A Scalable and Efficient In-Memory Interconnect Architecture for Automata Processing

Elaheh Sadredini, Reza Rahimi, Vaibhav Verma, Mircea Stan, *Fellow, IEEE*, and Kevin Skadron, *Fellow, IEEE*

Abstract—Accelerating finite automata processing benefits regular-expression workloads and a wide range of other applications that do not map obviously to regular expressions, including pattern mining, bioinfomatics, and machine learning. Existing in-memory automata processing accelerators suffer from inefficient routing architectures. They are either incapable of efficiently place-and-route a highly connected automaton or require an excessive amount of hardware resources. In this paper, we propose a compact, low-overhead, and yet flexible in-memory interconnect architecture that efficiently implements routing for next-state activation, and can be applied to the existing in-memory automata processing architectures. We use SRAM 8T subarrays to evaluate our interconnect. Compared to the Cache Automaton routing design, our interconnect reduces the number of switches 7×, therefore, reduces area overhead for the interconnect. It also has faster row cycle time because of shorter wires and consumes less power.

Index Terms—Interconnect, processing in memory, automata processing

1 Introduction

Data collection and the need for real-time analytics are rising rapidly. Pattern matching is an important operation used in many big-data applications such as network security, data mining, and genomics. These patterns are often complex, needing to support a variety of inexact matches. One leading methodology for inexact pattern matching is therefore to use regular expressions or equivalent finite automata to identify these complex patterns [1], [2], [3].

Researchers are increasingly exploiting in-memory hardware accelerators as performance growth in conventional processors is slowing down. The Micron Automata Processor (AP) [4] and Cache Automata (CA) [5] propose in-memory accelerators for automata processing. They both allow native execution of non-deterministic finite automata (NFAs), an efficient computational model for regular expressions, by providing a reconfigurable substrate to lay out the rules in hardware. This allows a large number of patterns to be executed in parallel, up to the hardware capacity.

For NFA processing in memory-centric models, each input requires two processing phases: *state matching*, where the input symbol is decoded and the states whose rules match the input symbol are detected through reading a row of memory, and *state transition*, where successor states are activated by propagating signals through the interconnect.

The interconnect design of existing automata processing accelerators, such as the AP and CA, are either incapable of efficient place-and-route of a highly-connected automaton or over-provision hardware resources for interconnect, at the expense of resources for state-matching [6]. However, real-world benchmarks are quite large in terms of number of states, too big to fit in a single hardware unit, and thus need multiple rounds of reconfiguration and re-processing of the data. This incurs significant performance penalties and makes state-matching resources a scarce resource.

To address the interconnect inefficiencies in the existing in-memory automata processing architectures, this paper presents a *reduced-crossbar (RCB)* design, a low-overhead and yet flexible interconnect architecture that efficiently implements state-transition. RCB design is inspired by intrinsic properties of real-world automata connectivity patterns. RCB requires at least 7× fewer switches compared to the full-crossbar (FCB) design used in CA. This in turn reduces the wire length, which results in shorter latency and lower power consumption. In addition, the area efficiency of RCB provides an opportunity to design a denser state matching, which can accommodate more states and thus results in fewer rounds of re-configuration and re-processing of data.

Across 19 applications from ANMLZoo [7] and Regex [8] benchmark suites, 17 of them can entirely map to RCB design and no FCB is required. To provide a general solution for every possible connectivity topology, we design a reconfigurable memory array for state-matching, in which blocks can be re-purposed as an FCB to provide full connectivity when needed (at the expense of some state capacity). In addition, to support an automaton with a larger number of states, we design global switches that provide inter-block connectivity between RCBs and FCBs blocks.

To efficiently allow many-to-many transitions in an automaton in memory, the underlying memory technology should be able to support logical OR functionality within memory rows in a subarray. This requires memory cells (a) to provide non-destructive read, and (b) to drive output to a "stable" state (logical OR in this case) when multiple bitcells drive a common bitline. 8T SRAM cells is an example of feasible memory technologies to implement automata interconnect in memory.

