Automata-to-Routing: An Open-Source Toolchain for Design-Space Exploration of Spatial Automata Processing Architectures

Jack Wadden, Samira Khan, and Kevin Skadron
University of Virginia
Charlottesville, Virginia 22904
{wadden,samirakhan,skadron}@virginia.edu

***Abstract*—Newly-available spatial architectures to accelerate finite-automata processing have spurred research and development on novel automata-based applications. However, spatial automata processing *architecture* research is lacking, because of a lack of automata optimization and place-and-route tools. To solve this issue, we propose a new, open-source toolchain–Automata-to-Routing (ATR)–that enables design-space exploration of spatial automata architectures. ATR leverages existing open-source tools for both automata processing and FPGA architecture research. To demonstrate the usefulness of this new toolchain, we use it to analyze design choices of spatial automata processing architectures. We first show that ATR is capable of modeling the logic tiles of a commercially-available spatial automata processing architecture. We then use ATR to compare and contrast two different routing architecture methodologies–hierarchical and 2D-mesh–over a set of diverse automata benchmarks. We show that shallower 2D-mesh-style routing fabrics can route complex automata with equal channel width, while using up to $4.2x$ fewer logic tile resources.**

I. Introduction

As we enter the era of "big data," quickly filtering and extracting information from petabyte datasets and tens to hundreds of gigabytes per second of network traffic has become increasingly important. One prominent methodology for identifying patterns in large files is to use regular expressions and equivalent finite state machines (finite automata), which are extremely powerful tools for pattern recognition and pattern mining. Finite automata recognize patterns by traversing a set of states according to transition rules guided by symbols from an input text. If the automaton enters into a "final" state, it has recognized a pattern in the text, and reports the location in the input stream and the pattern that was recognized.

Most prior research in automata processing is motivated by the acceleration of regular-expression ruleset matching for deep packet inspection [1, 2] and virus detection [3]. New research has identified novel, promising domains for automata-based application acceleration, including machine learning [4], bio-informatics [5], pattern-mining [6], entity resolution [7], natural-langauge processing [8], high-energy particle physics [9], and agent-based simulation and pseudo-random number generation [10].

Finite automata simulation typically has very low arithmetic computational intensity, and in general offers little spatial and temporal locality in its memory accesses. Thus, today's server-class CPUs struggle with large-scale automata processing (with possibly many hundreds of random memory accesses required per cycle). Graphics processing units (GPUs) offer a large amount of parallel resources, which can aid in hiding the latency of DRAM accesses. However, the random and varying parallelism of automata accesses can be hard to map to a GPU's SIMD architecture [11]. These off-the-shelf von Neumann architectures struggle to meet today's big-data and streaming line-rate processing requirements [11].

While von Neumann CPU and GPU architectures struggle with the difficult parallel memory accesses and/or local memory capacity challenges inherent in automata processing workloads, spatial architectures (reconfigurable networks of processing elements such as FPGAs) excel. Spatial architectures can place-and-route automata states and connections just like a sequential circuit, and are a natural target for the acceleration of finite automata processing.

Prior spatial automata acceleration has investigated using FPGA fabrics to place-and-route automata states [12, 13, 14]. New commercially available hardware such as Micron's Automata Processor [15] uses a finite-automata-specific reconfigurable fabric to improve state density over techniques on more general-purpose fabrics. In order to achieve the full potential of automata processing, it is necessary to further research automata-specific spatial architectures that can identify tradeoffs among different design decisions. However, of the new automata processing architectures published since Micron published details of the Automata Processor [16, 17, 18], all are either von Neumann-based techniques or implemented in existing FPGA fabrics.

The reason for this lack of spatial architecture research is three-fold:

1) A benchmark suite of diverse, real-world automata processing benchmarks was not available for fair apples-to-apples evaluations of automata processing architectures.

2) Suitable open-source tools for the optimization of automata (tantamount to good performance on both von Neumann and spatial architectures) did not exist. Thus, important standard optimizations must be reproduced, making research in automata processing high effort.

3) Furthermore, no open-source tools for spatial automata architecture research (including high-quality place-and-route algorithms and flexible, parametric automata-processing fabric description languages) existed for fair evaluation of spatial automata-processing architecture design choices.

To solve the above problems, this paper presents a new open-source toolchain–Automata-to-Routing (ATR)–that can place-and-route any AP application on a parameterizable spatial automata processing architecture. ATR builds upon three open-source tools: a newly available automata processing benchmark suite, ANMLZoo [11], an open-source framework for automata optimization and transformation, VASim [19], and a well-known existing FPGA architecture research framework, VPR [20]. Integration of these tools was not straightforward, and required additional capabilities to enforce spatial architecture design rules on abstract automata, and emit automata in a VPR-readable format. The ATR toolchain enables research on novel spatial automata processing architectures, and can be used to evaluate design choices in existing, commercially available processors.

To show the usefulness of the ATR toolchain we use it to explore bottlenecks in a commercially available automata processor. We first create a baseline model of Micron's Automata Processor (AP) [15] to see if ATR is capable of faithfully modeling the AP's logic-tile architecture. We compare placement performance of this baseline AP model architecture to the real AP, using the ANMLZoo benchmark suite. In many cases, the ATR toolchain can closely model the AP's logic-tile architecture, matching placement statistics to within an average of 7.9% for 9/14 benchmarks. This result indicates that we are able to accurately model the AP's tile architecture for most applications, and motivates future high-level design-space exploration of automata-processing logic-tile architectures using the toolchain.

