Entity Resolution Acceleration using the Automata Processor

Chunkun Bo[1], Ke Wang[1], Jeffrey J. Fox[2], Kevin Skadron[1]
[1]*Department of Computer Science,* [2]*Department of Material Science*
University of Virginia
Charlottesville, USA
Email: chunkun, kewang, jjf5x, skadron@virginia.edu

***Abstract*—Entity Resolution (ER), the process of finding identical entities across different databases, is critical to many information-integration applications. As sizes of databases explode in the big-data era, it becomes computationally expensive to recognize identical entities among all records with variations allowed across multiple databases. Profiling results show that approximate matching is the primary bottleneck. The Automata Processor (AP), an efficient and scalable semiconductor architecture for parallel automata processing, provides a new opportunity for hardware acceleration for ER. We propose an AP-accelerated ER solution, which accelerates the performance bottleneck of fuzzy matching for similar but potentially inexactly-matched names, and use several different real-world applications to illustrate its effectiveness. We compared the proposed method with several conventional methods and achieved both promising speedups and better accuracy (more correct pairs and less generalized merge distance cost) for different datasets.**

***Keywords*-Automata Processor; Entity Resolution; Acceleration;**

I. Introduction

Entity Resolution (ER), also known as Record Linkage or Purging/Merging problems, refers to finding records that store the same entity within a single database or across different databases [1]. ER is an important kernel of many information-integration applications. For example, Social Networks and Archival Context (SNAC) collects records from databases all over the world to provide an integrated platform for searching historical collections [2]. In such applications, the records of the same person may be stored with slight differences, because documents come from different sources, with different naming conventions, transliteration conventions, etc. SNAC needs to find the records referring to the same entity despite different representations and merge these records. The intuitive method of solving ER is to compare all possible pair records and check whether a pair represents the same entity.

Determining whether two records represent the same entity is usually computationally expensive. For example, the time complexity of the intuitive method is $O(N^2)$, where N is the number of records. Prior work has proposed different algorithms and computation models to improve the performance [3] [4] [5]. However, the performance is still unsatisfying and the average time used by record comparison is much longer than the cost of simple string comparison [6].

The Automata Processor (AP) is an efficient and scalable semiconductor architecture for parallel automata processing [7] introduced by Micron. It is a hardware implementation of non-deterministic finite automata (NFA) and is capable of matching a large number of complex patterns in parallel. The AP has been used in different fields such as association rule mining [8], bioinformatics [9], string kernel testing [10], natural language processing [11], etc. Such applications need inexact matching and high throughput, just as does ER. Therefore, we propose a hardware acceleration solution to ER using the AP and focus on string-based ER. To illustrate how the AP approach works, we present a framework and evaluate the suitability using several real-world ER problems in different applications.

In summary, we make the following contributions:

1. We propose a novel AP-based hardware acceleration framework to solve string-based Entity Resolution.

2. We present several automata designs for string-based ER, e.g. fuzzy name matching. We apply the proposed approach in SNAC to illustrate the effectiveness of the approach, and generalize it with small modifications for other string-based ER problems.

3. We compare the prototype of the AP approach with several conventional approaches (Apache Lucene, sorting, hashing, and suffix-tree methods) to evaluate the suitability of the proposed method. Results show both higher performance and better resolution accuracy using various datasets from different applications.

II. Related Work

Many methods have been proposed to solve ER. One is a domain-independent algorithm [3]. This paper proposed first computing the minimum edit-distance to recognize possible duplicate records, and then using a union/find algorithm to keep track of duplicate records incrementally. This proposed method achieves around 5x speedup compared with previous methods. Another method sorts the records and checks whether the neighboring records are the same [1]. For approximate duplicates, these researchers define a window size and a threshold of similarity, so that they can find similar records to satisfy application requirements. Apache Lucene is a high-performance search engine and uses a similar method [12]. The difference is that it calculates the score of a

This paper appears in Proceedings of BigData16. This is the authors' final manuscript. The authoritative version can be found in the digital library.

document based on the query, and sorts documents instead of every individual record. As databases become much larger, some researchers suggested dividing the original database into small blocks based on prior knowledge and processing smaller blocks, but the method is not always feasible because not all databases are easily divided [4]. However, we are unaware of any implementations of these algorithms using accelerators. Furthermore, we hypothesize that the AP's massive parallelism can achieve much higher performance than these methods. Therefore, we propose an AP-based approach for the Hamming distance-based method.

Some novel architectures have been proposed to scale ER to multiple nodes. For example, Kolb et. al. propose Dedoop, a cloud-based infrastructure to accelerate ER [5]. They gain 80x speedup by using 100 Amazon EC2 nodes. This paper only considers performance on a single node, but the AP would benefit cluster and cloud infrastructures as well.

III. Automata Processor

The Automata Processor is an efficient and scalable semiconductor architecture for parallel automata processing [7]. It uses a non-Von-Neumann reconfigurable spatial architecture, which directly implements NFA in hardware, to match complex regular expressions. The AP can also match other types of automata that cannot be conveniently expressed as regular expression, by describing the NFA directly. The ability to efficiently implement regular expressions or NFA processing makes the AP well-suited for inexact pattern-matching problems such as ER. Use of NFAs avoids the exponential growth in automata size that can occur with deterministic FAs (DFAs), thus achieving high density for the number of automata structures that fit on the AP.

