International Journal of Grid and Distributed Computing
Vol. 9, No. 4 (2016), pp.53-66
http://dx.doi.org/10.14257/ijgdc.2016.9.4.05

Automata Processor Architecture and Applications: A Survey

Nourah A.Almubarak, Anwar Alshammeri, Imtiaz Ahmad

Department of Computer Engineering, Kuwait University, Kuwait
nourah.almobarak.kuniv@gmail.com, anwar1991@windowslive.com,
imtiaz.ahmad@ku.edu.kw

Abstract

Finite automata-based algorithms are becoming increasingly important for a wide range of application domains to process large volumes of unstructured data. Current processor technologies are not well suited to accelerate massively parallel operations related to the search and analysis of complex and unstructured data streams pattern identification problems. Hardware designers are exploring new processing technologies to accelerate pattern identification problems. One such technology is the recently introduced Micron Automata Processor (AP), which is a novel and powerful reconfigurable non-von Neumann processor that can be used for direct implementation of multiple Non-deterministic Finite Automata (NFAs) running concurrently on the same input stream. AP has a linear-scalable, two-dimensional MISD (Multiple Instruction Single Data) fabric comprised of thousands of interconnected small processing elements called State Transition Elements (STEs) to analyze complex data streams simultaneously to accelerate solving massively complex problems. The AP is promising future technology which provides new operations and new avenues for exploiting parallelism to accelerate the growing and important class of automata-based algorithms. In this paper, we present a survey of the state-of-the-art in automata processor based hardware accelerators. We describe AP hardware architecture, its programming environments, summarize its current successful applications in a wide range of diverse fields and explore future research trends and opportunities in this increasingly important area of automata computing paradigm.

Keywords: *Automata Processor (AP); Computer Architecture; Hardware Accelerators; Non-Deterministic Finite Automata (NFA); Pattern Matching; Regular Expressions*

1. Introduction

In the era of big-data, the amount of data being generated is increasing at a very fast rate due to the explosive growth of on-line cloud services, social-networks, surveillance, and reconnaissance (ISR) systems, scientific advancement, healthcare and genome research among other applications [1]. The data analytics of increasingly data-intensive workloads is becoming critically important in both industry and academia for their future success. Collected data is often analyzed in a multitude of different ways, and many algorithms in areas such as data-mining, bioinformatics, linguistic vocabulary, deep packet analysis and spam filtering require identification of exact or fuzzy match character patterns. Conducting such searches through large data sets demands good support from both hardware and software. Current processor technologies are not well-suited for these tasks: single-threaded processing most naturally identifies a single pattern at a time, and, therefore, requires multiple passes through the data; vector or SIMD processors suffer from inherent branch divergence when analyzing data for multiple patterns; and multi-core processors and clusters often execute far fewer threads than there are patterns to be

ISSN: 2005-4262 IJGDC

identified. Hardware designers are exploring new processing technologies to accelerate pattern identification problems.

Nondeterministic Finite Automata (NFA) is a useful formalism for pattern identification problems that have a large amount of conditional branching such as bioinformatics, text-retrieval, and search engine applications, because of its intuitive mechanism [2]. An NFA consists of a set of states, the input alphabet, the transition function, the set of starting states and the set of accepting states. In finite automata, non-determinism arises when multiple states are active simultaneously, effectively allowing for parallel exploration of the data stream. A von Neumann sequential processor (CPU) can be used to implement NFA directly by creating a group of states and connections. It will keep track of the active states, and take the next token of input to compute and update the next states. Unfortunately, it should work sequentially, and as the number of the active states increases, the processing time will increase relatively. If the entire NFA becomes active the processing complexity will be O (n^2), which is not computationally feasible especially for large size problems. The other way was to translate the NFA to Deterministic Finite Automata (DFA) because the later requires singularity in a state which allows direct processing O (1). However, this approach may lead to exponential storage cost. Even the multi-core CPUs and GPUs cannot deliver the required performance for pattern identification problems [3, 4]. Hence, there is a need for parallel processors that can handle unstructured, complex and big data streams and support the direct implementation of NFA on the hardware.

Direct implementation of NFA in hardware has the potential to be more efficient than software executing on von Neumann architecture. Hardware-based automata can perform simultaneous and parallel exploration of all possible valid paths in an NFA, thereby achieving the processing complexity of a DFA without being subject to DFA state explosion. FPGAs offer the requisite parallelism and a number of relatively recent efforts have explored this direction [5, 6, 7, 8, 9]. However, the architectures based on NFA are limited by the capacity of FPGA chips since the transition table is mapped directly into the FPGA logic. The bottleneck of FPGA implementation comes from a limited number of flip-flops when the number of NFA states becomes larger, it may exceed the FPGA capacity. In addition, in any case, changes in the automata design, the FPGA must be reconfigured. The FPGA synthesis and implementation procedure can take hours and may need some manual adjustments to achieve the desired throughput [10]. Moreover, none of the FPAGs based solution supports streaming, which is an essential feature for pattern identification problems.

As the demands for analyzing more and more data increase, we are faced with new challenges. In particular, traditional architectures where data needs to be fetched from memory as to be processed are becoming a bottleneck (memory wall). A new paradigm called processing-in-memory (PIM) is being explored to avoid the memory bottleneck problem [11, 12]. In this paradigm, the systems are equipped with memory chips that have the capacity of both storage and computation. Memory is augmented with computational elements and thus, computation can be performed locally in memory. With the evolution of new emerging SDRAM technologies, PIM has become of great interests to the research community as well as industry. PIM paradigm presents an opportunity to reduce both energy and data movement overheads, and enhance parallelism.