The proposed interconnect optimization is general and can be applied to any memory-based interconnect, such as variations of gain cells or non-volatile memory (NVM), where memory cells have non-destructive read property and can implement OR functionality for routing.

This paper makes the following *contributions*:

- We propose a compact, flexible, and low-overhead interconnect architecture that efficiently implements the state transition stage in automata processing.
- We design a complete in-memory automata processing unit (APU) with RCB and compare it with Cache Automaton and the Automata Processor.
- We present a new place-and-route algorithm that is 1-2 orders of magnitude faster than the AP compiler, and provide an open-source cycle-accurate automata simulator to perform software optimization on the automata and map them to the proposed architecture.

2 Background and Motivation

Several ASIC and FPGA implementations have been proposed [9], [10], [11], [12] to accelerate pattern matching and automata processing. While ASICs provide high line-rates in principle, they are limited by the number of parallel patterns and shape of the automata and thus, do not provide general and scalable solutions. Moreover, FPGA solutions fail to map complex-to-route automata to the FPGA routing resources due to their logical interconnect complexity [11].

Recently, reconfigurable in-memory automata processing hardware accelerators have been proposed based on NFAs, both to exploit parallel state-matching and transitions, and the benefit of

- *E. Sadredini and K. Skadron are with the Department of Computer Science, University of Virginia, Charlottesville, VA 22903. E-mail: {elaheh, skadron}@virginia.edu.*
- *R. Rahimi, V. Verma, and M. Stan are with the Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA 22903. E-mail: {rahimi, vv8dn, mircea}@virginia.edu.*

Manuscript received 4 Mar. 2019; revised 1 Apr. 2019; accepted 3 Apr. 2019. Date of publication 8 Apr. 2019; date of current version 27 June 2019.
(Corresponding author: Elaheh Sadredini.)
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/LCA.2019.2909870

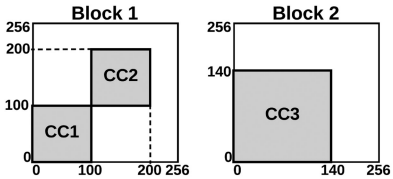

Fig. 1. Full-crossbar utilization.

the NFA's more compact representation. The AP [4] re-purposes DRAM arrays for the state-matching and proposes a hierarchical FPGA-style programmable interconnect design. Our study on a diverse set of 19 automata benchmarks reveals that congestion in the AP routing matrix cripples efficient state utilization, especially for difficult-to-route automata.

CA [5] proposes an in-SRAM automata processing accelerator by re-purposing L3 cache for the state-matching and using 8T SRAM cells for the interconnect. To address routing congestion in the AP, CA proposes to use a full-crossbar (FCB) topology to support full connectivity in an automaton, meaning there can be an edge between every two states at the cost of $O(N^2)$ (N is the number of states). This incurs a high area overhead, which means more than 50 percent of the hardware resources in CA are spent for interconnect! Moreover, FCBs are extremely inefficient and excessive for the needs of real applications, as explained bellow.

Inefficiency of FCB for Automata Applications. NFAs for real-world automata applications are typically composed of many independent patterns, which manifest as separate *connected components (CCs)* with no transitions between them. Each connected component has usually a few hundred states. All the connected components can thus be executed in parallel, independently of each other. Therefore, a crossbar switch can be utilized by packing connected components as densely as possible using a greedy approach [5].

Assume the FCB switch block's size is 256×256. In a greedy approach, CCs are first sorted based on the number of states in each component and then, are assigned to the interconnect resources. Assume there are three CCs of size 100, 100, and 140. Fig. 1 shows mapping of CCs to the FCB switch blocks. Switches in gray areas are configured for the corresponding CC. White areas (70 percent of total area), are unused switches. Moreover, within each connected component, transitions are sparse, meaning very few switches in the gray areas are used.