ATR cannot closely model five benchmarks. We demonstrate that this is due to differences between the AP and ATR's routing matrix designs. Micron's AP uses a deep hierarchical routing matrix, while VPR is only able to model 2D-mesh routing fabrics. Our results indicate that ATR's shallower, 2D-mesh routing fabric–in contrast to the AP's deep hierarchical fabric–reduces resource requirements by up to $4.2x$ for difficult-to-route benchmarks. We then identify two properties of automata that cause inefficient resource usage on the AP: average automaton subgraph size and average fan-out. We show that highly connected automata that span a significant portion of the chip are difficult to place-and-route efficiently in the deep hierarchical design of the AP. This result provides application specific insight into how future AP architectures should be designed.

II. Background

A. Automata Processing

Non-deterministic finite automata (NFAs or simply finite automata) are defined by a set of states and transition rules between states. Transitions in a finite automaton are driven by an input tape of symbols that are globally visible. For each input symbol, each active state in the automata considers the symbol and causes transitions according to its own transition rules. Each automaton has one or more start states that initiate computation. Each automaton also has one or more "accept" states. If an input symbol causes the automata to enter into an accept state, the ID of the accept state and the current position in the input symbol tape are reported.

Automata processing can consider any arbitrary number of automaton graphs and activity within those graphs. On von Neumann architectures, automata are usually simulated by consulting the transition rules for every active automata state in a loop. Therefore, naively, automata processing performance is related to the number of active states or "active set," and the number of frequently visited states or "visited set." Active set determines the bandwidth required from the memory hierarchy, and visited set determines the size of the cache or other memory structure required to effectively serve all memory requests [11].

Finite automata can be transformed into deterministic finite automata or DFAs. DFAs are constructed to only ever take one transition per input symbol, reducing active set and bandwidth requirements. However, DFAs can have exponentially more states than equivalent NFAs, greatly increasing storage requirements and visited set. Therefore, full DFA conversion is usually impractical for large, real-world finite automata. Simulation of many parallel automata and required transitions can cause poor performance in von Neumann architectures. However, automata can be minimized to reduce redundant states and redundant computation [11]. This process is analogous to circuit minimization techniques used in FPGA logic synthesis tools.

Spatial architectures (i.e., architectures with arrays of reconfigurable processing elements such as FPGAs) can place-and-route automata states in a reconfigurable fabric and do not need to access a memory store. This allows direct NFA implementation, and as long as the automata graph can fit within the resources of the spatial architecture, all states can operate in parallel and process a single input symbol per cycle. Therefore, for large, highly active automata, spatial architectures can be several orders of magnitude faster than von Neumann architectures for finite automata simulation [11].

B. Micron's Automata Processor

Micron, leveraging their experience in DRAM, has developed the Automata Processor (AP) [15]. The AP re-purposes the parallel address decode logic and parallel bit look-up inherent in DRAM arrays as a large set of dense parallel NFA rule look-up-tables. Coupled with a small amount of logic, each DRAM column can encode a state transition rule and state storage bit, and is dubbed a "state transition element" or STE. An illustrative diagram describing the micro-architecture of an STE is shown in Figure 1.

An STE acts as a type of logic gate. If an STE received an input signal on the previous cycle, it checks its DRAM column look-up-table to see if the current symbol is in its symbol set. If the STE matches the current input symbol, it activates and transmits an output signal to child STEs. Each STE receives and transmits signals to other STEs via a hierarchical, on-chip routing matrix. Input symbol broadcast, matching, and enable-signal broadcast happen within a single AP cycle.

The first-generation AP is divided into two disjoint "half-cores." Each half-core has 96 blocks. Each block has 16 Rows and thus each AP chip has 3,072 Rows. Each Row has 16 "groups-of-two" or GoTs. Each GoT has two STEs. GoTs have two input ports but only one output port. The output port is MUXed to select between either STE's output or the logical

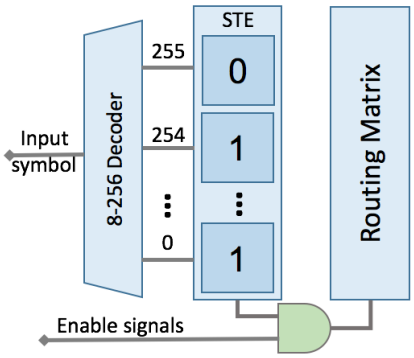

Fig. 1. An Automata Processing state transition element or STE. STEs repurpose memory columns as 8-input/1-output look-up-tables used for matching symbols from the input stream. Coupled with state logic, STEs implement a homogeneous automata state. Similar to an FPGA, inputs and outputs to STEs are routed through a reconfigurable routing matrix.

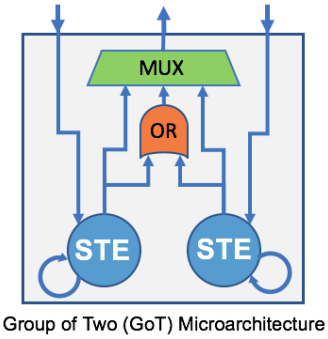

Fig. 2. Groups-of-Two (GoT) [21] can hold two STE nodes. Each STE has an input, can enable its pair STE, or enable itself. The output enable signal is chosen between the left STE, right STE, or the logical OR of both outputs.