By exploiting the PIM concept, Micron Company introduced an Automata Processor (AP) [13], which is a non-von Neumann processor that can be used for direct implementation of multiple NFAs at the same time. AP has a linear-scalable, two-dimensional MISD (Multiple Instruction Single Data) fabric comprised of thousands of interconnected symbol processing elements to achieve significant speedup. AP chip can directly implement NFA, which does not require any programming intervention to accelerate massively parallel operations related to the search and analysis of complex and unstructured data streams pattern identification problems. The AP provides better

reconfiguration than FPGA. It is flexible to add new automata to the chip without recompiling the existing automata. Also, it allows adding multiple chips to scale up the capacity and throughput of the design.

The AP has attracted a lot of attention since its release, and a lot of applications have been developed using the AP to solve a variety of problems in diverse fields. Since AP is a promising future technology, in this paper we discuss AP architecture, its programming environment, and survey its applications where AP played the key role in improving their efficiency and the throughput. In addition, key future research opportunities are identified in this increasingly important area. The rest of the paper is structured as follows. Section 2 describes the architecture of the automata processor. Section 3 introduces the programming environment for the AP. Section 4 surveys the applications of the AP to a wide range of problems. Section 5 concludes the paper and explores promising research topics.

2. AP Architecture

Micron's Automata Processor (AP) is a novel and powerful reconfigurable non-von Neumann MISD (Multiple Instruction Single Data) processor that can be used for direct implementation of multiple NFAs running concurrently on the same input stream. The AP is based on an adaptation of memory array architecture exploiting the inherent bit-parallelism of traditional SDRAM to analyze data streams across the chip. The conventional SDRAM consists of a two-dimensional array of rows and columns. Any memory access for read or write operations needs both a row address and a column address. The automata processor modified that strategy and made the row address represent the 8-bit input symbol, and invoke automata operations through its routing matrix structure in place of memory's column address as shown in Figure 1. The input symbol is first decoded (8-to-256 decode) and then provided to the memory array. All paths of the routing matrix are operating simultaneously on the same input symbol, and the memory arrays are distributed throughout the chip, providing O (1) lookup for a 48 K bit memory word [13].

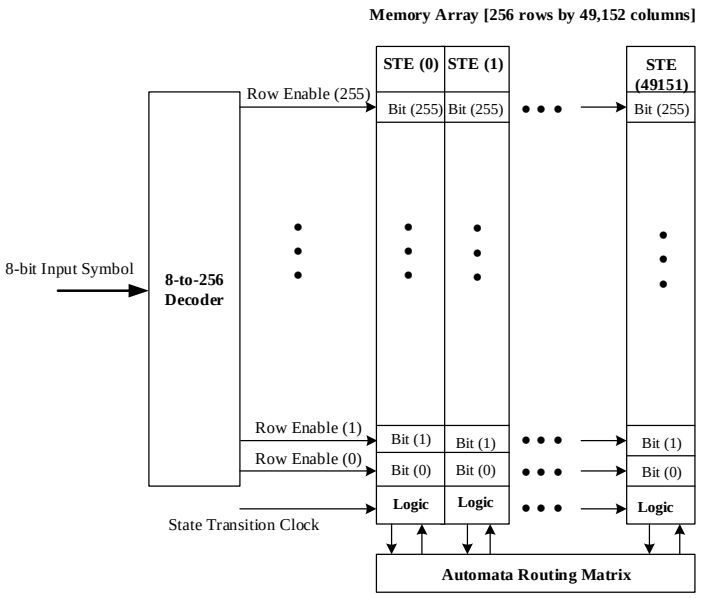

Figure 1. The Automata Processor Architecture [13]

2.1. Computational Elements

There are three major components on the AP: State-Transition-Element (STE), Counter Element and Boolean Element. Counter and Boolean elements are designed to work with STEs to increase the space efficiency of automata implementations and to extend computation capabilities beyond NFAs [13, 14, 15, 16].

2.1.1. The State Transition Element (STE): STEs model the states of NFAs and are the core component with the highest population density designed as a memory array that has control and a computational logic. A single STE logic bit presents in each column of the memory array, which is used to enable inputs, and an output decoder/driver. Their value is set to (1) if the STE is in the active state or set to (0) otherwise. The output is driven by the logical AND which takes the state bit and the output of the associated column of memory, and it outputs only if the selected bits were programmed to recognize an input symbol. One STE can match an 8-bit user-specified symbol in a clock cycle and STEs can activate each other via a reconfigurable routing network. Each STE is designed to recognize an input data value, which can be any character class over 8-bit symbols. These STEs are reconfigurable and can be reprogrammed to recognize new input data values. Each STE has three states: inactive, activated and matched. Only activated STEs will be able to inspect the next input symbol to perform a match against symbols accepted by that STE. Once the symbol on an STE is matched, the STEs connected to it will be activated to accept the next input and match that against their programmed symbol matches.

Automata designs on the AP are composed of a connected, directed network of State Transition Elements (STEs). These STEs are activated when the STEs are enabled by a neighboring STE and the input matches the STE's assigned 8-bit symbol class. STEs that represent a starting state are called Start STEs; those that represent accept state are called Reporting STEs. Multiple starting states are allowed, which enable parallel execution of multiple NFAs. A starting state STE is further classified into two subtypes namely all-input-start STE and start-of-data STE. A start-of-data STE is active only at the starting symbol of the input string, whereas all-input-start STE is active on every symbol of the string. For example, automaton recognizes the word "FLOW" appearing anywhere in an input stream shown in Figure 2.

