FAST SUPPORT FOR UNSTRUCTURED DATA PROCESSING: THE UNIFIED AUTOMATA PROCESSOR

Yuanwei Fang, Tung T. Hoang, Michela Becchi, Andrew A. Chien
Unstructured Data Applications

- Voice Assistant
- Web Search
- Intrusion Detection
- XML Mining/Parsing
- Motif Searching
- Compress/Decompress
- Text Data Mining
- Signal Triggering

- Unstructured Data: No pre-defined data model, text-intensive.
- Growing applications of “Big Data”; Dataset scaling in size, and sophistication
- Increasingly “real time”
Finite Automata: a Powerful Tool for Pattern Matching

NFA

Search Pattern: a+bc, bcd+, cde

DFA

copyright 2015, Yuanwei Fang, Andrew A. Chien
Why Automata Processing has Poor Performance on CPU/GPU?

- **Worst case for modern processors**
  - Sequential memory indirections
  - Unpredictable accesses / branches
  - Short data items
  - Memory hierarchy
Importance of General-purpose FA Processing

FA Model Innovation
- DFA, NFA (classical)
- A-DFA, delay-DFA [SIGCOMM’06]
- Hybrid-FA [CoNEXT’07]
- XFA [SIGCOMM’08]
- Counting-FA [CoNEXT’08]
- SFA [INFOCOM’11]
- Dual-FA [ToC’13]
- JFA [JSAC’14]
- ......

To date, no such general-purpose architecture exists!
Goals for a Unified Automata Processor (UAP)

• General-purpose:
  • support all finite-automata models

• High Performance:
  • fast enough to match the data movement speed in the memory system
  • low enough power to fit anywhere in the system

• Ease of Use:
  • same programming model as CPU
  • same way as people use regex library
General-purpose and Flexibility

- Detailed Analysis of Dozens of Models
  - identify key data representations
  - Identify common mechanism requirements

- Transition variety is the key
- Transition types define different FA models
Support Transition Diversity

Approach: Transition Decomposition

Example:

\[
\text{cnt++} \rightarrow \text{cnt++} \quad + \quad a \rightarrow \text{cnt++}
\]
UAP Memory Layout

- Using EffCLiP algorithm\(^1\) to pack transitions and actions

Basic UAP Engine
Vector Parallelism and Multi-step Execution

Vector Execution

Lane0 (DFA)  Lane1 (NFA)  Lane2 (JFA)  Lane3 (c-DFA)

Multi-step(symbol) Execution

input: 'a'
input: 'a'
input: 'a'
input: 'a'

input: 'b'
teleport: epsilon
input: 'a'
action: JUMP
input: 'b'

done
input: 'b'
done
input: 'b'
action: cnt+
done

+
because of their ability to provide high bandwidth memory efficiency on chip local memory. Each lane accesses only a single memory that provides a UAP lane with its own bank of the local interface also enabling lanes to be associated with their own banks. Each UAP's implementation connects 64 parallel lanes to a 3.3 Gbps memory that provokes UAP lane. (Figure 5.7)

Each UAP lane can traverse state activation per cycle: the state explosion, consisting of exact matches, and a counter stack. As shown in Figure 5.7, each UAP lane can be associated with a virtual state and a self transition. This is done by tuning a parameter called actionAddr. The parallel vector interface optimizes for a popular state with many self transitions. Finally, the default transitions used by A Version 2 of a DFA and D Version 3 of a NFA is a run of 300 patterns. This tool allows generation, we used 4 sets of character sets, and patterns from 0 to 255. Among these, the only exception is a NFA that has a theoretical peak of 1,000 regular expressions and a counter stack. These workloads are synthetically produced from trace data taken from a proprietary dataset generated from trace data taken from the Project Gutenberg. As of July 4th, 2013, the Spyware and RLE pattern dataset from Sno has a 462 fraction producing traces that trigger different FA traversal patterns. This is done by tuning a parameter called statementAddr.