Fig. 2 shows the total switch utilization within connected components. On average, fewer than 0.48 percent (maximum 1.2 percent in Levenshtein automata [7]) of switch cells are utilized in the FCB interconnect solution. This confirms that the FCB model is extremely inefficient for automata processing applications and forces larger area overhead, power consumption, and delay in the state-transitions phase.

3 Interconnect Architecture

To motivate our efficient and compact interconnect, we visualize the connectivity matrix for the automaton in each benchmark with an image. We first label each node in an automata with a unique index using breadth first search (BFS) numeric labeling, since BFS assigns adjacent indices to the connected nodes. To draw the image, we

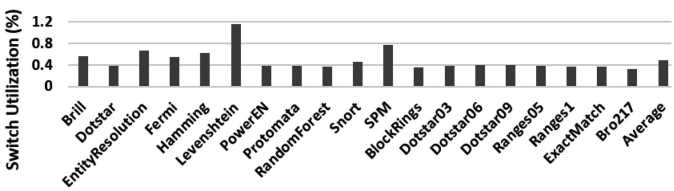

Fig. 2. FCB switch utilization (%) in different applications.

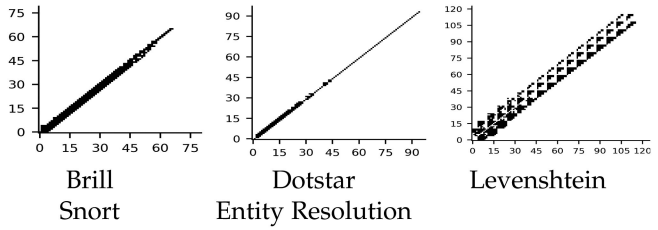

Fig. 3. Union heatmap of routing with BFS labeling.

model an edge (transition) between two nodes (with indices i and j) in an automaton with a black pixel at coordinate (i,j).

In Fig. 3, each graph shows the union over all connectivity images for connected components in one benchmark. We chose union to make sure that we have considered every possible transition, even for rare connection patterns. 17 out of 19 benchmarks from ANMLZoo [7] and Regex [8] suites represent a nice diagonal connectivity property.

This diagonal connectivity pattern motivates a more compact and efficient interconnect, and comes from two properties: first, the power of numeric BFS labeling, which tries to label a child node closely to its parent(s); second, CCs are mostly tree-shape graphs with short cycles and the nodes have a small out-degree. Motivated by these observations, we propose a *reduced* crossbar interconnect (RCB), which has switch patterns similar to what we observed in the union images.

Feasibility Support for RCB Design. In order to actually save area in RCB design, we need to compact the memory array, still with the same amount of input and output signals similar to an FCB, but with smaller area overhead, since it needs fewer switches. This might complicate the layout process because wiring congestion may happen while compacting the array. Automated layout generation tools sometimes are not clever enough to provide the best compacting scheme even for regular patterns like RCB. Therefore, we propose a simple scheme to compact a FCB array to a smaller RCB array.

Simply flipping the diagonal-shape interconnect to a horizontal or vertical block forces the wire congestion in one dimension and it does not utilize the other available dimension to contribute in signal routing. However, squeezing the diagonal-shape to a square shape would significantly compact the subarray and at the same time, spread the burden of signal routing in both dimensions.

Fig. 4 shows an example for an FCB subarray of size (9,9) with diagonal width of 3. In each square, the first index shows the row index and second one shows the column index. For example, a switch in the location (4,3) shows that the input signal comes from a STE labeled 4 (in BFS) and it is connected to a STE labeled 3. The