OR or their output signals. The micro-architecture of a GoT is detailed in Figure 2. The hierarchy and organization of AP structures is shown in Figure 3.

Analogous to final states in a finite automata, STEs can "report" by sending their signal from a GoT to one of two reporting ports in a Row. Each Row's reporting ports are statically routed to a particular portion of a reporting region [22] and do not use the general purpose routing fabric.

The re-purposing of DRAM arrays as look-up-tables, and the addition of a reconfigurable routing matrix, make the AP a form of spatial architecture. While the AP has been shown to accelerate a large number of application domains, it is just a single design point. This paper is the first study that quantitatively motivates changes to this architecture using a large and diverse benchmark suite of automata applications.

III. Automata-to-Routing Toolchain

This section first describes the three tools used to enable Automata-to-Routing: ANMLZoo, VASim, and VPR. We then describe the ATR architecture and how it enables spatial automata processing architecture research.

A. ANMLZoo Automata Benchmark Suite

ANMLZoo [11] is a public repository of 14 automata processing benchmarks and corresponding input stimuli. Twelve benchmarks are from real-world use-cases for automata processing and two are synthetic. Each benchmark is roughly classified into a "family" of automata based on the topology and size of disjoint automata subgraphs. The first family are regular expression automata (Dotstar, Brill, PowerEN, ClamAV, Protomata). Regular expressions tend to correspond to long, narrow automata, with low topological complexity. "Mesh" automata (Hamming, Levenshtein) have regular, 2-dimensional properties and can grow quadratically with problem size. "Widget" automata (EntityResolution, RandomForest, SPM) are generally composed of smaller custom automaton engines with complex topology that can vary in individual size and complexity depending on application and problem size. Synthetic automata (BlockRings, CoreRings) are designed to test particular properties of execution engines while controlling for automata properties such as active set and visited set.

Because both the number of automata states and routing complexity affect fabric utilization, there is no "right" way to standardize automata benchmarks. Instead of picking one metric, each ANMLZoo benchmark is compiled to completely fill the resources of an AP chip, thus standardizing for both state and routing resources at the same time. This methodology allows easy and fair comparisons of the capacity and routing capabilities of the AP versus other spatial architectures.

B. VASim Virtual Automata Simulator

VASim [19] is an open-source platform for manipulation, optimization, and simulation of finite automata. VASim offers an easy-to-use, object oriented view of automata-processing directed graphs and is designed from the ground up to be an easy to understand research platform for automata processing application and architecture research.

VASim offers traditional, well known automata minimization passes, e.g. prefix merging. Prefix merging is an algorithm, analogous to circuit minimization, that can greatly improve processing performance in von Neumann automata processing engines, and reduce resource requirements for automata implemented on spatial architectures [11]. VASim allows automata researchers access to these standard algorithms without the need for lengthy re-implementation. Furthermore, VASim also acts as an open-source repository for new automata optimization algorithms so that researchers can easily share their optimizations with others and use them for both application and architecture research.

VASim can also enforce architecture specific design rules. While von Neumann automata processing architectures generally do not place any restrictions on the number of states, fan-in, and fan-out allowed in an automata graph, spatial architectures are usually very sensitive to these parameters. Just as in FPGA place-and-route, a large fan-in or fan-out in an automaton might prevent a graph from being routable. For ATR, we extend VASim to automatically enforce fan-in requirements of spatial architectures.

C. Versitile Place and Route

Versatile Place and Route (VPR) [20] is a well known open-source tool for research on tiled, "island style" spatial architectures and is widely used in the FPGA design community. VPR allows researchers to define their own spatial

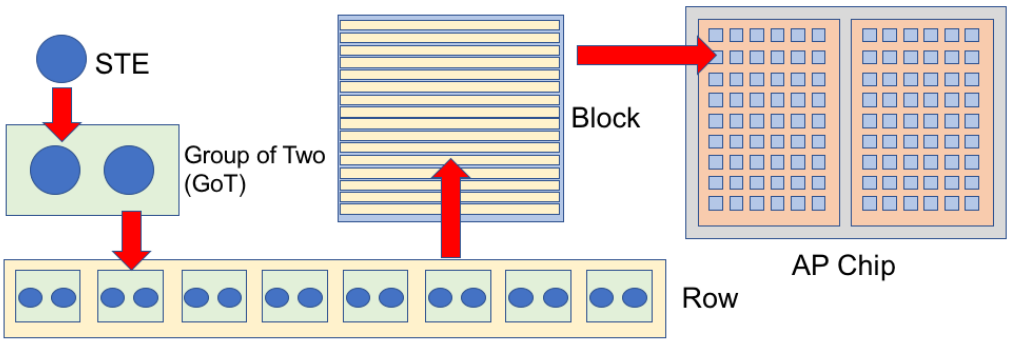

Fig. 3. Elements in the Micron AP architecture. State Transition Elements (STEs) are groupted into "Groups of Two" (GoTs). Eight GoTs form a Row. Sixteen Rows form a Block. Ninety-six blocks form a half-chip. Two distinct half-chips form the AP. We choose the Row as an appropriate tile for the ATR model AP because it is the first element that has access to the AP's routing matrix.

fabrics using an XML-based parameterizable architecture description language. The language describes fabric dimensions (number of logic tiles wide and tall), switch-block architecture, connection-block architecture, channel width, and logic-tile architecture. Logic tiles can be hierarchical, and may be composed of smaller elements such as basic LUTs and registers connected using a basic, but fairly flexible interconnect library.

