An Overview of Micron's Automata Processor

[1]Ke Wang, [1]Kevin Angstadt, [1]Chunkun Bo, [1]Nathan Brunelle, [1]Elaheh Sadredini,
[2]Tommy Tracy II,[1]Jack Wadden, [2]Mircea Stan, [1]Kevin Skadron
[1]Dept. of Comp. Sci., [2]Dept. of Elec. & Comp. Eng.
University of Virginia
Charlottesville, VA, 22904 USA
{kewang, angstadt, chunkun, njb2b, elaheh, tjt7a, jpw8bd, mircea, skadron }@virginia.edu

ABSTRACT
Micron's new Automata Processor (AP) architecture exploits the very high and natural level of parallelism found in DRAM technologies to achieve native-hardware implementation of non-deterministic finite automata (NFAs). The use of DRAM technology to implement the NFA states provides high capacity and therefore provide extraordinary parallelism for pattern recognition. In this paper, we give an overview of AP's architecture, programming and applications.

Keywords
Finite automata, Processor in Memory, DRAM, Data Mining

1. MOTIVIATION
As we collect more and more data about the world around us, and digitize more and more artifacts from our past, "big data" problems abound in every field of inquiry. In a recent survey of senior decision-makers from nine industries and ten countries, 70% of respondents consider their business's ability to exploit big data critical to their future success. Real-time processing is also increasingly important, as richer sensing and data collection allow meaningful interventions, in contexts ranging from healthcare to cybersecurity. However, many of the questions we want to explore with these data remain unanswerable, because we lack sufficient computational resources to analyze huge data sets in a timely and cost-effective fashion.

This is especially important because data sets are growing much faster than computing capacity. The Computer Sciences Corporation reports that the amount of data being generated by individuals and companies will be 44 times greater in 2020 than it was in 2009 [1]. Yet Moore's Law is slowing down, due to power constraints, limits on scalability of CMOS, and limits of von Neumann architectures to support high degrees of parallelism—especially irregular parallelism. Hardware accelerators help address this problem, as specialization allows greater efficiency. But for general-purpose systems, as in data centers, only accelerators with broad applicability are likely to gain widespread adoption.

Pattern-based algorithms (pattern matching, pattern recognition, etc.) are exceedingly common in data mining, cybersecurity, bioinformatics, and many other application domains. A common feature in many of these applications is the importance of inexact matching to identify groups or patterns with non-trivial edit distances from their reference patterns (e.g., DNA alignment in the presence of mutations). Each step in edit distance complicates the data structures and computations required to find an inexact match. This makes inexact pattern recognition extremely difficult across huge datasets, and often leads to using less accurate heuristics or restricting the allowed edit distance, at the expense of accuracy. Pattern-based algorithms are sometimes used with subsequent supervised-learning techniques, e.g., for classification, but in many cases, mining patterns of interest is the most expensive task.

Micron Technology has recently introduced a new processing architecture, the Automata Processor (AP) [2], that is a native-hardware implementation of a classic computational model, *non-deterministic finite automaton* (NFA). NFAs are primarily symbolic pattern-matching machines, which on one hand limits their generality. However, true NFAs are extraordinarily powerful at this task, because allowing an arbitrary number of states to be active at the same time allows massive parallelism.

2. ACHITECTURE OF THE AP PROCESSOR
2.1 Overview
In von Neumann architectures, the most efficient method to support finite automata (NFA or DFA) is a transition table in memory, with each table entry representing a state, and pointing to its successor state(s). Since an arbitrary number of states can be active and matching against the input every clock cycle, NFAs are a poor fit for von Neumann architectures, which cannot efficiently support the potentially large number of random accesses to the memory system every cycle. We have found that even GPUs' massive parallelism and bandwidth are unable to accommodate the high bandwidth demands of NFAs, with tens to hundreds of active states, and thus random-access memory lookups for each input symbol. Regular expression processing and NFA processing are thus generally converted into deterministic finite automata (DFAs), in which only one state can be active. This, however, leads to an exponential increase in the number of states, because every possible path through the NFA potentially needs a unique set of states in the DFA. The DFA transition tables are often very large, with very high miss rates in

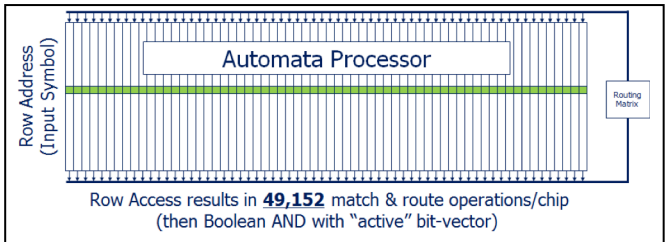

Figure 1 - Micron's Automata Processor design (courtesy Micron).