A. AP Functional Elements

The AP consists of three functional elements: STEs, Counters, and Boolean elements [7]. Each STE can be configured to match a set of any 8-bit symbols and up to 256 different characters can be stored in one STE. The STE activates a set of successor STEs connected to it when the symbols stored in it match the input symbol. The 12-bit counter will trigger an output event when the accumulated value reaches the pre-defined threshold. The Boolean elements can perform classic logical functions. The AP allows all STEs on the board to inspect the next input symbol in parallel, and it is able to process a new input symbol every clock cycle.

B. Speed and Capacity

The current generation AP chip (D480) is built on 50nm DRAM technology running at an input symbol rate of 133MHz. The D480 chip has two half-cores and each half-core has 96 blocks. Each block has 256 STEs, 4 counters, and 12 Boolean elements. In total, one D480 chip has 49,152 STEs, 2,304 Boolean elements, and 768 counter elements. Each AP board can have up to 32 AP chips, providing more than 1.5 million STEs.

C. Programming and Reconfiguration

The AP workbench is a graphical user interface tool for quick automata design and debugging. A "macro" is a container of automata for encapsulating a given functionality. The AP SDK also provides C and Python interfaces to build automata, create input streams, parse the output, and manage computational tasks. Furthermore, the symbols that an STE matches can be reconfigured quickly. The replacement time is around 0.24 milliseconds for one block. This feature is helpful when one needs to modify the symbols stored in the AP board without changing the automata structure [8], e.g. for multi-pass algorithms.

IV. Design Details using the AP

In this section, we present how to use the AP to solve string-based ER problems, using the Name Matching problem in SNAC as an example. At the end, we discuss how to generalize the AP approach to other ER problems.

A. Real-world ER Problems

In ER, *identity attributes* distinguish each entity from one another. *String-based* ER means that the *identity attributes* are strings. In this paper, we focus on solving real-world string-based ER problems.

When building the SNAC platform, the same person's name may not always be consistent from one record to the next because of typos, mis-spellings, different abbreviation, etc. These differences lead to three major problems. 1) One may miss some correct results when querying a particular record; 2) multiple entries for one entity waste storage space; 3) the duplicated items slow down the search speed. Therefore, SNAC needs to identify and combine potentially similar records, which is a typical ER problem. Similar problems also exist in DBLP, a website for browsing Computer Science bibliographic information [13]. As DBLP adds records to its database, the contents of the same record may have different representations.

We also present how to identify restaurant records from Fodor's and Zagat's restaurant guides using the AP.

B. Workflow

The general workflow of using the AP to solve string-based ER problems is shown in Figure 1. The CPU first reads the database and extracts the fields of interest from the original database in a pre-processing step, because the database may contain some other information. For example, for the ER problem in SNAC and DBLP, the fields of interest pertain to people's names. We store these fields of interest on the AP board. The input strings are then streamed into the AP and the AP compares the stored contents with the input. If the AP finds a match, it reports back to the CPU. Each record is assigned a number before being streamed into the AP. Based on the reporting STE ID and the offset of the reporting time, the CPU can tell which record finds a match and proceed to combine these

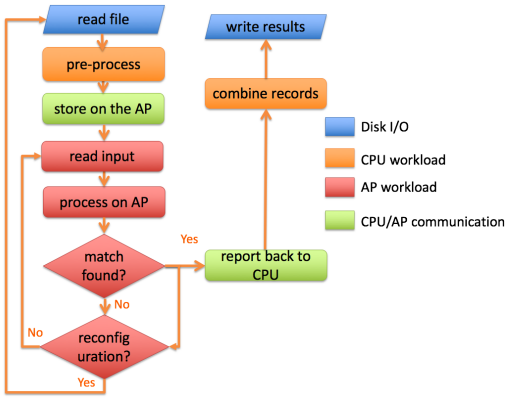

Figure 1. General workflow of the AP approach.

records. A reconfiguration phase is needed if the record number exceeds the capacity of the AP. Intuitively, to config the AP for the next batch of names, one can compile new automata structures for the records which have not been processed yet, but this usually takes a long time due to the high cost of routing for these new structures. Instead, we develop canonical matching macros, which new record values can be loaded without reconfiguring the automata structure; the data are then re-streamed. This approach introduces the cost of replacing symbols (milliseconds), but it is usually faster than compiling new structures (minutes). In the following sections, one will see several design choices to take advantage of the fast symbol replacement.

C. Extracting Name Formats

Intuitively, one can build an automaton for every single record. However, this only works well if the database is small and all records can be stored on one AP board. For large databases, to exploit the fast symbol replacement, we focus on designing a more general automata structure that can be shared by different records [8].

Extracting records formats is a preliminary step for most string-based ER solutions using symbol replacement. We start by describing the approach for solving the Name Matching problem in SNAC. One name is usually composed of several sub-names, like family name, middle name, and first name. We only use family name and first name for SNAC, because middle names are both less common and important for correct resolution in SNAC. Family names and first names are sufficient to evaluate the suitability of the proposed approach.

