranges in sPT (practical)</th>
<th># non-over ranges in dPT (practical)</th>
<th># non-over rectangles (theoretical)</th>
<th># non-over rectangles (practical)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FW1</td>
<td>269</td>
<td>539</td>
<td>100</td>
<td>111</td>
<td>23</td>
<td>77</td>
<td>8.44 x 10^16</td>
<td>1.97 x 10^7</td>
</tr>
<tr>
<td>FW1-100</td>
<td>92</td>
<td>185</td>
<td>19</td>
<td>45</td>
<td>20</td>
<td>48</td>
<td>1.17 x 10^9</td>
<td>8.21 x 10^5</td>
</tr>
<tr>
<td>FW1-1K</td>
<td>791</td>
<td>1583</td>
<td>221</td>
<td>314</td>
<td>23</td>
<td>75</td>
<td>6.28 x 10^12</td>
<td>1.20 x 10^8</td>
</tr>
<tr>
<td>FW1-5K</td>
<td>4653</td>
<td>9307</td>
<td>3429</td>
<td>5251</td>
<td>23</td>
<td>77</td>
<td>7.50 x 10^15</td>
<td>3.19 x 10^10</td>
</tr>
<tr>
<td>FW1-10K</td>
<td>9311</td>
<td>18623</td>
<td>3429</td>
<td>5251</td>
<td>23</td>
<td>77</td>
<td>1.20 x 10^17</td>
<td>1.71 x 10^11</td>
</tr>
<tr>
<td>ACL1</td>
<td>752</td>
<td>1505</td>
<td>221</td>
<td>314</td>
<td>23</td>
<td>75</td>
<td>5.13 x 10^12</td>
<td>9.67 x 10^6</td>
</tr>
<tr>
<td>ACL1-100</td>
<td>98</td>
<td>197</td>
<td>19</td>
<td>45</td>
<td>20</td>
<td>48</td>
<td>1.51 x 10^9</td>
<td>4.03 x 10^5</td>
</tr>
<tr>
<td>ACL1-1K</td>
<td>916</td>
<td>1833</td>
<td>19</td>
<td>45</td>
<td>20</td>
<td>48</td>
<td>1.13 x 10^13</td>
<td>1.32 x 10^7</td>
</tr>
<tr>
<td>ACL1-5K</td>
<td>4415</td>
<td>8831</td>
<td>3429</td>
<td>5251</td>
<td>23</td>
<td>77</td>
<td>6.08 x 10^15</td>
<td>2.42 x 10^8</td>
</tr>
<tr>
<td>ACL1-10K</td>
<td>9603</td>
<td>19207</td>
<td>3429</td>
<td>5251</td>
<td>23</td>
<td>77</td>
<td>1.36 x 10^17</td>
<td>1.90 x 10^9</td>
</tr>
<tr>
<td>IPC1</td>
<td>1550</td>
<td>3101</td>
<td>559</td>
<td>796</td>
<td>49</td>
<td>78</td>
<td>9.25 x 10^13</td>
<td>3.40 x 10^8</td>
</tr>
<tr>
<td>IPC1-100</td>
<td>99</td>
<td>199</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.57 x 10^9</td>
<td>9.49 x 10^6</td>
</tr>
<tr>
<td>IPC1-1K</td>
<td>938</td>
<td>1877</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.24 x 10^13</td>
<td>1.70 x 10^9</td>
</tr>
<tr>
<td>IPC1-5K</td>
<td>4460</td>
<td>8921</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.33 x 10^15</td>
<td>1.03 x 10^10</td>
</tr>
<tr>
<td>IPC1-10K</td>
<td>9037</td>
<td>18075</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.07 x 10^17</td>
<td>6.07 x 10^10</td>
</tr>
</tbody>
</table>

**Note:** sIP, dIP, sPT and dPT are source IP, destination IP, source Port and destination Port; FW, ACL, IPC are firewall policies, access control lists, and IP chain rules.