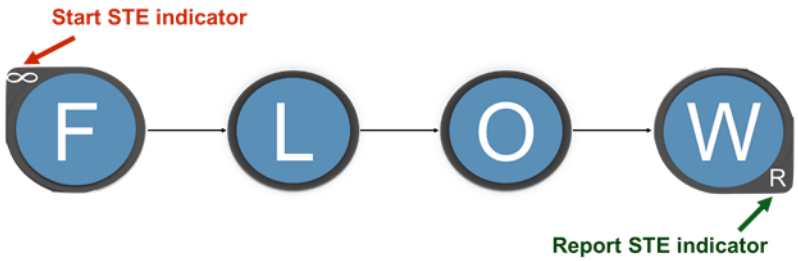

Figure 2. Automaton with All-Input-Start and Reporting STEs

2.1.2. The Counter Element (CE): The counter element is considered as an assistant to the STE to add more modeling capabilities to the Automata Processor. The counter element is a 12-bit counter that can be programmed to report the number of times a particular input data values has been recognized. Every time one of the counts enable signals are asserted, the counter counts by one. When the counter's count equals the pre-configurable count value, an output event is triggered. The counter has implemented several additional features to provide greater flexibility in designing various automata. For example, automata that count the three occurrences of character "A" before a final character "T" is shown in Figure 3.

Figure 3. Counter Element Illustration

2.1.3. The Boolean Element (BE): BE also known as a combinatorial element, is considered as a supplementary element to the STE and can be used to create combinational circuit complementing the NFA designs. These Boolean elements can be programmed to act as AND, OR, NOT, NAND, NOR, sum-of-products, and product-of-sums logical expressions to simplify designs, where a combination of the results of sub-expressions is required. BE activity can be determined by the input signals and it will continue processing in the same symbol cycle as the STEs that drive it. The combinational circuit has an optional synchronized enable, which is triggered by a special signal propagated simultaneously to all Boolean elements. An example of the Boolean element combining sub-expressions with the use of the synchronized enable is shown in Figure 4. In this example the automata reports, at the assertion of the synchronized enable, if the input stream up to that point contained both a lower case ASCII letter or an upper case ASCII letter and an ASCII digit.

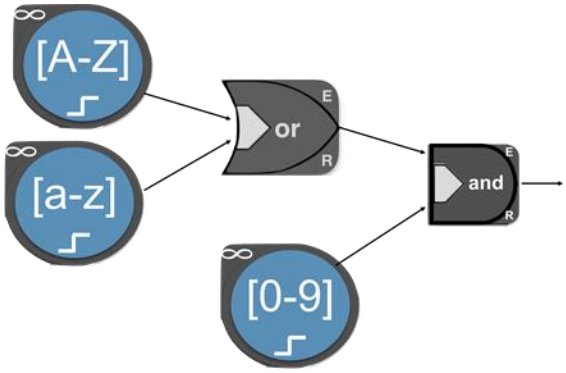

Figure 4. Boolean Elements

2.2. Programming Resources

A single AP chip contains 49152 STEs among which 6144 can report, 768 counter elements, and 2304 Boolean elements arranged hierarchically into rows, blocks, and half-cores as shown in Figure 5. Each half core contains 96 blocks of 256 STEs each, 4 counters and 12 Boolean elements. The physical routing capability, which is provided by the routing matrix, reduces as we move up the hierarchy from rows to half-cores. All the 16 STEs within a single row can be connected to each other and to the counter or Boolean element in the row. Every fourth row contains a counter element and the other three rows contain a Boolean element. However, elements from different rows need to be connected using block routing lines, 24 of which are shared by the 16 rows in each block. The connectivity between SETEs from 96 blocks in a half-core is even more limited. The routing matrix contains programmable switches to control the interconnection between the levels of the hierarchy, buffers, routing lines, and cross-point connections. Programming of different switch buffers controls the transitions between the different signal groups. A given signal can be selectively connected to numerous different inputs of

adjacent levels of the hierarchy. The programming of those signals is done by the application to the desired automata design [13, 14, 16].

