The Algorithm and Framework Designs and Optimizations for Scalable Automata Processing on HPC Platforms

Xiaodong Yu
Dept. of CS, Virginia Tech
Blacksburg, VA, USA
xdyu@vt.edu

Michela Becchi*
Dept. of ECE, NC State University
Raleigh, NC, USA
mbecchi@ncsu.edu

Danfeng (Daphne) Yao†
Dept. of CS, Virginia Tech
Blacksburg, VA, USA
danfeng@vt.edu

ABSTRACT

Automata processing could perform as the core of many applications in the areas such as network security, text mining, and bioinformatics. Achieving high-speed and scalable automata processing is exceptionally challenging. For one thing, the classic DFA representation is memory-bandwidth efficient but suffer from the state explosion problem in the presence of large datasets with complex patterns. And for another, the automata processing is inherently difficult to be parallelized due to the strong dependencies and unstructured memory-access pattern.

In this thesis, we provide a comprehensive scheme for improving the automata processing efficiency. At the algorithm level, we propose JFA that uses state variables to avoid state explosion. We also propose O^{3}FA to handle out-of-order packets in NIDS. At the implementation level, we propose a comprehensive GPU-based automata processing framework and a code-generation framework for Automata Processors. Moreover, we provide a toolchain to conduct the apple-to-apple automata processing accelerators comparison.

1 INTRODUCTION

Regular expression matching is an essential task in several application domains (bibliographical search, networking, and bioinformatics) and has received particular consideration in the context of deep packet inspection (DPI), a fundamental network security operation, most notably as the core of network intrusion detection systems (NIDS). Regular expression matching is traditionally performed using deterministic and non-deterministic finite automata - DFAs and NFAs, respectively. DFAs can achieve optimal matching speed at the cost of high memory space requirements; NFAs are space efficient but exhibit high memory bandwidth requirements, potentially leading to inferior processing speeds. As a result, the exploration space of the automata processing is characterized by a trade-off between the size of the automaton and the worst-case bound on the amount of per character processing.

From an implementation perspective, automata processing engine can be classified into two categories: memory-based and logic-based. Within the former, the FA is stored in the memory; within the latter, it is stored in logic gates. Memory-based solutions can be deployed on various platforms including general purpose multi-core processors, network processors, FPGAs, ASIC- and TCAM-based systems; logic-based designs typically target FPGAs. Particular attention has been paid to providing efficient logic- and memory-based representations of the underlying automata (namely, DFAs, NFAs and equivalent abstractions). Because of their massive parallelism and computational power, in recent years GPUs have been considered a viable platform for the automata processing.

More recently, Micron has announced their Automata Processor (AP) [4], a DRAM-based processing-in-memory (PIM) architecture dedicated to NFA simulation. AP can perform parallel automata processing within memory arrays on SDRAM dies by leveraging memory cells to store trigger symbols and simulate NFA state transitions. Current AP programming model requires the programmers to manipulate computational elements (STEs) and their inter-connections using a low-level language named Automata Network Markup Language (ANML).

2 PROBLEM STATEMENT

The state explosion issue can take place during DFA generation when the considered patterns contain bounded and unbounded repetitions of wildcards or large character sets. Several alternative FA representations have been proposed to address this problem [5]. However, these proposals all suffer from one or more of the following problems: some can avoid state explosion only on datasets of limited size and complexity; some have prohibitive worst-case memory bandwidth requirements or processing time; and some can only guarantee functional equivalence for restricted classes of regular expressions and require the user to manually filter out unsupported patterns.

In real-world scenarios, a network data stream can span multiple packets. Those packets can arrive at network security devices out of order due to multiple routes, packet retransmission, or NIDS evasion. To match the signatures of malicious traffic across packet boundaries, NIDS typically performs pattern matching after flow reassembly or packet reordering. However, this makes the detection vulnerable to denial-of-service (DoS) attacks, whereby attackers exhaust the buffer capacity by sending long sequences of intentionally misordered packets. While researchers have proposed solutions for exact-match patterns [3], regular-expression matching on out-of-order packets is still an open problem. Specifically, a key challenge is the matching of complex sub-patterns (such as repetitions of wildcards matched at the boundary between packets).

Automata processing is known as the "embarrassing sequential" computational pattern due to its tight dependencies between every two traversal steps and the unstructured memory-access pattern. A handful of proposals have exploited the parallelism intrinsic (i.e., coarse-grained packet-level parallelism and fine-grained data structure parallelism) to propose efficient automata processing designs for GPUs [2]. However, most GPU solutions aim at achieving good

*Master's advisor
†PhD advisor

performance on small datasets, which are far less complex and problematic than those used in real-world applications.

Micron's Automata Processor (AP) has been showcased successfully on a variety of applications [1]. However, the AP ecosystem suffers from two significant problems. First, the current APIs of AP (i.e., ANML) require manual manipulations of all computational elements. Second, multiple rounds of time-consuming compilation are needed for large datasets. Both problems hinder programmer productivity and end-to-end performance.