- Few applications reach the worst case bound
- Real-life rule sets have geometrical redundancy
Progress of Joint Research

**Efficient Algorithms**
- Exploiting real-life rule set redundancy
- HyperSplit Algorithm (Infocom)

**Fast Speed**
- Using SRAM-based solution on FPGA
- 100Gbps Throughput (FPT)

**Smoothed Analysis**
- Introducing Sampling-based Smoothed Analysis
- Practical evaluation (submitted)
Outline

• Background
• HyperSplit Algorithm
• Architecture for FPGA Implementation
• Evaluation with Smoothed Analysis
• Future Work
HyperSplit

- Memory-efficient packet classification algorithm
  - Uses 10% of the memory that other comparable algorithms requires

- Optimized k-d tree data structure

- Uses heuristics to select the most efficient splitting point on a specific field
Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Priority</th>
<th>Field-X</th>
<th>Field-Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>1</td>
<td>00~01</td>
<td>00~00</td>
</tr>
<tr>
<td>R2</td>
<td>2</td>
<td>00~01</td>
<td>00~11</td>
</tr>
<tr>
<td>R3</td>
<td>3</td>
<td>10~10</td>
<td>00~11</td>
</tr>
<tr>
<td>R4</td>
<td>4</td>
<td>11~11</td>
<td>11~11</td>
</tr>
<tr>
<td>R5</td>
<td>5</td>
<td>11~11</td>
<td>00~11</td>
</tr>
</tbody>
</table>

The diagram shows a grid with rules R1 to R5.
Example

X,01

X<=01

L

X>01

R

R1

R2

R3

R4

R5

X<=01

X>01

Lv-1
Example

X,01

Y,00

R

X<=01

X>01

Y<=00

Y>00

R1

R2

R

Lv-2

Lv-1

R1

R2

R3

R4

R5

00 01 10 11
Example

\[
\begin{array}{c}
\text{R1} \\
\text{R2} \\
\text{R3} \\
\text{RR}
\end{array}
\]
Example

```
X,01
  X<=01
   Y,00
    Y<=00
     R1
    Y>00
     R2
   X>01
    Y>00
     R3
   X<=10
    Y<=10
     R5
   X>10
    Y>10
     R4

Lv-1 | Lv-2 | Lv-3
-----|-----|-----
  11  | 10  |  R4
  10  |  01 |  R2
  01  |  00 |  R1
  00  |  10 |  R3
  10  |  11 |  R5
```

