Introduction

Cyber-security community scalability and timeliness requirements in real-life scenarios

Firewall & NIDS

Data leaks detection

Android ICC analysis

Line rate

Real time

Nightly

Offloading and buffering

Hours level delay

6340 hours

[AsiaCCS’17]
Introduction

Research trend in cyber-security community

Three years (’15-’17) stats of top-tier security conference publications

<table>
<thead>
<tr>
<th>Conference</th>
<th>Total # of papers</th>
<th># of papers emphasize on the algorithm efficiency and scalability</th>
<th># of papers focus on implementations on real HPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCS</td>
<td>416</td>
<td>49 (11.8%)</td>
<td>19 (2.2%)</td>
</tr>
<tr>
<td>IEEE S&amp;P</td>
<td>180</td>
<td>12 (6.7%)</td>
<td>10 (5.6%)</td>
</tr>
<tr>
<td>USENIX Security</td>
<td>224</td>
<td>12 (5.4%)</td>
<td>9 (4.0%)</td>
</tr>
<tr>
<td>NDSS</td>
<td>178</td>
<td>11 (6.2%)</td>
<td>4 (2.2%)</td>
</tr>
</tbody>
</table>
Introduction

HPC community

- Require application-specific tuning in addition to generic algorithm optimization to achieve optimal performance

Sparse matrix-vector multiplication (SpMV)

CT image reconstruction

cuSPARSE (3x vs. CPU)
w/ app.-specific opt. (21x vs. CPU)

cuSPARSE (up to 15x vs. CPU)

Mainly for bioinformatics, data mining, climate predicting etc.

Barely for cyber security applications.
Research trend in HPC community

Three years ('15-'17) stats of top-tier HPC conference publications

<table>
<thead>
<tr>
<th>Conference</th>
<th>Total # of papers</th>
<th># of papers focus on generic algorithms impl.&amp;opt. on HPC</th>
<th># of papers focus on applications (security apps) impl.&amp;opt. on HPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td>221</td>
<td>95 (43.0%)</td>
<td>29(0) (13.1%(0%))</td>
</tr>
<tr>
<td>ICS</td>
<td>111</td>
<td>34 (30.6%)</td>
<td>6(1) (5.4%(0.9%))</td>
</tr>
<tr>
<td>HPDC</td>
<td>77</td>
<td>25 (32.5%)</td>
<td>8(1) (10.4%(1.3%))</td>
</tr>
<tr>
<td>IPDPS</td>
<td>339</td>
<td>97 (28.6%)</td>
<td>37(3) (10.9%(0.9%))</td>
</tr>
</tbody>
</table>
Narrow the gap between cybersecurity and HPC communities in three dimensions

1. Identify the challenges

2. Design or refactor the algorithms

3. Provide the implementation and optimization frameworks
Targeted Algorithms

Two algorithms that are broadly adopted in a wide range of cybersecurity applications

- **Automata Processing**
  1. Intrusion Detection/Prevention System (IDS/IPS)
  2. Anomaly detection
  3. Packet filtering/Deep Packet Inspection (DPI)

- **Worklist algorithm**
  1. Taint analysis
  2. Point-to analysis
  3. Control-flow integrity
### Targeted Platforms

<table>
<thead>
<tr>
<th>Encoding/States</th>
<th>GPU</th>
<th>Micron’s AP</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logic</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Encoding/Transitions</th>
<th>Memory Configuration</th>
<th>Programming</th>
<th>Traversal Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Compilation, place&amp;route</td>
<td>strongly dataset dependent (active state set matters)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>weakly dataset dependent (input character/clock cycle + output processing)</td>
</tr>
</tbody>
</table>

**Xiaodong Yu 11/13/2018**

**SC’18 Doctoral Showcase**
Introduction to Automata Processor (AP)

Three Programmable Resources

- State Transition Element (STE) 49152 per chip
- Counter Element 768 per chip
- Boolean Element 2304 per chip
Contribution Summary

Algorithms

Frameworks
1. GPU Acceleration of Regular Expression Matching for Large Datasets [ACM CF’13]
3. Demystifying automata processing: GPUs, FPGAs or Micron's AP? [ACM ICS’17]
4. GPU-assisted Android Program Analysis Framework (ongoing work)
Contribution Summary