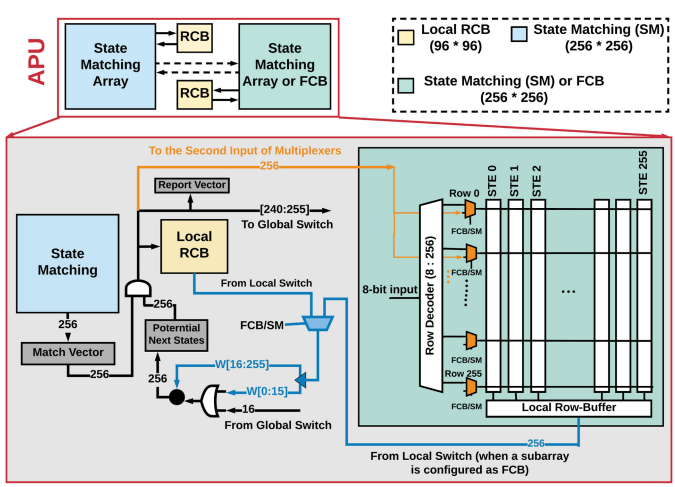

Fig. 4. FCB to RCB compression.

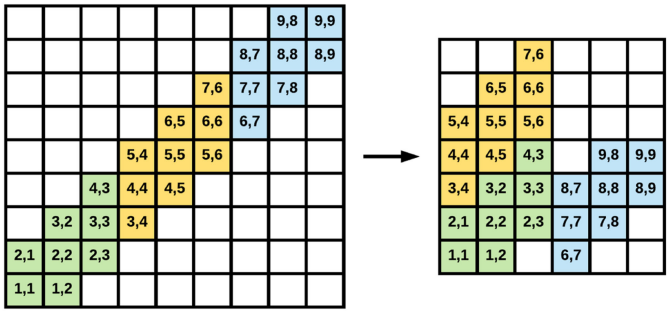

Fig. 5. Design of automata processing unit (APU).

left block shows the initial mapping of diagonal memory cells, while all the white regions are the wasted areas (or switches). The right block shows how moving nearby memory cells close to the lower left side can reduce 9×9 array to 7×6. Our placement guarantees that in each row and column, the number of wires has increased to a maximum of 3 times compared to the original FCB placement. For example, in row 4 of RCB, there are two inputs (wordlines), 2 and 9, and in column 3, there are two output signals (bitline signal), 3 and 6.

Our calculation shows that an FCB of size 256×256 and diagonal width 21 can be reduced to a RCB of size 96×96, which results in approximately $7\times$ switch saving. From our experiments, we found that the diagonal width of 21 is a safe margin to accommodate all the transitions (except in Entity Resolution and Snort). It should be noted that in the routing subarrays, there is no need for decoding the input because the "active state vectors" (or an array of registers) are directly connected to the wordlines. Therefore, RCB does not incur any extra area overhead for extra decoders. Moreover, RCB has smaller bit-lines due to area compression, which potentially leads to a shorter memory access cycle and lower power consumption.

Mapping to SRAM Subarrays. To model transitions from multiple STEs to one STE, the output should be connected to multiple inputs. This is equal to logical OR of active inputs. To have a fair comparison with FCB, we adopt 8T SRAM cells as the reconfigurable switches to evaluate our interconnect architecture. The cell consists of a 6T SRAM cell and two additional transistors, which connect the cell to a bitline. This allows a 6T cell to drive the bitlines only when the cell holds '1' and the input signal is '1'. This implies that 8T cells can support OR functionality.

4 Automata Processing Unit

The architecture of a working automata processing unit (APU) is shown in Fig. 5. An APU consists of two state matching (SM) arrays (SRAM subarray of size 256×256) and a dedicated RCB interconnect (SRAM subarray of size 96×96) for each SM unit as the local interconnect. The local RCBs are connected through global switches (which are FCB units) in order to support a larger automaton (or CC). The global FCB switch allows 8 APUs to communicate, allowing up to 32 transitions among APUs.

In each APU, one of the SM arrays can be configured as FCB interconnect to support uncommon cases in which a connected component does not fit into an RCB interconnect. When a SM subarray needs to be configured as an FCB instead of regular state match operation, the FCB/SM signal of that array is set to one. This signal selects the word lines of the target subarray to be driven by the match vector register bits instead of the decoder output. This mode halves state capacity of the contributed tile but provides the ability to accept connected component without any limitation on interconnect shape. To support this functionality, an array of 2:1 multiplexers needs to be added for one of the subarrays in APU (FCB/SM mutliplexers). This has less than 2.5 percent area overhead based on industry 28 nm 2:1 mux area numbers.