CODES/ISSS '16, October 01-07, 2016, Pittsburgh, PA, USA
 ISBN 978-1-4503-4483-8/16/10…$15.00
DOI: http://dx.doi.org/10.1145/2968456.2976763

the L1 (often 100%, in our studies), and high miss rates even in last-level caches.

A native-hardware implementation of NFAs is therefore extraordinarily powerful, because allowing an arbitrary number of states to be active at the same time allows massive parallelism—limited only by the available number of state elements—and allows NFAs to explore many permutations concurrently. Micron's AP (Fig. 1) uses memory in a different way. Each column in the memory array represents one NFA state. The AP chips have a native input symbol of 8 bits, so each of the 256 possible input bytes activates the corresponding row of the array, reading out the response of the entire NFA to that symbol. These are combined with a bit vector indicating which states are active, and the AP's routing matrix forwards state transitions. This architecture leverages the inherent bit-level parallelism of a memory array and allows any or all of the states to respond to each input symbol concurrently, every clock cycle.

2.2 AP Board

The current-generation AP boards, with 32 AP chips operating at 133 MHz, are standard PCI-Express boards, although our group also explores other system architectures. The current boards allow the AP chips to be partitioned among multiple concurrent data streams, allowing full utilization of the available interface bandwidth for problems in which the automata of interest do not fill the entire board.

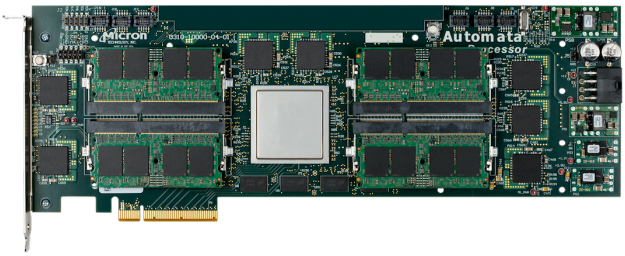

Figure 2 – Testing Board of Micron's Automata processor

The AP boards also contain an FPGA. At the moment, this only implements the PCIe-bus host as well as the DRAM memory controllers for interfacing with the AP ranks. (The AP chips currently require a non-standard DDR interface.) However, this FPGA will allow programmers to develop acceleration pipelines or loops that use the AP to accelerate pattern-matching tasks, and use the FPGA hardware acceleration of other tasks, minimizing interactions with the host CPU. Micron has disclosed an effort to develop support for user programming of the FPGA and efficient FPGA-AP communication.

Launching a program on the AP follows a classic offload model. The host CPU performs an initial configuration step to instantiate the desired automata, sends data to be the AP, and then retrieves the results. The first-generation boards have 4GB of on-board buffering, allowing data transmission and processing in parallel. The AP also supports fast reconfiguration; a full reconfiguration of the board takes only 50 milliseconds. If the connectivity does not need to be changed, and only the matching symbols need changing, this is slightly faster. In our work so far with diverse applications, such as association rule mining and DNA algorithms, we find such symbol replacement to be a common building block, because we find that a common paradigm is to create a macro to represent some pattern-matching template, e.g. to count instances of sequences within an edit distance of k, and then populate the board with as many of these macros as possible and load them with patterns of interest. If the set of candidates of interest exceeds the board's capacity, multiple passes are required, but only the patterns—not the macro structures—need updating. This paradigm is especially valuable because it moves compilation overhead out of the critical path.