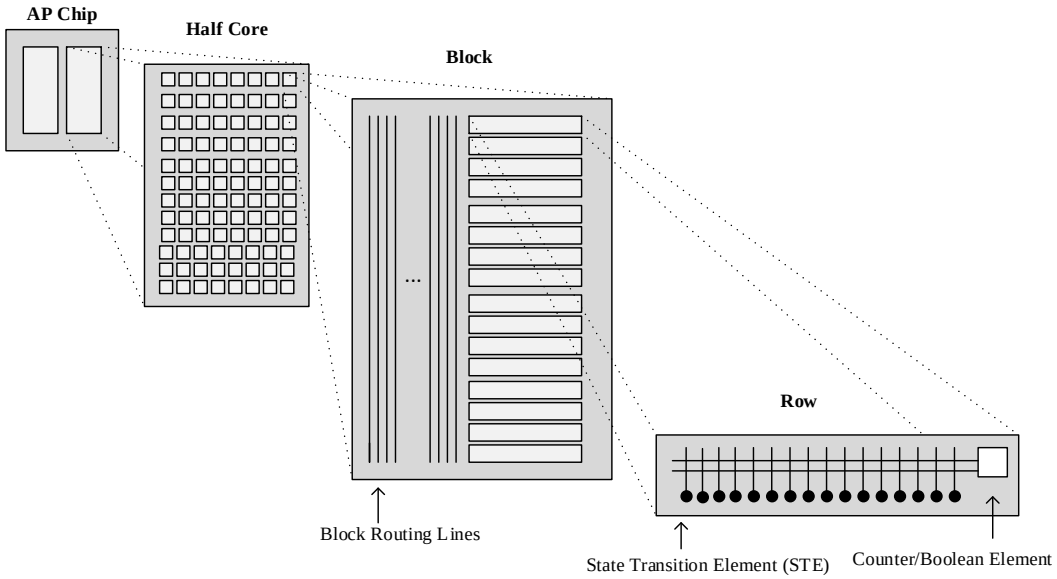

Figure 5. Hierarchical Layout of Processing Elements in the Automata Processor Chip [16]

Micron's current generation AP can run at an input symbol rate of 133MHs, with each chip supporting two half-cores. However, an AP chip does not interface with the host processor directly. Multiple AP chips are arranged into an Automata Processor board (AP board), which is then connected to the host processor. Currently, development AP boards come in three form factors. The smallest AP board contains two AP chips and connects to the host processor over the Universal Serial Bus (USB) interface. The two AP chips are organized into a single rank. The rank contains a high-speed intra-rank bus which allows two separate data flows to be streamed to the two AP chips in parallel, or a common data flow to be broadcast to both the AP chips. The second AP board contains 32 AP chips and is connected to the host processor using a high-speed Peripheral Component Interconnect Express (PCIe) interface. The chips are organized into 4 ranks containing 8 chips each. The third AP board is the largest of the three and contains 48 AP chips organized into 6 ranks. It is also connected to the host processor using a PCIe interface. In total, this AP board contains 2359296 STEs, 36864 counter elements, and 110592 Boolean elements. Those interconnected processing elements are programmed to perform a targeted task or operation in parallel. The CPU initializes the AP chips; send data and fetched results using PCIe transactions [13, 16].

2.3. AP Architecture Limitation

The duty of the routing matrix is to control the exchange of signals to and from the automata elements, following the programmed instructions in the application. The design of the routing matrix is all about a group of elements organized in a complex hierarchical net structure. The ideal theoretical design of the automata implies that every element can be potentially connected to all other elements. But the actual physical silicon implementation limits the routing capacity because of tradeoffs in clock rate, propagation delay, die size, and power consumption. All programmable elements are affected by these capacity limitations caused by the routing matrix. Large fan-in or fan-out automata requirements need duplication of elements. Large automata may affect the efficient mapping of those automata onto the Automata Processor and this will increase the compilation times as well as decrease the efficiency of the functions of the elements in the Automata Processor.

3. Programming Environment

The design of AP is based on the DRAM technology. Micron developed an SDK to facilitate the configuration of the chip with automata designs and to provide run-time control of sets of automata processors. Executing programs on the AP involves two steps: configuration and execution. In the configuration stage, the user-defined automata are programmed into the AP by first performing the compilation and then loading operations. During the execution stage, these automata are executed in parallel on streaming data. If an automaton reaches its accept state, an output is created along with the offset in the stream where the event occurred [13-16]. It is a well-established fact in computability theory that finite automata and regular expressions are equivalent, *i.e.*, any finite automata can be represented as a regular expression and vice-versa. The composition of an automata network can be described using both: 1) Direct implementation of regular expressions in PCRE (Pearl Compatible Regular Expression) syntax, or 2) Automata Network Markup Language (ANML) which is an XML-based language. Additionally, the SDK provides C and Python interfaces to create automata, input stream, parse output and manage computational tasks on the AP board [10].

3.1. Pearl Compatible Regular Expression (PCRE)

A regular expression is a sequence of characters and special symbols that represent a set of exact-match strings. Regular expressions can be directly converted to automata for programming the AP. The PCRE syntax is well known to the programming community and AP has been designed to have a high degree of compatibility with PCRE. Programming the automata processor with PCRE goes through two steps. First building the expression database that contains all the expressions of an automaton. Secondly, compile the expression database. The default compilation process of PCRE-API of Micron Company applies optimizations to minimize STE resource usage within the automata processor. It offers the ability to modify the default compilation settings if needed. Also, it has many ready-made functions that help the programmer in building the required automation. Although the regular expression is common means for defining automata, but their use with the AP also has drawbacks. For the problems accelerated by the AP, regular expression representation is often an exhaustive enumeration of all accepting data sequences, which are difficult to maintain and places an unnecessary burden on the developer [10].

3.2. Automata Network Markup Language (ANML)

AP can be configured through ANML, which is an XML language for describing the composition of automata networks, contains elements that represent automata processing resources [13, 15, 16]. Using ANML, a software programmer can explicitly describe how these automata processing resources are connected together to create an automata network by configuring the elements, configuring the connections, providing input and allowing the automata network to compute. The processing elements in ANML are very similar to the classical state diagram representation. In the state diagram, the states are depicted using circles and transitions between the states are drawn as directed edges. A state is identified using state-labels, which are placed inside the circle. A short incoming arrow shows a start state and the boundary of the circle with double lines shows an accepting state.