X,10
  X>=01
   Y,10
    Y<=10
     R5
   Y>10
     R4
```
Memory Access

- HyperSplit-1 vs. HiCuts-1
  - 50~80% less access

- HyperSplit-8 vs. HiCuts-8
  - 10~30% less access

- HyperSplit-1 vs. HSM
  - 20~50% less access

Memory Usage

Memory Usage: HyperSplit vs. HiCut

- HyperSplit-1 vs. HiCuts-1
  - 1~2 orders less memory
- HyperSplit-8 vs. HiCuts-8
  - 1~2 orders less memory
- HyperSplit-1 vs. HSM
  - 1~2 orders less memory
Preprocessing Time

- HyperSplit-1 vs. HiCuts-1: 1~2 orders less time
- HyperSplit-8 vs. HiCuts-8: About 1 orders less time
- HyperSplit-1 vs. HSM: 1~4 orders less time
Outline

• Background
• HyperSplit Algorithm
• Architecture for FPGA Implementation
• Evaluation with Smoothed Analysis
• Future Work
Existing Solutions

**SRAM Based**
- Advantage:
  - Price
  - (generally) # of Rules
- Disadvantage:
  - Speed

**TCAM Based**
- Advantage:
  - Speed
- Disadvantage:
  - Price
  - Power consumption
  - Chip size
  - Range to Prefix Conversion

Challenges & Goals

Memory Usage
- Needs to be memory efficient that can support large rule sets

High Performance
- Requires high throughput and deterministic performance

On-the-fly update
- To allow rules to be changed and updated without downtime
Mapping Decision Tree into Hardware
Mapping Decision Tree into Hardware

X,01

Y,00  X,10

R1  R2  R3

Y,10

R5  R4
Mapping Decision Tree into Hardware

INPUT PACKET

STAGE 1

STAGE 2

STAGE 3

STAGE 4

MATCHED RULE

INPUT PACKET

STAGE 1

STAGE 2

STAGE 3

STAGE 4

MATCHED RULE
Hardware Implementation

STAGE n
Architecture Optimization (1)

Node-merging: Pipeline Depth Reduction

[@addr0  
d1,v1  
addr1]

[@addr1  
d1,v1  
addr2]  
[@addr1+1  
d1,v1  
addr3]

[@addr2  
child1]  
[@addr2+1  
child2]  
[@addr3  
child1]  
[@addr3+1  
child2]

[@addr0  
d1,d2,d3  
v1,v2,v3  
addr1]

[@addr1  
child1]  
[@addr1+1  
child2]  
[@addr1+2  
child3]  
[@addr1+3  
child4]
Algorithm Evaluation (1)

Node-merging Optimization

- Reduce tree height (pipeline depth) by almost 50%!
- Minimal memory overhead
Architecture Optimization (2)

Leaf-pushing: Controlled BRAM Allocation

- Sizes of BRAM on each stage needs to be predetermined
- Different rule sets will result in different memory usage per stage
- Limits the size of a certain stage by pushing leafs to lower levels of the pipeline
Algorithm Evaluation (2)

Leaf-pushing Optimization

Nodes distribution without leaf pushing

Nodes distribution with leaf pushing
Architecture Optimization (3)

Dual Pipeline

- Take advantage of dual-port BRAM
- Double the throughput without increasing memory usage
Test Setup

- Tested with a publicly available rulesets from Washington University
  - Used the ACL 100, 1K, 5K, 10K rulesets

- Design is implemented on a Xilinx Virtex-6
  - Model: VC6VSX475T
  - Containing 7,640Kb Distributed RAM and 38,304Kb Block RAM
  - Using Xilinx ISE 11.5 tool
### FPGA Performance

<table>
<thead>
<tr>
<th>Rules</th>
<th>Max Clock (MHz)</th>
<th>Max Thrupt (Gbps)</th>
<th>Tree depth</th>
<th>#slices used / available</th>
<th>#RAMs used / available</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl1_100</td>
<td>139.1</td>
<td>142</td>
<td>7</td>
<td>444/37440</td>
<td>10/516</td>
</tr>
<tr>
<td>acl1_1K</td>
<td>134.0</td>
<td>137</td>
<td>11</td>
<td>602/37440</td>
<td>18/516</td>
</tr>
<tr>
<td>acl1_10K</td>
<td>115.4</td>
<td>118</td>
<td>12</td>
<td>747/37440</td>
<td>103/516</td>
</tr>
</tbody>
</table>

### FPGA Comparison

#### Comparison with FPGA-based approaches

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Max #rules</th>
<th>Max Thrupt (Gbps)</th>
<th>Pipeline depth</th>
<th>#slices used</th>
<th>#RAMs used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our approach</td>
<td>50K</td>
<td>142</td>
<td>12</td>
<td>747</td>
<td>103</td>
</tr>
<tr>
<td>HyperCuts on FPGA [Jiang]</td>
<td>10K</td>
<td>128</td>
<td>20</td>
<td>10307</td>
<td>407</td>
</tr>
<tr>
<td>HyperCuts Simplified [Liu]</td>
<td>10K</td>
<td>7.22</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

#### Comparison with multi-core based approaches

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Max Throughput (Gbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our approach</td>
<td>142</td>
</tr>
<tr>
<td>HyperSplit on OCTEON [Qi]</td>
<td>6.4</td>
</tr>
<tr>
<td>RFC on IXP2850 [Liu]</td>
<td>10</td>
</tr>
</tbody>
</table>
Conclusion

- FPGA provides a flexible and excellent solution to the packet classification problem