Algorithms

Frameworks
1. GPU Acceleration of Regular Expression Matching for Large Datasets [ACM CF’13]
3. Demystifying automata processing: GPUs, FPGAs or Micron's AP? [ACM ICS’17]
4. GPU-assisted Android Program Analysis Framework (ongoing work)
O$^3$FA: A Scalable Finite Automata-based Pattern-Matching Engine for Out-of-Order DPI
Problem Definition

- Input stream: MB or even GB level
- Single packet payload is only 64KB → multiple substreams carried by different packets

- Out-of-Order Packets: packets arrive to the destination in a random order → match escape from the NIDS due to cross-packet matching

input stream: \textit{abcdefg}h  malicious pattern: \textit{cdef}

Remaining transitions

\begin{align*}
\text{no match} & \quad \text{no match} \\
\text{match } \textit{cdef} & \\
\textit{abcdefg} & \quad \textit{efgh} \quad \textit{abcd}
\end{align*}
Buffering and reassembling: Industrial standard solution for the Out-of-Order Deep Packet Inspection (DPI)

**Drawback:**
1. Very resource-intensive
2. Attackers can exhaust the packet buffer capacity
O³FA Design

Input stream: $aabcdefg$   malicious pattern: $bcde$

One match: $cabcdeab$

Two packets: $P1=aabc$   $P2=defg$

Arrived in reversed order: $P2 \rightarrow P1$

Segments: $P2=defg$   $P1=aabc$

  suffix of $bcde$   prefix of $bcde$

When $P2$ arrives, detect and record only $de$
When $P1$ arrives, retrieve $de$ and concatenate it to $P1$
Then we can reconstruct the match $aabcde$
O³FA Design

Advanced challenge: complex sub-patterns in RegEx
   – the most general unbounded repetition: dot-star terms (.*)

Difficulties to handle dot-star terms:

How to represent and detect prefix/suffix of malicious patterns that contain dot-star terms?
   construct FA for prefix/suffix set that contain dot-star terms

How to retrieve the segment through recorded information that matched prefix/suffix with dot-star terms?
   Restore the shortest segment (may not be the same as original segment) that can match the same prefix/suffix
**O³FA Design**

Observation:
Input stream: `aabafgcdef` malicious pattern: `.*b.*cde`

One match: `aabafgcdef`

Two packets: 
- `P1=aabaf`
- `P2=gcdef`

Arrived in reversed order: `P2 → P1`

Segments: 
- `P2=gcdef`
- `P1=aabaf`

When `P2` arrived, suffix `.*cde`
shortest segment `cde`

`P1 → P1’=aabafcde`

Segment `gcde` and `cde` are equivalent
O³FA Design

- O³FA (Out-of-Order Finite Automata)
  - Regular DFA
    \[ \overline{c} \text{ from } 1; \overline{d} \text{ from } 2; \overline{b} \text{ from } 0; \overline{e} \text{ from } 3; . \text{ from } 4 \]
    
    ![DFA Diagram]

- Supporting-FAs
  - prefix NFA
  - non-anchor suffix NFA
  - anchor suffix NFA

- States buffer instead of packets buffer

![Buffer Diagram]

8 Bytes
4 Bytes
Evaluation

Buffer size savings:

- O³FA states buffer vs. traditional packets buffer

State buffer requires 20x-4000x less size than packet buffer
Robotomata: A Framework for Approximate Pattern Matching of Big Data on an Automata Processor
Automata-based APM: Levenshtein Automaton

Target pattern: \textit{object} … which allows a Levenshtein distance of up to 2

\[ \epsilon \] allows automaton to change its state spontaneously, i.e. without consuming an input symbol

\[ * \] allows automaton to be triggered by any input character

Input string: \texttt{object} \hspace{1cm} Output: detected (with one error \( \rightarrow \) D)

Input string: \texttt{objijfct} \hspace{1cm} Output: detected (with two errors \( \rightarrow \) I, S)
Challenges of APM on AP