The developers of Micron Company do not recommend that designers write an ANML code directly, instead, they recommend constructing automata first with the ANML programming interface. Micron Company created a graphic tool called ANML workbench, where the automata networks can be constructed and exported to XML files. This tool also provides simulator to simulate the matching process for the user generated input stream to check the correctness of the user's design. It is offering a drag and drop environment, where the designer can choose from the available elements (STE, Boolean, counter) and set the wanted specifications and properties for them such as a symbol set for STEs, target count for counter elements, and logical configuration for logical elements. The transitions between the elements can be made by drawing lines between the elements to connect them. Furthermore, designers can make use of a function called macro object in the ANML workbench. The macro object is a block used to encapsulate a particular pattern in or for reusability for the patterns that appear frequently to make their design more convenient.

Although the construction and the modeling of NFA are straightforward and simple using ANML workbench, it can be complex and difficult in the case of large and complex designs. An alternative to a graphical solution is the use of specialized executable programs that uses other compiled languages to generate ANML. The programming environment for the AP may be very simple, coming up with efficient algorithms and automata designs to utilize the power of this processor may require creativity and novelty on behalf of programmers who are trained on traditional processing platforms.

3.3. RAPID Programming

The AP is currently challenging to program using an XML-based language since it requires both the knowledge of automata theory and also the non-standard architecture. This is analogous to assembly language programming on a traditional, von Neumann architecture. Similarly, the use of regular expressions with the AP has its own limitations. To overcome the limitations of the current programming languages for the AP, the authors [17] have proposed a high-level language called RAPID that maintains the performance benefits of AP while providing concise, clear and maintainable and efficient representations of pattern identification algorithms. RAPID program consists of one or more macros and a network, written in a combination of imperative and declarative styles. A macro uses sequential control flow to define an algorithm for matching input data stream. The network consists of a list of macros that are instantiated in parallel, allowing for simultaneous recognition of many patterns in a data stream. Macros and network provide a programming abstraction that maps naturally to both pattern-matching problems and computational model of the automata processor while providing a familiar structure similar to function or procedures. Further, the authors [17] introduce a tessellation technique for configuring the AP, which significantly reduces compile time, increases programmer productivity, and improves maintainability. The evaluation of the RAPID

programs demonstrated that the programs are compact, expressible at a higher level of abstraction and order of magnitude faster than the current solutions. In future, RAPID may become the prime choice for programming the AP with the addition of debugging and testing tools.

4. Automata Processor Applications and Techniques

Since the release of the Automata Processer (AP), it became an active field of research. A number of researchers have developed algorithms and evaluated their effectiveness for a wide range of applications on the AP. The following applications and techniques are success examples of testing the efficiency and strengths of the AP.

4.1. Brill Tagging

Brill Tagging [18, 19] is a classic rule-based POS (part-of-speech) tagging algorithm, which makes an assignment of a tag to input tokens, such as, nouns, verbs, and adjectives *etc*. The algorithm proceeds in two steps: training and tagging. During the training step, it identifies the most frequent tag for all recorded tokens as well as contextual rules for updating the tags. The tagging step is further divided into two stages for tagging new untagged corpora. In the first stage, it assigns the most frequent tag to the new corpora, and in the second stage, it updates the initial tags based on the contextual rules. The Brill Tagging algorithm is one of the most widely used rule-based approaches, however, it is a time-consuming algorithm for both training and tagging. Making it faster is desirable for both to increase the speed and the sophistication of semantic analysis. Zhou *et al*. [20, 21] developed a technique by transforming the rules as regular expressions and by implementing the regular expressions in parallel on AP to reduce the computational time of the second stage of tagging. Experimental results on one of the commonly used POS tag-sets the Brown corpus [19, 20] showed that AP achieves a linear speedup with the number of rules as compared to the state-of-the-art POS-tagging technique on Intel Core i5 machine. The study reveals that the AP may be a promising platform for semantic analysis tasks.

4.2. Entity Resolution

Entity resolution [22] is the process of extracting identical entities in a single database or across multiple databases. It is an important process for many applications such as social networks applications, where the records of a single user could be stored in multiple databases. In order to collect these records, all possible pair of records must be compared together, which lead to computationally expensive O (n^2) execution time. Many techniques have been proposed at the software level for the entity resolution problem to reduce the execution time. One such technique is the domain-independent algorithm [23]; which is based on detecting approximately duplicate records by computing the minimum edit distance to extract pairs of similar records and then apply Union/Find algorithm to keep track of the duplicate records. The sorting algorithm also was used to sort all the records and then check the neighboring records to find duplicate pairs. Other hardware solution for the entity resolution problem includes a generalized pattern matching micro-engine to accelerate FSM-based applications through translating regular expressions to DFA tables, then storing these tables into memory and finally executing the DFA in parallel using vector-instruction interface, so they can reduce the instruction count [24]. In contrast to these approaches, AP is a memory-based architecture, which is capable of implementing the automata directly and efficiently. Bo *et al*. [25] modeled the entity resolution problem as fuzzy name matching problem and proposed an automata design which can efficiently be implemented on AP. Comparison of results with the state-of-the-art algorithm Apache Lucene [26] showed that the AP based solution achieves both higher

performance (434x speedup on average) and better accuracy in finding an exact or fuzzy match. The AP shows a great promise for accelerating entity resolution problem.

