Architectural Support for Efficient Large-Scale Automata Processing

Hongyuan Liu*, Mohamed Ibrahim*, Onur Kayiran†, Sreepathi Pai‡, and Adwait Jog*
*College of William & Mary †Advanced Micro Devices, Inc. ‡University of Rochester
Email: {hliu08,maibrahim}@email.wm.edu, onur.kayiran@amd.com, sree@cs.rochester.edu, ajog@wm.edu

I. The Problem and Motivation

With the near-end of Moore's law and Dennard scaling, domain-specific architectures are getting extremely popular in both industry and academia. We focus on accelerating the processing of Non-deterministic Finite Automata (NFA), a widely used representation of the popular compute model based on Finite State Machine (FSM). This model is widely used for a large set of workloads in the domains such as pattern matching, data analytics, malware detection, bio-informatics, XML parsing, browsers, and search engines. In general, NFA-based applications are very hard to accelerate via traditional von Neumann architectures (e.g., CPUs and GPUs) because of the inherent lack of parallelism and irregular memory accesses. To this end, we propose mechanisms to efficiently architect an Automata Processor (AP) [1], which is a spatial, in-memory, and non-von Neumann that takes advantage of the internal and natural parallelism of memory (DRAM).

One of the fundamental challenges with spatial architectures such as AP is that program size is a first order concern – there are a fixed number of STEs available and a spatial program (e.g., all states across all NFAs) must fit *completely* to begin execution. Otherwise, execution may be impossible, or in the best case multiple rounds of reconfiguration and input re-execution may be required that can incur significant performance penalties.

We found in our MICRO'18[1] paper that not all states of an NFA are enabled during execution leading to its severe under-utilization. Specifically, a large fraction of states unnecessarily take space in the AP chip (leading to its underutilization) but are not part of any state transitions. We refer to such never-enabled states as *cold* states and the remaining (enabled) states as *hot* states. Figure 1 quantitatively shows our observation across 26 diverse applications [1] sorted in the increasing order of their percentage of hot states (across all NFAs in an application). We find that on average 59% of states are cold and it can be up to 99% in applications such as `CAV4k`. Due to this underutilization, many applications with a large number of states (either from the same or different NFAs) are impossible to execute or need to be split into multiple batches.

[1]Hongyuan Liu, Mohamed Ibrahim, Onur Kayiran, Sreepathi Pai, and Adwait Jog, Architectural Support for Efficient Large-Scale Automata Processing, in the Proceedings of 51st International Conference on Micro-Architecture (MICRO), Fukuoka, Japan, Oct, 2018

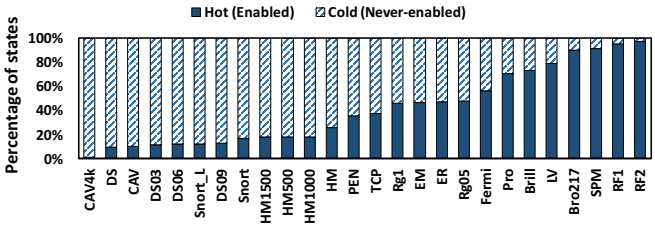

Fig. 1: A large portion of NFA states are cold (never-enabled) but are still configured on the AP leading to its underutilization.

II. Our Approach

To address the performance overhead incurred due to input re-executions and re-configurations across multiple batches, we propose to develop efficient NFA partitioning to accelerate large-scale NFA applications. The key idea behind the partitioning mechanism is to allow only the *hot* and active NFA states to reside in the AP thereby increasing its throughput and reducing the number of batches that need to be processed by AP. To

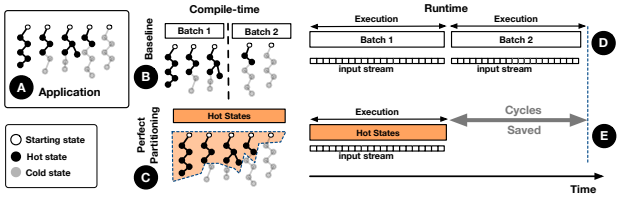

Fig. 2: An illustrative figure showing that by not configuring cold states on AP, all the hot states can fit onto an AP at the same time, reducing the number of re-executions over the input and hence saving time.

illustrate the benefits of configuring the AP with only hot states, Figure 2 shows two scenarios: a) the baseline AP execution, and b) the AP that only executes hot states. The execution in both cases considers the same application (Ⓐ). In the baseline scenario, if the number of total states is more than the AP capacity, the execution will need to be done in batches as discussed before. In this example, the compiler partitions the application into two batches, where each batch can individually fit in the AP (Ⓑ). Hence, the same input stream is executed twice in a sequential manner (Ⓓ). However, with the oracular knowledge of cold states, the compiler can generate a *perfect partition* of the application with only the hot states (Ⓒ). If this perfect partition fits in the AP, it can execute on it by

consuming the same input stream only once (❸), resulting in significant savings in the execution cycles.