Programmability

Mathematical format

- ANML: Automata Network Markup Language
  - Requires expertise with both automata theory and AP architecture
  - Example

```
1 int main()
2 {
3   ap_enml_t anml = 0;
4   ap_enml_network_t anmlNet;
5   struct ap_enml_element element;
6   ap_enml_element_ref_t element[5];
7   ap_automaton_t a;
8   ap_element_map_t elementMap;
9 // Create the ANML object
10   anml = AP_CreateAnml();
11 // Create the automata network in the ANML object
12   AP_CreateAutomataNetwork(anml, anmlNet, "anml");
13 // Create the elements that match "a" and start the search
14   element.res_type = RT_STE;
15   element.start = START_OF_DATA;
16   element.match = "a";
17   element.match = 0;
18   AP_AddAnmlElement(anmlNet, &element[0], &element);
19 // Create the elements that match "b" report the match
20   element.res_type = RT_STE;
21   element.start = NO_START;
22   element.match = "b";
23   element.match = 1;
24   AP_AddAnmlElement(anmlNet, &element[1], &element);
25   The rest four STEs are created in the same manner
26   with different symbols attribute
27 // Connect the STEs together to search "abc"
28 // and allow one Levenshtein distance
29 // Match
30   AP_AddAnmlEdge(anmlNet, element[0], element[1], 0);
31   AP_AddAnmlEdge(anmlNet, element[4], element[5], 0);
32 // Insertion
33   AP_AddAnmlEdge(anmlNet, element[0], element[2], 0);
34   AP_AddAnmlEdge(anmlNet, element[2], element[4], 0);
35 // Substitution
36   AP_AddAnmlEdge(anmlNet, element[1], element[3], 0);
37   AP_AddAnmlEdge(anmlNet, element[3], element[5], 0);
38 // Deletion
39   AP_AddAnmlEdge(anmlNet, element[2], element[5], 0);
40 // Compile the ANML object to create the automaton
41   AP_CompileAnml(anml, &a, &elementMap, 0, 0, options, 0);
42   return 0;
```

---

Xiaodong Yu 11/13/2018
SC’18 Doctoral Showcase
Challenges of APM on AP

End-to-End Performance

Example: alua dataset in BLAST

460K automata states vs. 50K STEs

20 * 130s = 2600s
Paradigm-based approach

- Four (4) paradigms: 3 error types (S,I,D) + 1 match type (M)

End-users only need to provide \{error types, pattern length, error number\}

- \textit{ROBOTOMATA} generates the AP automata via a hierarchical construction
Robotomata: Hierarchical Construction

Paradigm set

Error types

Building block

Length, Error#

Block matrix

Inter-block connection

AP Automata
Robotomata: Hierarchical Construction

In back end and transparent to end users!!

{error types, pattern length, error#}

ROBOTOMATA vs. ANML-AN

Xiaodong Yu 11/13/2018
SC’18 Doctoral Showcase
Single BIG Automaton vs. Robotomata Cascadable Macro

macro
Robotomata: Cascadable Macros

In back end and transparent to end users!!!

Enable cascadable macros

AP automata \( \neq \) macros

Classical macros

Cascadable macros

\[ 20 \times 130s = 2600s \]

\[ 130s + 19 \times 3s = 187s \]
**Evaluation**

**Overall performance comparison**

- Bio: BLAST igseqprot dataset, 85k patterns
- IR: NHTSA Complaints dataset, 100k patterns

ROBOTOMATA can achieve up to 461x speedup over its CPU counterpart and 33.1x and 14.8x speedups over the ANML and FU-based Automata Processor (AP) versions, respectively.
1. GPU-based Computed-Tomography Image Reconstruction
   - [IEEE/ACM CCGrid’16] [ACM CF’17] [JSPS’18(journal)]
   - Computational core is SpMV
   - Symmetry-based (app.-aware) compression in addition to CSR format
   - A uniform CUDA kernel handling both SpMV and SpMV_T

2. A Data Layout-aware Auto-tuning Framework for Faster Convolutions on ROCm Platform
   - Work done during the internship at AMD
   - Paper in submission

3. Systematic Study of the Relevance between Cache Configuration and Side-channel Attack (ongoing work)
Thank you!

Questions?