In all cases, we selected the workloads so as to st...
Use Regex Library for FA Processing

High level applications

Regular Expressions

"^MICRO[0-9]{4}.*UAP"

Regular Expression Compiler
Regex Runtime Library

Software Compiler
Link
Computer
Use UAP for FA Processing

High level applications

Regular Expressions

"^MICRO[0-9]{4}.*UAP"

Software Compiler

Link

UAP

Computer

UAP Compiler

UAP stub

UAP operations

Copyright 2015, Yuanwei Fang, Andrew A. Chien
Code to do FA Processing

#include <regex.h>

... Your Program’s Code ...

regex_t regex;
int ret;
regcomp(&regex, "^MICRO[0-9]{4}.*UAP", 0);
ret = regexec(&regex, "abcdefghijklmn", 0, NULL, 0);
if (!ret) {
    puts("Match");
}
else if (ret == REG_NOMATCH) {
    puts("No match");
}

... Your Program’s Code ...
Exploiting UAP in Software System

```c
#include <regex.h>
... Your Program’s Code ...
regex_t regex;
int ret;
regcomp(&regex, "^MICRO[0-9]{4}.*UAP", 0);
ret = regexec(&regex, "abcdefghijklmn", 0, NULL, 0);
if (!ret) {
    puts("Match");
} else if (ret == REG_NOMATCH) {
    puts("No match");
}
... Your Program’s Code ...
```
Exploiting UAP in Software System

```c
#include <regex.h>
...
Your Program's Code ...
regex_t regex;
int ret;
regcomp(&regex, "^MICRO[0-9]{4}.*UAP", 0);
ret = regexec(&regex, "abcdefghijklmn", 0, NULL, 0);
if (!ret) {
...
load Compiled FA descriptions ...
// Initialize UAP, Clear acceptance flags
write_state_regs(initial states, memory offsets)
vec64 = 0;
// Looping consuming 256 symbols per iteration
while (all_zeros(vec64)) {
    ... load stream data into vector registers ...
    traverse(0, 256, vec64);
    complete_traverse();
}
...
Evaluation

• Workload Datasets
  • Network Intrusion Detection (DFA, A-DFA, JFA, A-JFA, NFA)
  • Compression/Decompression (DFA, RLE-DFA)
  • Signal Triggering (c-DFA, c-NFA)
  • Database Query (flat-JSON-NFA)

• Simulation
  • Cycle-accurate Logic, Local Memory, Cache, DRAM
  • Detailed 32nm TSMC design, UAP runs @ 1.2GHz
  • Energy-accurate modeling of each element
Vector Parallelism and Multi-step Execution significantly reduce instruction counts for ALL FA MODELS
Line Rate (single stream performance)

UAP@1.2GHz (DFA) ~ 9.1Gbps
227x/70x best published CPU/GPU result on realistic dataset!
Power of General Purpose FA Computation

General-purpose FA processing enables UAP to outperform IBM RegX by 5x and CPU/GPU by 100x.
Area and On-chip Power Breakdown

64-lane UAP 2.2 mm², 507 mW
Half size of a single Cortex A-9 core logic
Half power of a single Cortex A-9 core
Related Work

- **Software Implementation**
  - CPU: [Becchi@ANCS09, Mytkowicz@ASPLOS14, Zhao@ASPLOS15, Cameron@PACT14], best throughput 3Gbps (small datasets)
  - GPU: [Yu@CF13, Zu@PPoPP12], best throughput 13Gbps (small datasets)

- **Customized Hardware**
  - Single model
    - DFA: ASIC [Lunteren@MICRO12, Brodie@ISCA06, Tan@ISCA05], TCAM [Liu@USENIX Security10], best line rate 9.2 Gbps
    - NFA: MicronAP [Dlugosch@TPDS14], line rate 1 Gbps
  - Multiple models
    - FPGA [Yang@ToC12], line rate 1~4 Gbps

- **UAP combines general purpose FA computing with top performance**
  - DFA line rate 9.1Gbps
  - Significant throughput DFA (450Gbps), NFA (250 Gbps)
Summary

- UAP is a general-purpose FA processing architecture
- UAP increases performance by >100x vs CPUs and GPUs; Matches customized ASIC and FPGA implementations
- UAP is small and low-power, so it can be easily integrated across future computing systems
- UAP is easy to use. It makes software processing of unstructured data using FA’s efficient and pervasively usable

Funding Acknowledgement
- DARPA HR0011-13-2-0014
- NSF CCF-1421765 and CNS-1319748
- Keysight Research
- Synopsys Academic program