Once an architecture description file has been defined, VPR takes a Berkeley logic interconnect format (.blif) logic circuit netlist as input. VPR packs each logic tile with circuit elements, places and optimizes placement of tiles in the architecture, and then routes each tile within the reconfigurable fabric.

While VPR is geared towards placement and routing of gate-level logic, VPR also allows place-and-route of "black box" elements, to represent hard logic such as multipliers that may exist as non-LUT computation blocks in the FPGA fabric. VPR is so general, that it allows the definition of *any arbitrary* black box element, not just traditional FPGA logic elements. Thus, in ATR, we use VPR to model reconfigurable arrays of automata state transition elements, rather than traditional logic gates. This new use-case for VPR highlights its flexibility as a back-end architecture description and place-and-route tool for *any* application domain targeting an island style reconfigurable array of processors, as long as a benchmark suite and optimization and transformation tools exist to support this domain.

D. ATR Toolchain Architecture

Automata-to-routing (ATR) combines ANMLZoo [11], VASim [19], and VPR [20] into a new toolchain for spatial automata processing research. Figure 4 describes the high-level architecture of the ATR toolchain. First, automata are fed into VASim. VASim is responsible for optimizing automata (e.g. applying prefix merging) and applying transformations to enforce user-specified design rules. One particular transformation required for this work, fan-in relaxation, is described below. Once automata have been optimized and transformed to fit the design rules of a particular architecture, VASim emits automata netlist in a VPR readable .blif format. ATR then feeds the .blif circuit and spatial automata processor description file defined by the researcher to VPR. VPR then packs, places, and routes the automata circuit into the architecture.

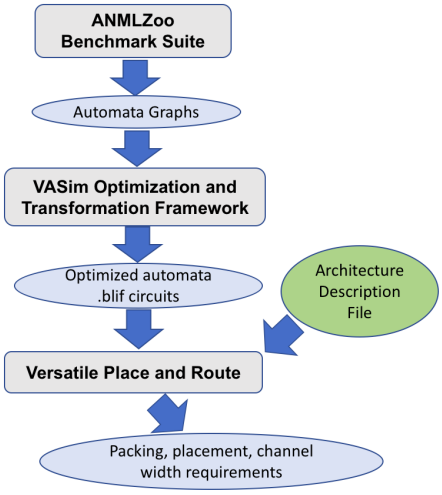

Fig. 4. The Automata-to-Routing or ATR toolchain flow. ANMLZoo applications are used to evaluate automata architectures. These automata graphs are fed to VASim which parses and optimizes the automata. VASim can also enforce design rules on automata and automatically transform them to fit an architecture without changing the semantics of the automata. VASim emits these automata graphs as .blif circuit files for corresponding automata processing spatial architecture models. VPR takes an architecture description and places-and-routes circuits in this hypothetical architecture.

Results from place-and-route can then be used to evaluate the "goodness" of a given architecture depending on desired properties. ANMLZoo provides a large set of diverse benchmarks to fairly evaluate the relative merits of hypothetical and real automata processing architectures.

Together, ANMLZoo, VASim, and VPR form Automata-to-Routing, a powerful spatial architecture research toolchain. The following sections describe modifications to the open-source VASim tool to enable this toolchain and results and conclusions from placing and routing ANMLZoo benchmarks on both real and ATR modeled spatial automata processing architectures.

IV. VASim Extensions

A. Design Rule Transformation: Fan-in Relaxation

We add one new transformation to VASim to better allow it to serve spatial-architecture researchers: fan-in relaxation.

While abstract automata states can have arbitrarily large fan-in, spatially-routed automata can have fan-in restrictions based on the underlying spatial architecture's routing matrix and tile architecture. Fan-in relaxation duplicates a state that has a fan-in that violates a maximum defined by the architect.

When duplicating a state, inputs to the original state are divided among the new duplicate states, while output edges are copied. A picture illustrating a simple example of input duplication is show in Figure 5. If the fan-in was N before relaxation, fan-in is guaranteed to be at most $ceil(N/2)$ after relaxation. Note that this technique doubles the fan-in of the child states of the duplicated state. This algorithm proceeds in a breadth-first manner to ensure that the required fan-in is enforced for all nodes in a single pass.

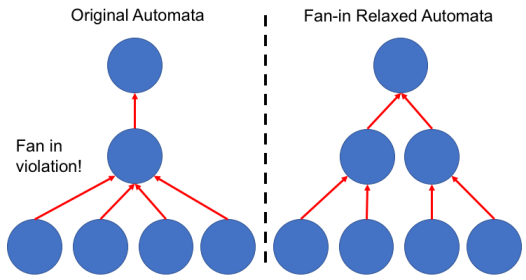

Fig. 5. Fan-in relaxation example. The maximum fan-in is reduced from 4 to 2 by duplicating a state.