5 Results

We evaluate interconnect architecture on memory arrays with 8T cells using 28 nm technology and nominal voltage 0.9V. To calculate area, power, and row cycle time of memory arrays, we use a foundry standard memory compiler. We develop an in-house cycle-accurate automata simulator[1] to perform software optimization on the automata, map them to the proposed architecture, and extract per-cycle statistics.

5.1 Interconnect Efficiency

This section compares architectural benefits RCB over FCB used in CA. RCB interconnect is a memory block of 96×96, whereas FCB is a memory block of 256×256, meaning that RCB consumes $7\times$ fewer switches (or memory cells) than FCB, which reduces area overhead for the interconnect.

To study the applicability of RCB design in real-world and synthetic automata applications, we calculate the number of required RCB and FCB blocks for each application. The compiler iterates over the connected components (CCs) and checks if they can fit in a RCB switch block. If not, a FCB switch is needed to accommodate connectivity.

In Table 1, we compare the number of required routing blocks of our interconnect approach, which is a hybrid of RCB and FCB, versus the baseline FCB, which is proposed in CA and assumes full connectivity for all the connected components. As shown, all the connected components in 17 out of 19 applications can entirely map to RCB blocks, and no FCB block is needed. This means that when using RCB blocks, the total number of switches (memory cells) required for these applcaitons is $7.1\times$ less than when using FCB blocks. This again confirms that the FCB is extremely excessive for automata applications.

In Entity Resolution, there are many long distance loops, and none of the CCs can fit in the RCB switch block (Fig. 3). In Snort, our interconnect accommodates most of the CCs in RCB blocks (only 19 FCB and 256 RCB), whereas the baseline uses 270 FCBs.

Levenshtein is a difficult-to-route automata. The AP compiler can fit this benchmark in an AP chip with 48K states. However, only 13 percent of the state matching resources in a block are utilized in Levenshtein automata, and the remaining 87 percent cannot be used because there are not enough routing resources left. This implies that 87 percent of the STEs of an AP chip are wasted to deal with the routing congestion. However, our interconnect model allows 100 percent of the STEs in Levenshtein (or other applications with even more complex interconnect) to be utilized.

AutomataZoo [13] benchmark suite is a more generalized version of ANMLZoo, allowing to generate automata with different configurations. Generated automata have similar connectivity patterns as ANMLZoo automata, and thus, can benefit from RCB interconnect design.

5.2 Overall Area Overhead

This section evaluates the area benefit of our RCB design in a complete automata processing architecture, call Reduced Interconnect Architecture (RIA). RIA replicates the APUs (see Fig. 5), with having 8T memory arrays for both state matching and local/global interconnects. This is because state matching arrays can be configured as FCB. Fig. 6 shows the area overhead for state matching, interconnect, and total overhead of RIA, CA and the AP, assuming supporting 32K states. Compared to CA, RIA reduces area overhead of interconnect $3.8\times$. Overall area overhead (both state match

1. https://github.com/anonymousUser0/cal

TABLE 1
Comparison of Our Interconnect Approach (Hybrid RCB and FCB) with CA Interconnect (FCB Only)

		Brill	Dotstar	Entity Resolution	Fermi	Ham-ming	Leven-shtein	Power EN	Proto-mata	Random Forest	Snort	SPM	Block Rings	Dotstar 03	Dotstar 06	Dotstar 09	Ranges 05	Ranges 1	Exact Match	Bro 217
CA [5]	FCB	168	378	0	160	47	12	160	165	139	270	419	192	49	50	50	50	50	50	10
Our Idea	RCB	0	0	500	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0
	FCB	168	378	0	160	47	12	160	165	139	252	419	192	49	50	50	50	50	50	10

Our idea requires up to 7.1X fewer switches (memory cells) than CA.