4.3. Association Rule Mining

Association rule mining (ARM) [27] is a widely used data mining technique for discovering sets of frequently associated items in large databases. As an example, ARM would try to recognize buying patterns from a database such that "50% of all customers who bought pancake mix and eggs would also buy maple syrup" to be able to better organize the supermarket using the association information. ARM technique has also been widely used in recommendation systems, on-line stores, web usage mining, traffic accident analysis, intrusion detection, market basket analysis and bioinformatics among other applications [27]. Multiple renowned algorithms have been proposed to solve ARM problem because of its key role in the emerging field of data analytics. The two well-known and widely used algorithms are the Apriori algorithm proposed by Agrawal and Srikant [27] and the Equivalence Class Clustering (Éclat) algorithm developed by Zaki *et al*. [28]. Apriori algorithm repeatedly scans the database for combinations referred to in Apriori as itemsets. Apriori is known to be a precise and efficient algorithm to minimize the search space of the candidate itemsets using the property of Downward-Closure. The main functions in Apriori algorithm are to generate new candidate itemsets from previous frequent itemsets and to count the support number of candidate itemsets and then comparing it with the threshold value named minimum support (minsup). The main bottleneck of the Apriori algorithm is the repeated scanning and counting element for support step. Many algorithms and accelerators have been proposed to enhance the performance of the Apriori algorithm and Éclat algorithm using multi-processors and GPUs [29, 30]. However, the parallel algorithms face the same limitations when it comes to dealing with large sets of unstructured or even structured databases.

As we mentioned previously it is challenging to deal with large datasets related with ARM, however, these challenges can be handled with the release of Micron automata processor chip. Wang *et al*. [31] have developed AP algorithm to speedup ARM Apriori algorithm. The AP algorithm consists of two components: matching and counting. The matching components are implemented by an NFA to recognize a given itemset, which also captures the cases of discontinuous patterns of items. The counter components use an on-chip counter element to calculate the frequency of a given itemset. High parallelism was achieved by utilizing a huge amount of pattern matching and counting elements of the AP board. Several performance optimization strategies were proposed and implemented to improve the performance including data-pre-processing. The experimental results show that the proposed AP algorithm for ARM provides a speedup up to 94x as compared with the Apriori single-core CPU implementation and also outperforms one of the fastest multicore and GPU implementation of Éclat algorithm [30].

4.4. Levenshtein Nondeterministic Finite Automata

Approximate string matching is used in many application domains including bioinformatics, text retrieval, spell checking, and search engine applications. Levenshtein non-deterministic finite state automaton (NFA) is a fast (linear time) method of determining approximate string matches without explicitly calculating the edit distance between the input string and the search string pattern [32]. A von Neumann simulation of Levenshtein NFA incurs exponential run-time overhead in the general case, which can be avoided by converting it to DFA, but at the expense of heavy pre-computation and high space overhead. Micron AP allows executing NFAs directly in hardware and does not have the same scalability limitations as von Neumann architecture. Therefore, AP presents an ideal platform to implement Levenshtein NFA. Tracy *et al*. [33] presented a novel technique for directly executing a pipelined Levenshtein NFA on Micron AP. The

mechanism of the Levenshtein's structure yields the NFA machine to be in a Mealy finite state machine form because of its aid to allow multi transitions with various inputs triggering. Mealy-type Finite State Machine is unable to be realized by the AP architectures so it should be converted into a Moore-type Finite State Machine. The authors described a method to convert Mealy FSM to Moore FSM and other performance optimizations [33]. The results show the viability of AP to execute large instances of the Levenshtein's NFA in parallel to be used as the building block for future approximate string matching applications.

4.5. Discovering Motifs in Biological Sequences

Finding patterns in DNA or Protein sequences is of paramount importance to bioinformatics field. Discovering of these patterns called motifs helps in the investigation of diseases in human and medical therapies. Searching motifs faced several difficulties in early years and so many algorithms emerged to get the best motifs search. One of the types of the motifs search called the Planted Motif Search Problem (PMP) is of prime importance to researchers. According to the authors [16, 34, 35], the time required for solving a typical PMP problem instance on 48-cores machine take up to 46.9 hours beside some instances remain unsolved. Roy and Aluru [34, 35] applied the AP to solve the PMP in genomics by developing an efficient algorithm. The key success of the AP for solving such problems is by casting the exact algorithms through multi NFAs. AP chip hardware architecture allows for high-speed enhancement of searches for numerous complex patterns. The results show that AP is capable of efficient solutions for large fuzzy matching NP-Complete problems.

4.6. Data Encryption

Cyber-security has become one of the most important issues in the current era of cyber warfare. The AES (Advanced Encryption Standard) is a symmetric block cipher, which is the key element of many encryption algorithms for providing data security, integrity, and privacy. The Micron automata processor is a new accelerator that is capable of processing multiple input data streams concurrently at a high rate. The author [36] proposed a series of automata designs for S-box, multiplication operation and other key operation performed in AES for implementing it on the Micron automata processor. The compilation and simulation results demonstrate the effectiveness of implementing AES on the new technology. The proposed designs were successfully validated, completed, and simulated on the AP workbench. The AP implementation of AES significantly outperforms Intel CPU core (Inter AES-NI) implementation. The performance results demonstrated a promising future of the AP as an accelerator in the area of cyber security.