Three applications from the ANMLZoo automata benchmark suite have high fan-ins: ClamAV, EntityResolution, and Snort. All three required the fan-in relaxation transformation before successful place-and-route on the architectures modelled below.

B. Design Rule Transformation: Group-of-Two Grouping

In order to save logic and routing resources on-chip, the AP D480 couples every two STEs into a *Group-of-Two* or GoT. A Group-of-Two allows configurable connectivity between a pair of STEs without consuming global routing resources [21]. A figure describing the GoT functionality is shown in Figure 2. VPR is currently unable to model the logical OR of signals, and so, we implement a pass in VASim to identify pairs of STEs that might use this OR-gate and pre-group them into a GoT element.

C. .blif Emission Algorithm

VASim is also extended to convert abstract automata to .blif circuit files readable by VPR. .blif files have three distinct sections that define a circuit. The first section defines the top level input/output signals. The second section describes all instantiations of circuit elements. The third section describes module definitions of each circuit element.

We guide VASim to emit a single clock input pin and no output pins. Because I/O is handled by specialized IP blocks in the AP and not by individual pins in the chip, we essentially ignore I/O pins and turn off dangling block removal in VPR. Output reporting signals for STEs are left unconnected to better match the AP's statically routed reporting architecture [22].

We then guide VASim to emit a circuit element for each STE and pre-grouped GoT in the automata graph. Each STE and GoT lists its input-enable ports, assigning the wire names of each STE that connects to it. It also lists a single output wire that can connect to other STEs in the netlist. We finally guide VASim to emit circuit element module definitions for STEs and GoTs. While this methodology is specific to Micron's AP, ATR is flexible to support the addition of any arbitrary processing elements and I/O architectures. Thus, allowing researchers to investigate the impact of new, hypothetical automata-processing elements.

V. Modelling Micron's Automata Processor

In order to demonstrate the usefulness of ATR, we first attempt to model Micron's Automata Processor in the VPR architecture description language as accurately as possible. By attempting to accurately model the AP, we can create a *baseline architecture* from which we can evaluate potential architectural changes. Thus, researchers will be able to explore new spatial architecture design parameters (such tile complexity or channel width), new hardware features, or entirely new spatial architectures for automata processing. Previously, no such capability was available to researchers, inhibiting advances in this area.

A. Defining A Baseline Tile Architecture

Because the AP's routing matrix is hierarchical in nature, and the VPR tool assumes a 2D mesh-style routing fabric, the AP architecture cannot be modeled 100% faithfully. We must choose a layer of the AP's hierarchy (described in Figure 3) to expose to VPR's 2D routing fabric as the base level tile to model. We choose the AP Row as the tile abstraction for our baseline AP model. The AP Row is the first AP structure to directly connect the global reconfigurable routing network [15] and thus is a natural (although admittedly imprecise) structure to implement as the base tile.

B. Defining A Baseline Routing Network

Once the baseline tile abstraction has been defined, we can design our baseline routing architecture to model the corresponding level of the AP's routing network.

Detailed architectural descriptions of the AP's routing network have not been made publicly available. However, patents filed by Micron that correspond to the AP give us a high-level idea of how tiles might connect to the routing matrix [21]. Figure 6 describes the assumed routing matrix organization of the AP. This organization is presumptive and not authoritative. The rest of this paper refers to the presumptive modeled architecture as the "model AP."

Each model-AP Row has 8 input and 8 output connections to the global routing matrix. GoTs within a Row can receive input from any of these input wires via the intra-row interconnect matrix, and can send output to a single output wire. GoTs can choose output from either internal STE, or the OR of their output. VPR is unable to model the GoT's OR gate functionality, and so these structures must be identified using VASim's GoT identification pass (discussed in Section IV-B) prior to VPR packing and placement. Final states report activations via two reporting ports statically placed in each row structure.

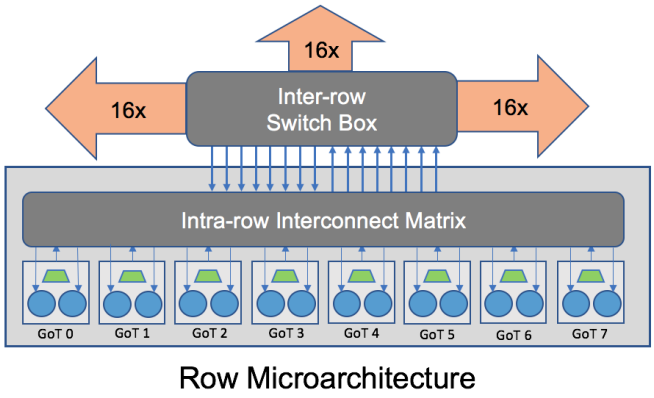

Fig. 6. Model-AP routing architecture configuration with channel width of 16, and Row architecture with 8 Groups-of-Two (GoT). Each GoT has two inputs, but selects a single output between either STE or the OR or their outputs as detailed in Figure 2.

The model-AP Row routing switch block is 16 tracks wide, suggesting a channel width of 16. Because channel width and wire segment length are co-designed and unknown for the real AP hardware, we pick a segment length of one. We leave refinement of these parameters for future work.