Despite the abundance of work on high-speed automata processing, there is still a lack of clarity as to how existing software and hardware solutions are related to and compare with each other. This unclarity is because: (i) existing solutions are based on different automata models; (ii) some automata processing architectures are designed to optimize the peak performance of a single input stream, while others offer better support for stream-level concurrency; (iii) automata processing must operate in two steps: the preprocessing step and the traversal step. Although the effect of preprocessing overhead on performance can be significant for applications with more dynamic pattern sets or traversal times in the order of a few seconds, it is largely neglected by previous studies.

3 RESEARCH CONTRIBUTION HIGHLIGHTS

3.1 JFA

We propose JFA, a finite automaton extended with state variables to avoid state explosion. JFA satisfies all of the following requirements: (i) small memory footprint; (ii) limited worst-case processing time (or memory bandwidth requirements); (iii) coverage of large datasets and arbitrary regular expressions; (iv) functional equivalence to NFA and DFA representations; and (v) automated construction (since manual intervention is error-prone). In particular, (i) and (ii) are both fundamental to line-speed processing, which requires the use of fast and compact memories.

3.2 O^{3}FA

We propose O^{3}FA, a new finite automata-based DPI engine to perform regular-expression matching on out-of-order packets in real time, i.e., without requiring flow reassembly. Two features of O^{3}FA distinguish it from previous solutions: (i) it doesn't require packet buffering; (ii) it can handle regular expressions with complicated sub-patterns. O^{3}FA consists of regular DFAs coupled with a set of supporting FAs either in NFA or DFA form. The supporting FAs are used to detect and record - using only no more than five states - segments of packets that can potentially be part of a match across packet boundaries. While processing packets out-of-order, those segments can be dynamically retrieved from the recorded states and can be then used to resolve matches across packet boundaries.

3.3 GPU-based Automata Processing

In this work, we present the GPU-based solution of different automata representations on datasets of practical size and complexity. We show three schemes to avoid some of the limitations of a baseline NFA-based design. We discuss the impact of state-of-the-art memory compression schemes on DFA solutions. We also propose a software managed cache on GPU and evaluate its efficiency.

3.4 ROBOTOMATA

We propose a paradigm-based approach to hierarchically generate automata on AP and use this approach to create Robotomata, a framework for APM on AP. By taking in the following inputs - the types of APM paradigms, the desired pattern length, and the allowed number of errors - our framework can generate the optimized APM-automata codes on AP, to improve programmer productivity. The generated codes can also maximize the reuse of pre-compiled macros and significantly reduce the time for reconfiguration. Our evaluations show that the generated codes can achieve up to 30.5x speedup with respect to configuration while maintaining the computational performance. Compared to the counterparts on CPU, our codes achieve up to 393x overall speedup, even when including the reconfiguration costs.

3.5 Comparison Toolchain

We propose a toolchain to allow an apples-to-apples comparison of NFA acceleration engines on three platforms: GPUs, FPGAs and Micron's AP. Since NFAs do not suffer from state explosion, it allows us to perform an evaluation on large-scale datasets without posing any restrictions on the kind of patterns supported. Specifically, we compare GPU- and FPGA- based NFA engines with Micron's AP. AP extends NFAs' functionality with counters and boolean elements. To ensure functional equivalence and the same degree of programmability across the considered platforms, we extend existing FPGA- and GPU-based designs to support these features, and we propose a compiler toolchain to automatically deploy extended NFAs (in ANML form) onto these three platforms. Our evaluation covers resource utilization, throughput and preprocessing costs for real-world NFAs used in networking and bioinformatics applications, as well as synthetic datasets covering regular expressions datasets with various characteristics.

4 SIDE AND ONGOING WORK

We propose cuART, a complete compression and parallelization solution for the ART-based Computed Tomography image reconstruction on GPUs. We identify the computational patterns in the ART as the SpMV and SpMV_T with application-specific characteristics. Instead of using the standard libraries, we optimize the SpMV implementation by wisely leveraging these specific features.

I'm currently working on two projects: (i) accelerating the android program analyses on GPUs; (ii) a systematic study regarding the effect of cache configuration on the timing side-channel attack vulnerability.

REFERENCES

[1] C. Bo, V. Dang, E. Sadredini, and K. Skadron. 2018. Searching for Potential gRNA Off-Target Sites for CRISPR/Cas9 Using Automata Processing Across Different Platforms. In *2018 IEEE Int'l Symp. on High Performance Computer Arch. (HPCA).*

[2] N. Cascarano, P. Rolando, F. Risso, and R. Sisto. 2010. iNFAnt: NFA Pattern Matching on GPGPU Devices. *SIGCOMM Comput. Commun. Rev.* (2010).

[3] X. Chen, K. Ge, Z. Chen, and J. Li. 2011. AC-Suffix-Tree: Buffer Free String Matching on Out-of-Sequence Packets. In *2011 ACM/IEEE Seventh Symp. on Arch. for Networking and Commun. Systems.*

[4] P. Dlugosch, D. Brown, P. Glendenning, M. Leventhal, and H. Noyes. 2014. An Efficient and Scalable Semiconductor Architecture for Parallel Automata Processing. *IEEE Trans. on Parallel and Distributed Systems* 25, 12 (2014).

[5] R. Smith, C. Estan, S. Jha, and S. Kong. 2008. Deflating the Big Bang: Fast and Scalable Deep Packet Inspection with Extended Finite Automata. In *the ACM SIGCOMM 2008 Conf. on Data Commun. (SIGCOMM '08).*