2.3 Programming

AP software, integrated as an AP SDK package, allows development, testing, and performance modeling. It is available through Micron's new automata developers' portal (www.micronautomata.com). The Micron's AP SDK provides Automata Network Markup Language (ANML), an XML-like language for describing automata networks, as well as C, Java and Python binding interfaces to describe automata networks, create input streams, parse output and manage computational tasks on the AP board. A "macro" is a container of automata for encapsulating a given functionality, similar to a function or subroutine in common programming languages. Deploying automata onto the AP fabric involves two stages: placement-and-routing compilation (PRC) and loading (configuration) [1]. In the PRC stage, the AP compiler deduces the best element layout and generates a binary version of the automata network. In the cases of large number of topologically identical automata, macros or templates can be precompiled in PRC stage and composed later [13]. This shortens PRC time, because only a small automata network within a macro needs to be processed, and then the board can be tiled with as many of these macros as fit. A pre-compiled automaton only needs the loading stage. The loading stage, which needs about 50 milliseconds for a whole AP board [13], includes two steps: routing configuration/reconfiguration that programs the connections, and the symbol set configuration/reconfiguration that writes the matching rules for the STEs. The changing of STE rules only involves the second step of loading, which takes 45 milliseconds for a whole AP board. The feature of fast partial reconfiguration plays a key role in a successful AP implementation several applications.

3. APPLICATIONS

NFAs naturally support efficient execution of complex regular-expression processing tasks. In this paper, we elaborate on additional applications enabled by fast NFA processing, other than regular-expression processing.

3.1 Bioinformatics

The DNA (l,d) motif search problem is known to be NP-hard, and the largest solved instance reported to date is (26,11). Roy and Aluru [3] proposed a novel algorithm using streaming execution over a large set of NFAs and achieved over 200X speedups on an AP board over a 48-core CPU cluster. They also demonstrated PROTOMATA [4], an AP-accelerated protein motif algorithm, and achieved up to a half million times speed-up over single-threaded CPU with one AP board.

Edit distance matching is an important computational kernel of many bioinformatics applications. The Levenshtein NFA recognizes input strings within a set edit distance of a configured pattern in linear time. Tracy et al. [5] introduced a novel technique for executing a pipelined Levenshtein NFA using the AP, avoiding the run time and space overheads associated with CPU and GPU implementations. The experiments show that run time remains linear with the input while the space requirement of the automaton becomes linear in the product of the configured pattern length and edit distance. These properties allow the AP to execute large instances of the Levenshtein NFA or many small instances in parallel, thus making the automaton a viable building block for future approximate string applications on the AP.

3.2 Data Mining

Association rule mining (ARM) is a widely used data mining technique for discovering sets of frequently associated items in large datasets. Frequent Set Mining (FSM) and Sequential Pattern Mining (SPM) are examples of ARM techniques that learn associations among variables in structured datasets. They have become important data mining techniques with broad application domains in business, health, software engineering, cybersecurity, etc. [6]. A frequent set is simply a set of items that often show up together in many transactions. A sequential pattern refers to a hierarchical pattern consisting of a sequence of frequent transactions (itemsets) with a particular ordering among these itemsets. Wang et. al [7, 8] recently demonstrated the AP solutions for accelerating both SPM and FSM. Up to 129X and 49X speedups are achieved by the AP-accelerated FSM on seven synthetic and real-world datasets, when compared with the Apriori single core CPU implementation and Eclat, a more efficient FSM algorithm, compared to a 6-core multicore CPU. The proposed AP solution also outperforms the state-of-the-art PrefixSpan and SPADE algorithms on multicore CPU by up to 452X and 49X. The AP advantage grows with larger datasets.

3.3 Machine Learning

Part-of-speech tagging is an important step in many natural language processing pipelines and is used to improve the quality of tasks such as speech recognition, and speech synthesis. Brill tagging is a rule-based POS tagger that identifies where a set of pre-learned, contextual rules can be applied to a pre-tagged database, improving the correctness of the tags. Brill tagging can be inefficient because thousands of contextual rule patters may need to be searched for every token of the database. By converting the Brill rules to parallel automata, and loading them onto the AP, all rule patterns can be searched in parallel, improving performance even over multi-core computation. The AP has been shown to accelerate the Brill tagging task by ~31X-63X, over a server-class, multi-core CPU [9].

String kernel (SK) is a widely used technique in the Machine Learning field, particularly for biological sequences analysis. Instead of comparing whole sequences, string kernel methods count the occurrences of representative short subsequences, called K-mers. In the string kernel model, a mapping function projects the input strings to a higher-dimensional (number of K-mers) feature space. Bo et al. [10] proposed to recognize a number of predefined sting kernels, including exact match, mismatch and gappy match by using an AP board. This solution achieved 8x, 25x, 139x, 418x 1438x and 3978x speedups over PatMaN (a rapid CPU alignment tool) for mismatch=0, 1, 2, 3, 4, 5 respectively.