VI. Place-and-Route Results

We use the ATR toolchain to place-and-route each ANMLZoo automata benchmark on our baseline representative AP architecture. Three ANMLZoo benchmarks (ClamAV, EntityResolution, and Snort) were transformed using fan-in relaxation to have maximum fan-ins of 8. We measure and report the number of rows required to successfully pack, place, and route each benchmark. VPR was configured to prevent unrelated clustering. Unrelated clustering allows automata states from different disjoint subgraphs to be placed into the same tile. While this generally allows for a denser packing, it can greatly increase channel width requirements, and is turned off to satisfy the model AP channel width requirements. Place-and-route results are presented below.

A. Tile Resource Requirements

Figure 7 plots the tiles (AP Rows) required to place and route each ANMLZoo automata benchmark. The first bar represents the performance of ATR with VASim's automata prefix-merging optimizations turned off. The second bar represents the performance of ATR with VASim's prefix-merging optimizations turned on. VASim's optimizations attempt to simulate optimizations performed in Micron's automata compiler, and are thus always considered for fair model comparison. The third bar represents performance of ATR with optimizations and GoT grouping turned on. GoT grouping only makes a difference if automata states benefit from the internal OR gate functionality. Figure 7 also plots the number of tiles the Micron AP compiler requires to place-and-route each ANMLZoo benchmark.

Some model AP results closely match the results from Micron's compiler and architecture. For ClamAV, Dotstar, Fermi, Protomata, RandomForest (RF), Snort, and BlockRings, the number of model AP tiles are within $2.1\% - 9.5\%$ of the real AP compiler stack and architecture. These particular application results indicate that we are able to model the AP architecture and compiler stack.

Some applications actually require fewer tile resources than the AP when placed-and-routed in our model. For Brill, EntityResolution (ER), Hamming, Levenshtein, and SPM, the number of model AP tiles is considerably lower than the real AP. This indicates that, for these applications, the model AP architecture performs better than the real AP hardware and compiler stack. We discuss possible reasons for improved performance over the AP architecture and suggest improvements to the AP to take advantage of these results in Section VII.

B. Routing Resource Requirements

VPR uses a binary search to identify the smallest number of required tracks per channel to successfully route a circuit in a reasonable amount of time. This metric can be used as a proxy for how difficult the placed circuit was to route. A higher minimum required channel width indicates a more difficult routing problem, and suggests that the architecture requires at least that many routing tracks to successfully implement that circuit. Figure 8 shows the minimum channel width requirements derived by VPR for each ANMLZoo benchmark for the model AP.

Most benchmarks can be routed with 6 tracks per channel. However, three applications required 14 tracks per channel. All applications have a minimum channel width requirement less than the model AP's 16-wide channels.

VII. Evaluating the AP's Routing Matrix Using ATR Modelling

While the Automata Processor is capable of routing many applications in ANMLZoo with the same efficiency as the model AP, the previous section identified five benchmarks (Brill, EntityResolution, Hamming, Levenshtein, and SPM) where the model AP performs much better. In the case of Levenshtein, the model AP is able to route all automaton subgraphs while using $4.2x$ fewer tile resources.

We suspect the reason for this improvement is the difference in the routing matrix topology. The AP's hierarchical routing matrix has four levels [15]. In contrast, VPR's underlying routing matrix assumes a 2D-mesh, island-style architecture with only one layer. Because they lack a deep hierarchy, 2D-mesh fabrics are much more flexible in the graph size and graph topologies they can place-and-route efficiently. To highlight this difference between tree-based routing fabrics and mesh-based routing fabrics, we plot each ANMLZoo benchmark as a function of its average automaton subgraph size and the average node fan-out. These results are shown in Figure 9.

The darker region encompassing in Figure 9 highlights where automata are difficult to route using a hierarchical routing matrix. Importantly, these automata have both high connectivity, and large subgraph sizes.

Brill is a large tree structure with a single, very large automaton subgraph. However, large automaton subgraphs are not enough to cause poor performance in the AP's routing matrix. For instance, CoreRings and RandomForest both have average automata subgraph sizes greater than Brill. The key

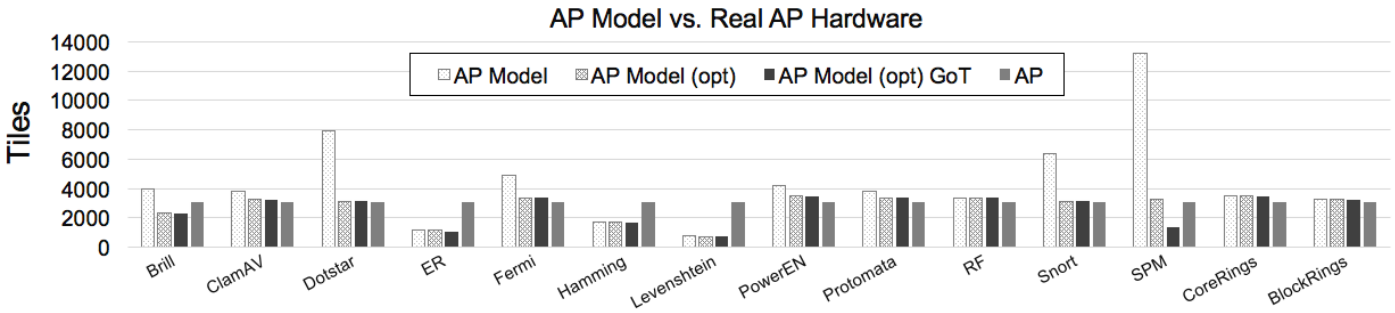

Fig. 7. Compilation results from our AP model implemented in the ATR toolchain and compiled by Micron's AP compiler for the first generation AP D480 chip. "opt" refers to automata graphs optimized using VASim's prefix-merging optimization. "GoT" refers to graphs with pre-grouped GoTs using VASim's GoT grouping pass. ATR is capable of accurately modeling the resource usage of the AP in many cases. Large deviations are due to limitations of VPR's support for deep hierarchical routing matrices.