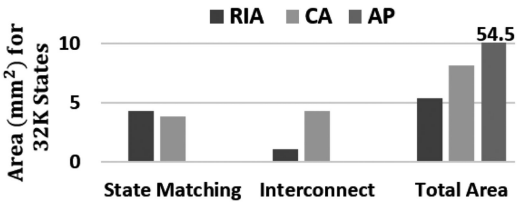

Fig. 6. Area overhead of RIA (Reduced Interconnect Automata), CA, and the AP normalized for 32K states.

and routing) of RIA is 1.4× and 10× less compared to CA and the AP respectively, all in 28nm technology.

6 Conclusion

Motivated by connectivity patterns in the real-world automata benchmarks, we propose a high-speed, dense, and low-power reconfigurable in-memory *reduced* crossbar interconnect (RCB) for state transitions in automata. RCB compacts the switch patterns in a full crossbar interconnect and provides a 7× reduction in the number of switches. This, in turn, reduces power consumption and delay due to shorter wires. Overall, eAP presents 3.8× less area overhead compared to Cache Automaton.

Acknowledgments

This work is funded, in part, by the NSF (CCF-1629450) and CRISP, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by MARCO and DARPA.

References

[1] T. Tracy, Y. Fu, I. Roy, E. Jonas, and P. Glendenning, "Towards machine learning on the automata processor," in *Proc. Int. Conf. High Perform. Comput.*, Springer, 2016.
[2] E. Sadredini, R. Rahimi, K. Wang, and K. Skadron, "Frequent subtree mining on the automata processor: Challenges and opportunities," in *Proc. Int. Conf. Supercomputing*, 2017, Art. no. 4.
[3] E. Sadredini, et al., "A scalable solution for rule-based part-of-speech tagging on novel hardware accelerators," in *Proc. Int. Conf. Knowl. Discovery Data Mining*, 2018, pp. 665–674.
[4] P. Dlugosch, D. Brown, P. Glendenning, M. Leventhal, and H. Noyes, "An efficient and scalable semiconductor architecture for parallel automata processing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3088–3098, Dec. 2014.
[5] A. Subramaniyan, J. Wang, E. R. M. Balasubramanian, D. Blaauw, D. Sylvester, and R. Das, "Cache automaton," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, 2017, pp. 259–272.
[6] J. Wadden, et al., "Automata-to-routing: An open-source toolchain for design-space exploration of spatial automata processing architectures," in *Proc. IEEE 25th Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2017, pp. 180–187.
[7] J. Wadden, et al., "ANMLZoo: A benchmark suite for exploring bottlenecks in automata processing engines and architectures," in *Proc. IEEE Int. Symp. Workload Characterization*, 2016, pp. 1–12.
[8] M. Becchi, M. Franklin, and P. Crowley, "A workload for evaluating deep packet inspection architectures," in *Proc. IEEE Int. Symp. Workload Characterization*, 2008, pp. 79–89.
[9] V. Gogte, A. Kolli, M. J. Cafarella, L. D'Antoni, and T. F. Wenisch, "Hare: Hardware accelerator for regular expressions," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, 2016, pp. 1–12.
[10] Y. Fang, T. T. Hoang, M. Becchi, and A. A. Chien, "Fast support for unstructured data processing: The unified automata processor," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, 2015, pp. 533–545.
[11] R. Karakchi, L. O. Richards, and J. D. Bakos, "A dynamically reconfigurable automata processor overlay," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs*, 2017, pp. 1–8.
[12] Y. Zhuo, et al., "CSE: Parallel finite state machines with convergence set enumeration," in *Proc. Int. Symp. Microarchitecture*, 2018, pp. 29–41.
[13] J. Wadden, et al., "AutomataZoo: A modern automata processing benchmark suite," in *Proc. Int. Symp. Workload Characterization*, 2018, pp. 13–24.