A. Topological-level NFA Partitioning

Any realistic implementation that eliminates cold states from NFAs (i.e., partitions NFAs into cold and hot states, and only configures hot states on to the AP) has to deal with at least three challenges. First, although it is not possible to predict cold states with 100% accuracy in general, we need to develop low-overhead techniques to improve the accuracy of prediction as much as possible. Second, in the case of a mis-prediction, some transitions may require states that were not configured on the AP. To this end, we need a mechanism working as a safety net to handle a transition from a state on the AP to a state that is not on the AP. Third, to minimize the cost of such mis-predictions, transitions should be unidirectional to avoid re-executions of inputs on the AP.

Our proposed partitioning scheme systematically addresses these challenges. First, we observe that in the majority of applications, the hot states have low normalized depth (i.e., they are closer to the starting state of the NFAs). Therefore, a state is hot or cold is highly correlated with its normalized depth. Overall, "shallow" states are more likely to be hot while "deep" states are more likely to be cold. Based on this observation, we developed a low-overhead profiling-based scheme to identify the topological layer that acts as a *partition layer* for each NFA in the application. This results in two sets of states: predicted hot and predicted cold. Second, our proposed scheme handles transitions out of the AP by adding *intermediate reporting states* that piggyback on existing AP reporting hardware. Finally, to ensure unidirectional transitions, we partition the NFA at a specific topological order. Since the matching always proceeds from a lower to a higher topological order, edges that cross partitions go only in one direction.

B. Handling Mis-predictions

To efficiently handle the intermediate reports generated from the execution of the predicted hot set, we propose to: a) enable the states that intermediate reporting state directs to, and b) continue the matching process from the cycle (i.e., the input position) where the intermediate report was generated at. In order to support the aforementioned steps, we propose an augmented AP which supports two modes: BaseAP mode, and SparseAP (SpAP) mode. The BaseAP mode execution is similar to the baseline AP execution, however, AP in this mode is configured with only the predicted hot set. Once the execution of BaseAP mode finishes, the generated intermediate reports are handled in the SpAP mode. In the SpAP mode, the AP is configured with the predicted cold set. The AP in this mode not only consumes input symbols but is also driven by the intermediate reports. In this context, we develop two major operations for the SpAP mode: *enable* and *jump*. The enable operation allows each intermediate report to enable the appropriate state in the predicted cold set. The jump operation skips over the input symbols that are not necessary for handling the intermediate reports. Since no back-edge exists

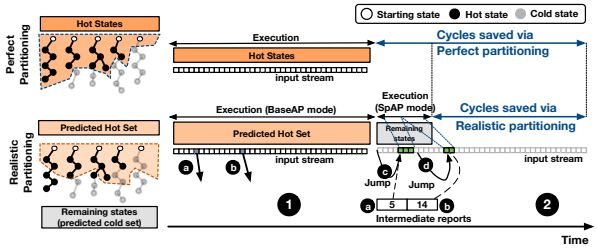

Fig. 3: Illustration of performance benefits under realistic partitioning: because of the *jump* operation, only a portion of input symbols are executed in the SpAP mode execution.

from predicted cold states to predicted hot states, no back and forth switching between BaseAP and SpAP modes is required.

Illustrative Example. Figure 2 earlier discussed the performance benefits of perfect partitioning. Under realistic partitioning, inaccurate predictions of cold states require intermediate report handling. Figure 3 shows an illustrative example demonstrating the benefits of executing AP in BaseAP and SpAP modes. The execution starts in the BaseAP mode (❶) that is configured with the predicted hot set. During its execution, two intermediate reports are generated at input position 5 and input position 14, respectively and are stored (ⓐ, ⓑ). Once all the input symbols are consumed, the SpAP mode begins (❷), which is driven by both the input stream and the intermediate reports. If no state is enabled, SpAP mode jumps to the input position where the next intermediate report was generated. In this example, initially, it jumps to the input position 5 of the first intermediate report directly (ⓒ). During the execution, when there is no enabled state (at input position 8), the SpAP jumps to input position (14) of the next intermediate report (ⓓ). Therefore, under SpAP, only a portion of the input symbols are executed (green shaded part in ❷).

III. Summary of Evaluation Results

In summary, we make use of the inherent properties of NFAs to avoid using compute resources for states that are never used during execution by a low-cost software/hardware-coordinated approach. We evaluate our mechanisms with all (26) applications in the ANMLZoo benchmark suite [1] and the Regex benchmark suite. We build our mechanisms on top of the open-source virtual automata simulator – VASim. Across the evaluated applications, our newly proposed execution model for AP in conjunction with an accurate/low-overhead profiling mechanism obtains 2.1× geometric mean speedup (up to 47×) and 32% in performance/area over the baseline AP execution. In the original paper, we also show additional results related to sensitivity to AP resources and provide a theoretical performance model.

References

[1] J. Wadden, V. Dang, N. Brunelle, T. Tracy II, D. Guo, E. Sadredini, K. Wang, C. Bo, G. Robins, M. Stan, and K. Skadron, "ANMLZoo: A Benchmark Suite for Exploring Bottlenecks in Automata Processing Engines and Architectures," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*, 2016.