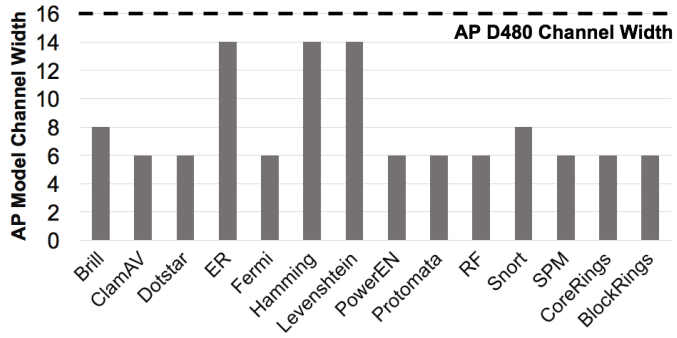

Fig. 8. Minimum channel-width requirements determined by VPR for each ANMLZoo benchmark for the model AP. All benchmarks are able to be placed-and-routed successfully using less than 16 routing tracks per channel, the maximum channel width of the model AP routing matrix.

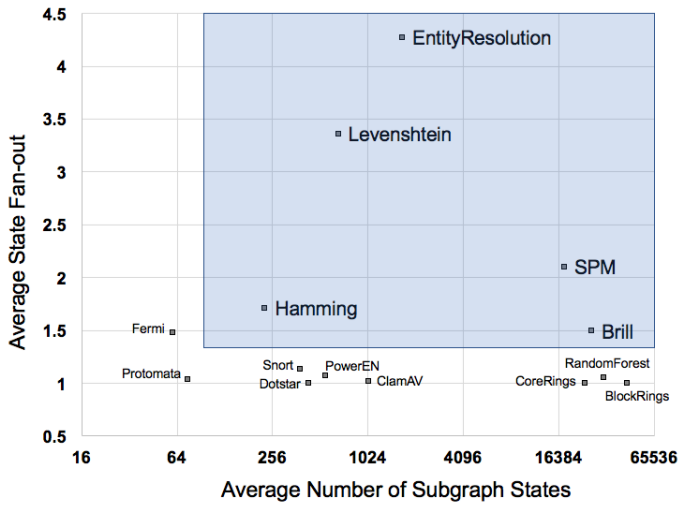

Fig. 9. Each ANMLZoo benchmark plotted as a function of the average size of each disjoint automaton subgraph, and the average fan-out of each node. The darker region highlights an area where automata are larger, with larger fan-out. EntityResolution, Levenshtein, Hamming, Brill, and SPM are all applications where 2D-mesh, spatial-automata processors perform much better than the 4-layer hierarchical routing matrix of the AP.

difference is that Brill has a large enough average fan-out (1.49) that most likely causes congestion in the roots of the AP's hierarchical routing matrix. CoreRings and RandomForest are essentially very large loops, that can be routed fairly easily through the routing hierarchy, even though they are larger than the Brill automaton.

However, some applications like Fermi have relatively similar fan-out (1.48), but route efficiently in the AP. Fermi has relatively complex connectivity, but each automaton subgraph is so small that it is able to fit within just a few tiles. As long as the connections between tiles are relatively few, the AP is able fit these automaton into the leaves of the hierarchical routing matrix without causing congestion in the roots.

EntityResolution and Levenshtein represent more challenging cases, where both the average automaton subgraph size and fan-out are higher. EntityResolution and Levenshtein must use $2.7x$, and $4.2x$ more tiles respectively when routed using the hierarchical routing matrix than the model AP's 2D-mesh routing fabric.

While the above applications are not easily routed using the AP's hierarchical routing matrix, most ANMLZoo applications, especially those that are regular-expression based, perform very well. Because regular-expression processing is such an important motivation for automata processing acceleration, we conclude that the AP's routing matrix architecture is suitable for regular expression rule-set acceleration in most cases. However, for some applications such as Levenshtein edit-distance automata (an extremely important string comparison kernel for a wide variety of application domains), the AP's routing matrix is not an ideal design.

VIII. Conclusions and Future Work

This paper presented the Automata-to-Routing (ATR) toolchain. ATR is a full-stack, automata-processing toolchain that enables research of spatial automata-processing architectures. Prior to ATR, no research toolchain for spatial automata-processing architecture research toolchain existed, preventing exploration of this exciting new class of architectures. ATR combines three different existing open-source software tools–the ANMLZoo [11] benchmark suite, the VASim [19] open finite automata simulator and optimization framework, and VPR [20], a flexible reconfigurable array simulator and place-and-route tool–into one toolchain that allows researchers to experiment with spatial, automata architectures. ATR enables evaluation of new and existing automata processing-tile organizations, and routing-matrix architectures.

