Cache Automaton: Repurposing Caches for Automata Processing

Arun Subramaniyan, Jingcheng Wang, Ezhil R. M. Balasubramanian
David Blaauw, Dennis Sylvester, Reetuparna Das, University of Michigan-Ann Arbor

1 Introduction

Finite State Automata (FSA) are powerful computational models for extracting patterns from large streams (TBs/PBs) of unstructured data such as system logs, social media posts, emails, and news articles. FSA are also widely used in network security [6], bioinformatics [4] to enable efficient pattern matching.

2 Motivation

Compute-centric architectures like CPUs and GPGPUs perform poorly on automata processing due to irregular memory accesses and can process only few state transitions every cycle due to memory bandwidth limitations. On the other hand, memory-centric architectures such as the DRAM-based Micron Automata Processor (AP) [2] can process up to 48K state transitions in a single cycle due to massive bit-level parallelism and reduced data movement/instruction processing overheads.

Micron Automata Processor: The Micron AP repurposes DRAM columns to store FSM states and the row address to stream input symbols. It implements homogeneous non-deterministic finite state automata (NFA), where *each state has incoming transitions only on one input symbol*. Each state has a label, which is the one-hot encoding of the symbols it is required to match against. Each input symbol is processed in two phases: (1) `state-match`, where the states whose label matches the input symbol are determined and (2) `state-transition`, where each of the matched states activates their corresponding next states.

3 Challenges and Novel Contributions

We explore SRAM-based last-level caches (LLCs) as a substrate for automata processing that are faster and integrated on processor dies.

Cache capacity: One immediate concern is whether caches can store large automata. Interestingly, we observe that AP sacrifices a huge fraction of die area to accommodate the routing matrix and other non-memory components required for automata processing and only has a packing density comparable to caches.

Repurposing caches for automata processing: While the memory technology benefits of moving to SRAM are apparent, repurposing the 40-60% passive LLC die area for massively parallel automata computation comes with several challenges. Processing an input symbol every LLC access ($\sim$20-30 cycles @ 4GHz), would lead to an operating frequency comparable to DRAM-based AP ($\sim$200 MHz), negating the memory technology benefits. Increasing operating frequency further can be made possible only by architecting an (1) in-situ computation model which is cognizant of internal geometry of LLC slices, and (2) accelerating `state-match` (array read) and `state-transition` (switch+wire propagation delay) phases of symbol processing.

Accelerating state-match: This is challenging because industrial LLC subarrays typically have 4-8 bitlines sharing a sense-amp. This means that only 1 out of 4-8 states stored can match every cycle leading to gross under-utilization and loss of parallelism. To solve this, we leverage sense-amp cycling techniques that exploit spatial locality of state-matches.

Accelerating state-transition: Accelerating state-transition at low-area cost requires the design of a scalable interconnect that efficiently encodes and supports multiple state-transitions on the same cycle, often to the same destination state. We observe that a 8T SRAM memory array can be re-purposed to become a compact state-transition crossbar for automata supporting large fan-in for states.

Scaling to large automata: Supporting all-to-all connectivity between states requires prohibitively large and slow switches. Large real-world automata are typically composed of several connected components, grouped into densely connected partitions with only few (8-16) interconnections between them. This motivated us to explore a hierarchical switch topology with local switches providing rich intra-partition connectivity and global switches providing sparse inter-partition connectivity. To this end, we also design a compiler that automates the mapping of states into SRAM arrays.

4 Evaluation

We propose and evaluate two architectures and mapping policies, one optimized for performance and the other optimized for space, across a set of 20 diverse benchmarks from ANMLZoo [5] and Regex [1] suites. We also demonstrate acceleration of parsing activities in browser front-end as a case study where FSA computations are a bottleneck taking up to 40% of the loading time of web pages [3]. The performance optimized and space optimized designs provide a speedup of $15\times$ and $9\times$ over Micron's AP respectively.

References

[1] Becchi et al. A workload for evaluating deep packet inspection architectures. In *IISWC*, 2008.
[2] Dlugosch et al. An efficient and scalable semiconductor architecture for parallel automata processing. *IEEE TPDS*, 2014.
[3] Jones et al. Parallelizing the web browser. In *First USENIX Workshop on Hot Topics in Parallelism*, 2009.
[4] Roy et al. Discovering motifs in biological sequences using the micron automata processor. *IEEE/ACM TCBB*, 2016.
[5] Wadden et al. Anmlzoo: a benchmark suite for exploring bottlenecks in automata processing engines and architectures. In *IISWC*, 2016.
[6] Yu et al. Fast and memory-efficient regular expression matching for deep packet inspection. In *ACM/IEEE ANCS*, 2006.

978-1-5090-6764-0/17 $31.00 © 2017 IEEE
DOI 10.1109/PACT.2017.51