- HyperSplit algorithm is suitable to hardware implementation with an efficient mapping
  - optimizations used to reduce tree length, constraint the memory usage of each stage, and improve performance
- Consume less resource than other FPGA-based solutions and much faster than multicore based solutions
Outline

• Background
• HyperSplit Algorithm
• Architecture for FPGA Implementation
• Evaluation with Smoothed Analysis
• Future Work
Current Algorithm Evaluation

**Worst-case Evaluation:**
- Use the worst-case performance to evaluate the practical performance

**Drawbacks**
- may be defined in a contrived and extreme circumstance
- may provide a significantly pessimistic evaluation result
Current Algorithm Evaluation

Average-case Evaluation:
- Measures the expected performance of an algorithm over a pre-defined distribution of the inputs.

Drawbacks
- May vary greatly from distribution to distribution.
- Is usually difficult to model the ‘practical’ distribution of inputs in complex applications.
- Tend to result in an overly optimistic evaluation.

For complicated network algorithms, worst-case and average-case analyses cannot reveal practical performance!
New Algorithm Evaluation Method

- Not Practical! ✗
- More Accurate! 😊

Smoothed Analysis
- worst case
- average case
- smoothed case
New Algorithm Evaluation Method

Smoothed Analysis:

\[ \max(E_g(M_A(x + \sigma g))) \]

First Use:

- shadow-vertex simplex algorithm
- worst-case complexity: exponential
  smoothed complexity: polynomial

To facilitate analysis for COMPLICATE algorithms in COMPLEX environment...

- **Aim:**
  Simplified while maintaining Accuracy

- **Method:**
  Sampling-based method (SSA)

- **Formula:**
  $$\max_x (E_g (M_A(x+\sigma g)))$$
SSA Framework

- **STEP1: Inputs Generation**
  - gather $N$ worse cases and constitute set $W$

- **STEP2: Sampling**
  - sample in the neighborhood of each instance $x$ in $W$

- **STEP3: Calculate Results**
  - calculate expectations of result set for each $x$
  - obtain the maximum of all the expectations as SSA result
SA vs. SSA

Divergences

- Smoothing “each input” vs. “local maximums”
- “not sampling” vs. “sampling”

SA and SSA reach ALMOST THE SAME evaluation results!

- With proper parameter selection
  - e.g., choose enough cases into the particular set, with a high enough sampling rate
Case Study

Two algorithms for Packet Classification Problem

- Computational Geometry Algorithm (CG)
- HyperSplit Algorithm (HS)

Evaluate and compare worst-case, average-case, and SSA performances
Case Study: Memory Usage

Memory Usage (KB):

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Worst-case</th>
<th>Average-case</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG</td>
<td>14061</td>
<td>1769.01</td>
</tr>
<tr>
<td>HS</td>
<td>7606</td>
<td>763.24</td>
</tr>
</tbody>
</table>

Worst-case Performance: Bad!
Average-case Performance: Good!  \{ Conflict! \}
Case Study: Memory Usage

SSA Conclusion

- Two algorithms both
  - hardly to be entrapped into a worse case “plateau”

Corresponding to the great practical performance results
Case Study : Tree Depth

Tree Depth:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Worst-case</th>
<th>Average-case</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG</td>
<td>28</td>
<td>24.12</td>
</tr>
<tr>
<td>HS</td>
<td>29</td>
<td>21.59</td>
</tr>
</tbody>
</table>

Worst-case Performance : CG > HS
Average-case Performance: CG < HS

Conflict!
Case Study: Tree Depth

(SSA Conclusion

- Both algorithms
  - only have worse-case “peaks” rather than worse-case “plateaus”.
- CG wins HS in speed narrowly
  - based on the contour line in speed
  - at cost of memory usage
Conclusions

- SSA reveals **PRACTICAL, CLOSE-TO-REAL PERFORMANCE**
- SSA can enhance existing benchmark generator
- “Fast algorithms, smoothed analysis, and hardness results”

X. Ren, Y. Qi, B. Yang, J. Li, and S.-H. Teng, Sampling-based smoothed analysis for network algorithm evaluation, (submitted)
Outline

• Background
• HyperSplit Algorithm
• Architecture for FPGA Implementation
• Evaluation with Smoothed Analysis
• Future Work
Future Work

- Regular Expression Matching algorithmic study
- Novel “explosion free” algorithm
- Many-core and FPGA: architecture and parallel processing
- Sampling-based Smoothed Analysis: further empirical validation and evaluation

Thank you and Questions?