A variety of applications employ ensemble learning models, using a collection of decision trees, to quickly and accurately classify an input based on its vector of features. Tracy et al. recently showed an AP implementation of ensemble learning based on *Random Forest* algorithm [11]. The net result is a solution which when evaluated using two applications, namely handwritten digit recognition and sentiment analysis, produce up to 63X and 93X speedups over single-core, CPU solutions.

3.4 Other Applications

Throughput is a critical factor for successful security check of network intrusions. Roy et al. [4] proposed a parallel intrusion scanning technique by using the AP. As an example, Fast-SNAP network data is scanned for 4312 signatures of intrusion using a single board of the AP at 10.3 Gbps.

Entity Resolution (ER), the process of finding identical entities across databases, is critical to many information integration applications. As sizes of databases explode, it becomes computationally expensive to recognize identical entities for all records with variations allowed. Bo et.al [12] proposed an AP-accelerated ER solution, which accelerates the performance bottleneck of fuzzy matching for similar but potentially inexactly-matched names, and use a real-world application to illustrate its effectiveness. Results show 9.5x to 400x speedups, with 9.2% more correct pairs and 43% better generalized merge distance (GMD) cost over Apache Lucene. 2.8x to 23.2x speedups are achieved compared with a sorting-based method with 18.1% more correct pairs and 51% less GMD cost.

To improve the software development productivity, Angstadt et al. [13] presented RAPID, a high-level programming language and combined imperative and declarative model for programming pattern-recognition processors.

4. SUMMARY

The AP leverages bit-level parallelism to provide native support for efficient NFA execution, enabling dramatic speedups for a variety of algorithms.The University of Virginia established Center for Automata Processing (http://cap.virginia.edu) to build a vibrant ecosystem of researchers, developers, and adopters for the exciting new Automata Processor.

5. ACKNOWLEDGMENTS

This work was supported in part by the NSF (CCF-0954024, CCF-1116289, CDI-1124931, EF-1124931); Air Force (FA8750-15-2-0075); Virginia Commonwealth Fellowship; Jefferson Scholars Foundation; the Virginia CIT CRCF program under grant no. MF14S-021-IT; by C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA; a grant from Micron Technology.

6. REFERENCES

[1] http://www.csc.com/big_data/flxwd/83638-big_data_just_beginning_to_explode_interactive_infographic
[2] P. Dlugosch et al. , An efficient and scalable semiconductor architecture for parallel automata processing. IEEE TPDS , vol. 25, no. 12, 2014.
[3] I. Roy and S. Aluru, Finding motifs in biological sequences using the micron automata processor. In Proc. of IPDPS'14
[4] I. Roy et al., High Performance Pattern Matching using the Automata Processor. In Proc. of IPDPS'16, 2016.
[5] T. Tracy, M. Stan, N. Brunelle, J. Wadden, K. Wang, K. Skadron, G. Robins. In *Proc. of ASBD*, 2015
[6] C. C. Aggarwal and J. Han, editors. Frequent Pattern Mining. Springer International Publishing, Cham, 2014.
[7] K. Wang et al. Association rule mining with the micron automata processor. In Proc. IPDPS '15, 2015.
[8] K. Wang, Elaheh Sadredini and Kevin Skadron. Sequential Pattern Mining with the Micron Automata Processor. In Proc. Computing Frontiers 2016
[9] K. Zhou et al. Regular expression acceleration on the micron automata processor: Brill tagging as a case study. Big Data CA, 2015
[10] C. Bo, K. Wang, Y. Qi, and K. Skadron. String kernel testing acceleration using the Micron Automata Processor.Workshop on Computer Architecture for Machine Learning, 2015
[11] T. Tracy II, Y. Fu, I. Roy, E. Jonas, P. Glendenning. Toward machine learning on the Automata Processor.ISC-HPC 2016.
[12] C. Bo, K. Wang, J. Fox, and K. Skadron. Entity Resolution Acceleration using Automata Processor. In *Proc. ASBD*,2015
[13] K. Angstadt, W. Weimer, and K. Skadron. Proceedings of the ACM International Symposium on Architectural Support for Programming Languages and Operating Systems ASPLOS 2016