4.7. Stochastic Computing

Probabilistic automata (or stochastic automata) are incredibly useful as modeling formalism for finance, manufacturing, communication, and many other disciplines. For example, Markov chains (a kind of probabilistic automata) are used extensively in all fields concerned with simulation. Wadden *et al*. [37] reported non-obvious abilities of the AP by examining the uses of random and stochastic symbol sequences as input. The paper presented the construction of probabilistic automata and showed how they can be used to simulate Markov chains, Brownian motions and stochastic computation on AP. The combination of probabilistic automata and stochastic computation open up a wide range of applications [38, 39, 40], previously not known to be implementable on the AP, with potential for impressive speedup over CPUs and GPUs. The authors also suggest some changes to be made to the AP for more efficient implementation of stochastic computation on AP.

4.8. Track Pattern Recognition

Track pattern recognition is a computationally challenging problem of prime importance in the reconstruction and analysis chains of all High Energy Physics (HEP) experiments. The authors [41] have successfully demonstrated a proof-of-concept for the suitability of the Micron Automata Processor in particle track finding applications. Compared with multi-core CPUs and GPUs, AP implementation was reported to be two times faster. Previously, the applications of AP has been limited to text-based searches. However, the use of AP for track pattern finding application opens up a whole new set of applications including real-time image processing.

5. Conclusion and Future Work

Automata Processor (AP) is a promising computing architecture to accelerate massively parallel operations related to the search and analysis of complex and unstructured data streams pattern identification problems. AP is a novel non-von Neumann processor, which can implement directly multi NFAs concurrently in hardware without considering the space overhead and the need for explicitly programming the NFA. In this paper, we reviewed the automata processor architecture, its programming environments and current applications in diverse fields. This survey should be of particular interest to researchers who are considering the use of the AP in their research study. There is much more to be learned about automata processor to make the best use of it in accelerating pattern identification problems.

The AP is currently challenging to program using an XML-based language since it requires both the knowledge of automata theory and also the non-standard architecture. Automata computing paradigm requires software development tools including debugging and testing tools which are expressible at a higher level of abstraction and utilize the hardware resources efficiently to accelerate the growing and important class of automata-based algorithms. In addition, automata processors have to support a wide range of finite automata models to create potential opportunities for future research [42]. Furthermore, interesting directions in this increasingly important area of automata computing include exploring the processing of multiple symbols per clock period to achieve higher throughput. The combination of probabilistic automata and stochastic computation is another avenue to explore to extend the application domain of automata processor.

References

[1] I. Abaker, T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The Rise of "Big Data" on Cloud Computing: Review and Open Research Issues," Information Systems, Vol. 47, **(2015)**, pp. 98-115.

[2] J. E. Hopcroft, R. Motwani, and J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, 2nd ed., MA, USA: Addison-Wesley, **(2001)**.

[3] M. Gongora-Blandon and M. Vargas-Lombardo, "State of the Art for String Analysis and Pattern Search Using CPU and GPU Based Programming," Journal of Information Security, Vol. 3, No. 4, **(2012)**, pp. 314-318.

[4] Y. Zu, M. Yang, Z. Xu, L.Wang, X. Tian, K. Peng, and Q. Dong, "GPU-Based NFA Implementation for Memory Efficient High Speed Regular Expression Matching," ACM SIGPLAN Notices, Vol. 47, No. 8, **(2012)**, pp. 129-140.

[5] Y.-H. E. Yang and V. K. Prasanna, ''High-Performance and Compact Architecture for Regular Expression Matching on FPGA,'' IEEE Transactions on Computers, Vol. 61, No. 7, **(2012)**, pp. 1013-1025.

[6] Y. Kaneta, S. Yoshizawa, S.-I. Minato, and H. Arimura, ''High-Speed String and Regular Expression Matching on FPGA,''presented at the Asia-Pacific Signal Information Processing Association Annu. Summit Conf., Xi'an, China, **(2011)**.

[7] H. Wang, S. Pu, G. Knezek, and J. Liu, ''Min-Max: A Counter-Based Algorithm for Regular Expression Matching,'' IEEE Transactions on Parallel and Distributed Systems, Vol. 24, No. 1, **(2013)**, pp. 92-103.

[8] V. Kosar and J. Korenek, "Towards Efficient Field Programmable Pattern Matching Array," In Proceedings of the 2015 Euromicro Conference on Digital System Design (DSD), **(2015)**, pp. 1-8.