We show that ATR is capable of modelling the logic tiles of existing spatial, reconfigurable, automata-processing architectures such as Micron's Automata Processor with sufficient accuracy to enable research on new architectures for spatial automata processing..

We present a study–enabled by ATR–comparing and contrasting performance of the AP's hierarchical routing matrix against a proposed 2D-mesh style routing matrix. We identify four benchmarks where a 2D-mesh routing matrix performs better, using up to $4.2x$ fewer logic tiles. We also characterize two properties of automata graphs–average state fan-out and average number of subgraph states–that correlate with poor routability on the AP.

Future work is needed on further extensions to the ATR tools, design-space exploration of spatial automata processing tile microarchitectures, identifying improved tile microarchitectures. Future work may also include area and power modelling of spatial automata-processing architectures to identify additional trade-offs between logic-tile and routing resources.

IX. Acknowledgements

This work was partly funded by C-FAR, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, Achievement Rewards for College Scientists (ARCS), Micron Technologies, and NSF grant no. EF-1124931

References

[1] M. Roesch, "Snort: Lightweight intrusion detection for networks.," in *Proceedings of the USENIX Large Installation Systems Administration Conference (LISA)*, 1999.

[2] V. Paxson, "Bro: a System for Detecting Network Intruders in Real-Time," *Computer Networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[3] ClamAV, "ClamAV Rules." Available at https://www.clamav.net/.

[4] T. Tracy II, Y. Fu, I. Roy, E. Jonas, and P. Glendenning, "Towards machine learning on the automata processor," in *Proceedings of the International Conference on High Performance Computing*, Springer, 2016.

[5] T. Tracy II, M. Stan, N. Brunelle, J. Wadden, K. Wang, K. Skadron, and G. Robins, "Nondeterministic Finite Automata in Hardware—the Case of the Levenshtein Automaton," *Proceedings of Architectures and Systems for Big Data (ASBD), in conjunction with ISCA*, 2015.

[6] K. Wang, E. Sadredini, and K. Skadron, "Sequential pattern mining with the Micron Automata Processor," in *Proceedings of the ACM International Conference on Computing Frontiers (CF)*, 2016.

[7] C. Bo, K. Wang, J. J. Fox, and K. Skadron, "Entity Resolution Acceleration using Micron's Automata Processor," *Proceedings of Architectures and Systems for Big Data (ASBD), in conjunction with ISCA*, 2015.

[8] K. Zhou, J. J. Fox, K. Wang, D. E. Brown, and K. Skadron, "Brill tagging on the Micron Automata Processor," in *Proceedings of the IEEE International Conference on Semantic Computing (ICSC)*, pp. 236–239, 2015.

[9] M. H. Wang, G. Cancelo, C. Green, D. Guo, K. Wang, and T. Zmuda, "Using the Automata Processor for fast pattern recognition in high energy physics experiments—a proof of concept," *Nuclear Instruments and Methods in Physics Research*, 2016.

[10] J. Wadden, N. Brunelle, K. Wang, M. El-Hadedy, G. Robins, M. Stan, and K. Skadron, "Generating efficient and high-quality pseudo-random behavior on Automata Processors," in *Proceedings of the 2016 IEEE 34th International Conference on Computer Design (ICCD)*, pp. 622–629, Oct 2016.

[11] 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 IEEE International Symposium on Workload Characterization (IISWC)*, 2017.

[12] Y. H. E. Yang and V. K. Prasanna, "Space-time tradeoff in regular expression matching with semi-deterministic finite automata," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, 2011.

[13] X. Wang, "Techniques for efficient regular expression matching across hardware architectures," Master's thesis, University of Missouri-Columbia, 2014.

[14] M. Becchi, *Data structures, algorithms and architectures for efficient regular expression evaluation*. PhD thesis, Washington University in St. Louis, 2009.

[15] P. Dlugosch, D. Brown, P. Glendenning, M. Leventhal, and H. Noyes, "An efficient and scalable semiconductor architecture for parallel automata processing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3088–3098, 2014.

[16] Y. Fang, T. T. Hoang, M. Becchi, and A. A. Chien, "Fast support for unstructured data processing: the unified automata processor," in *Proceedings of the ACM International Symposium on Microarchitecture (MICRO)*, pp. 533–545, 2015.

[17] K. Atasu, F. Doerfler, J. van Lunteren, and C. Hagleitner, "Hardware-accelerated regular expression matching with overlap handling on ibm poweren processor," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1254–1265, 2013.

[18] V. Gogte, A. Kolli, M. J. Cafarella, L. D'Antoni, and T. F. Wenisch, "HARE: Hardware accelerator for regular expressions," in *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, Oct 2016.

[19] J. Wadden and K. Skadron, "VASim: An open virtual automata simulator for automata processing application and architecture research," Tech. Rep. CS2016-03, University of Virginia, 2016.

[20] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Proceedings of the International Workshop on Field Programmable Logic and Applications*, pp. 213–222, Springer, 1997.

[21] D. Brown, H. Noyes, I. Xu, and P. Glendenning, "Methods and systems for routing in a state machine," Mar. 25 2014. US Patent 8,680,888.

[22] Micron Inc., "Designing for the Micron D480 Automata Processor." http://www.micronautomata.com/documentation/anml_documentation/c_D480_design_notes.html.