[9] R. D. Cameron, T. C. Shermer, A. Shriraman, K. S. Herdy, D. Lin, B. R. Hull and M. Lin, "Bitwise Data Parallelism in Regular Expression Matching," In Proceedings of the 23rd International Conference on Parallel Architectures and Compilation (PACT'14), **(2014)**, pp. 139-150.
[10] X. Wang, Techniques for Efficient Regular Expression Matching across Hardware Architectures, Graduate Thesis, University of Missouri-Columbia, **(2014)**.
[11] A. Morad, L. Yavits and R. Ginosar, "GP-SIMD Processing-in-Memory," ACM Transactions on Architecture and Code Optimization, Vol. 11, No. 4, Article 63, **(2015)**, pp. 1-26.
[12] D. P. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, M. Ignatowski, "TOP-PIM: Throughput-Oriented Programmable Processing in Memory," In Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing, **(2014)**, pp. 85-98.
[13] 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, **(2014)**, pp. 3088-3098.
[14] K. Skadron, M. R. Stan, K. Wang, and J. J. Fox, "The Automata Processor: a Powerful New Computing Accelerator," University of Virginia, **(2015)**.
[15] C. R. Sabotta, Advantages and Challenges of Programming the Micron Automata Processor, Graduate Theses and Dissertations, Iowa State University, Ames, Iowa, **(2013)**.
[16] I. Roy, Algorithmic Techniques for the Micron Automata Processors, Ph. D. Dissertation, Georgia Institute of Technology, **(2015)**.
[17] K. Angstadt, W. Weimer and K. Skadron, "RAPID Programming of Pattern-Recognition Processor," In Proceedings of the 21st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16), Atlanta, Georgia, **(2016)**.
[18] A. Voutilainen, "Part-of-Speech Tagging," The Oxford Handbook of Computational Linguistics, **(2003)**, pp. 219-232.
[19] S. Mohammad, and T. Pedersen, "Guaranteed Pre-Tagging for the Brill Tagger," Proceedings of the 4th International Conference on Computational Linguistics and Intelligent Text Processing, **(2003)**, pp. 148-157.
[20] K. Zhou, J. J. Fox, K. Wang, D. E. Brown and K. Skadron, "Brill Tagging using the Micron Automata Processor," Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing, **(2015)**, pp. 236-239.
[21] K. Zhou, J. Wadden, J. J. Fox, K. Wang, D. E. Brown and K. Skadron, "Regular Expression Acceleration on the Micron Automata Processor: Brill Tagging as a Case Study," Proceedings of the 2015 IEEE International Conference on Big Data, **(2015)**, pp. 335-360.
[22] H. Wang, Innovative Techniques and Applications of Entity Resolution, IGI Global Publisher, **(2014)**.
[23] A. E. Monge and C. P. Elkan, "An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records," In Proceedings of the SIGMOD 1997 Workshop on Data Mining and Knowledge Discovery, **(1997)**.
[24] Y. Fang, R. Rasool, D. Vasudevan, and A. A. Chien, "Generalized Pattern Matching Micro-Engine," Fourth Workshop on Architectures and Systems for Big Data, Minneapolis, MN, USA, **(2014)**.
[25] C. Bo, K. Wang, J. J. Fox, and K. Skadron. "Entity Resolution Acceleration using Automata Processor," In Proceedings of the Workshop on Architectures and Systems for Big Data (ASBD), Portland, Oregon, USA, **(2015)**.
[26] Apache Lucene. http://lucene.apache.org.
[27] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," In Proceedings of the 20th International Conference on Very Large Data Bases, **(1994)**, pp. 487-499.
[28] M. J. Zaki, "Scalable Algorithms for Association Mining," IEEE Transactions on Knowledge and Data Engineering, Vol. 12, No. 3, **(2000)**, pp. 372–390.
[29] M. J. Zaki, M. Ogihara, S. Parthasarathy, and W. Li, "Parallel Data Mining for Association Rules on Shared-Memory Multi-Processors," In Proceedings of the 1996 ACM/IEEE Conference on Supercomputing, **(1996)**, pp. 1-18.
[30] F. Zhang, Y. Zhang, and J. D. Bakos, "Accelerating Frequent Itemset Mining on Graphics Processing Units," Journal of Supercomputing, Vol. 66, No. 1, **(2013)**, pp. 94-117.
[31] K. Wang, M. Stan, and K. Skadron, "Association Rule Mining with the Micron Automata Processor," In Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium (IPDPS), **(2015)**, pp. 689-699.
[32] 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," In Proceedings of the Workshop on Architectures and Systems for Big Data (ASBD), Portland, Oregon, USA, **(2015)**.
[33] K. U. Schulz and S. Mihov, "Fast String Correction with Levenshtein Automata," International Journal on Document Analysis and Recognition, Vol. 5, No. 1, **(2002)**, pp. 67-85.
[34] I. Roy and S. Aluru, "Finding Motifs in Biological Sequences using the Micron Automata Processor," In Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium, **(2014)**, pp. 415-424.

[35] I. Roy and S. Aluru, "Discovering Motifs in Biological Sequences using the Micron Automata Processor," IEEE/ACM Transactions on Computational Biology and Bioinformatics, **(2015)**. (DOI:10.1109/TCBB.2015.2430313)
[36] A. Kongmunvattana, "Automata Design for Data Encryption with AES using the Micron Automata Processor," IJCSNS International Journal of Computer Science and Network Security, Vol. 15, No. 7, **(2015)**, pp. 1-5.
[37] J. Wadden, K. Wang, M. Stan and K. Skadron, "Uses of Random and Stochastic Input on Micron's Automata Processor," University of Virginia, **(2015)**.
[38] A. Alaghi and J. P. Hayes, "Survey of Stochastic Computing," ACM Transactions on Embedded Computing Systems, Vol. 12, Issue 2s, Article No. 92, **(2013)**, pp. 1-19.
[39] P. Li, D. J. Lilja, Q. Weikang, K. Bazargan, and M. D. Riedel, "Computation on Stochastic Bit Streams Digital Image Processing Case Studies," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 22, No. 3, **(2014)**, pp. 449-462.
[40] V. Canals, A. Morro, A. Oliver, M. L. Alomar, and J. L. Rossello, "A New Stochastic Computing Methodology for Efficient Neural Network Implementation," IEEE Transactions on Neural Networks and Learning Systems, in press, **(2015)**.
[41] M. H. L. S. Wang, G. Cancelo, C. Green, D. Guo, K. Wang, and T. Zmuda, "Fast Track Pattern Recognition in High Energy Physics Experiments with the Automata Processor," arXiv:1602.08524v1 [physics.ins-det], **(2016)**.
[42] 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 48th International Symposium on Microarchitecture (MICRO-48), **(2015)**, pp. 533-545.