We choose a subset from the whole database randomly as a basis to extract a representative set of formats for family name and first name. Table I shows common variants of family names and first names. However, not all names can be represented by these formats (fewer than 1%). In this case, we treat it as a failure (no match found). Refinement of these rare cases is left for future work.

D. Automata for Family/First Name

After extracting name formats, we show how to design automata for these formats. The designs are also important

Family Name Formats	First Name Formats
Abc (basic)	Bcd (basic)
Abc Bcd	Bcd X.
Abc Bcd Cde	Bcd Cde
Abc II	Bcd X. (Bcd Xyz)
	B. X.
	B. X. (Bcd Xyz)
	B. X. (Bcd X.)
	Bcd Cde (Xyz)
	Bc. Xyz (Bcde Xyz)
	Bcd, Cde
	Bc
	Bcd O. X. (Bcd Opq Xyz)
	Bcd (Bcd X.)
	Bcd Cde Def Efg

Table I
NAME FORMATS IN SNAC.

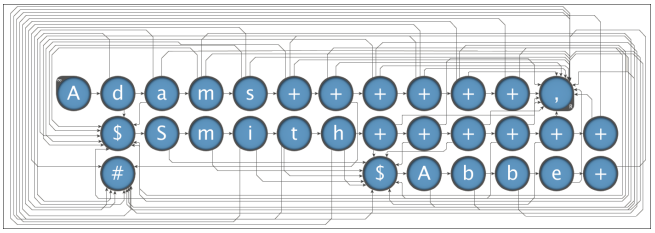

Figure 2. Exact-matching automata design for family name (exact match for "*Adams Smith Abbe*").

for generalizing the AP approach in other ER problems, because they share similar design ideas and techniques.

Figure 2 shows the exact-matching automaton for family names. Though exact-matching automata cannot recognize the same entities with different representations, it is important to understand how the following fuzzy automata work. The three rows correlate with *Abc, Bcd, Cde* in Table I. The first few STEs in each row store the characters in the name to be matched. The subsequent '+' signs are used to pad the remaining positions, so that family names with different lengths can share the same structure. The '$' and the '#' represent spaces and Roman numerals in the database. The ',' STE is configured as a reporting STE. When this STE is activated, it will report. The lengths can be modified according to different dataset characteristics. In this paper, all the four family name formats in SNAC share this structure, although it may consume more STEs than using different automata for different names. Again, this is to utilize the feature of the fast symbol replacement.

This design may lead to some false positives because it aims to support arbitrary string lengths; all of the STEs after the second STE in a given row are connected to the reporting STE. For example, if the automaton in Figure 2 reads *Ada*, it reports a match; but *Ada* is not the name we want. False positives are typically acceptable, and we still need to check the first name. The chance that we get a false positive for both family name and first name is small.

However, the exact-matching cannot fully solve the ER problem and we need to execute 'fuzzy' matching. A fuzzy macro in this paper refers to an automaton that can recognize

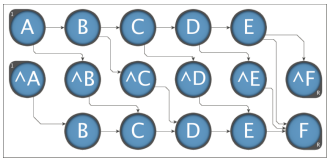

Figure 3. Structure of fuzzy macro calculating Hamming distance. It matches sequence *ABCDEF* with Hamming distance = 0, 1.

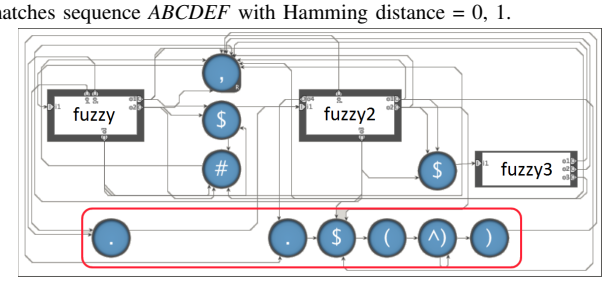

Figure 4. Fuzzy-matching automata design for first name (fuzzy macros allow Hamming distance = 0, 1).

a string with variances. One fuzzy macro example is shown in Figure 3. It matches sequence the *ABCDEF* and reports when the Hamming distance is ≤ 1. This structure is also used in [9]. Column i corresponds to the ith symbol in the sequence. The STEs in odd rows activate on symbols in the target name and the ones in even rows activate when there are mismatches. The Hamming distance can be extended up to k with more $(2k + 1)$ rows . All macros in this paper adopt this structure, but with different sequence lengths. For example, in Figure 4, the name length of macro *fuzzy* and *fuzzy2* is 11 while the length of macro *fuzzy3* is 5. Furthermore, macro structures are not limited to Hamming distance. One can have macro designs for other distances, like general edit distance in [14].

With these fuzzy macros, we can find the same names with variances (Figure 4). The first three rows are used to match the three corresponding parts in family name formats, and we use fuzzy macros in each row to recognize variances of names. The design for the first name (Figure 4) is similar to family name, but we need several extra STEs (last row in red rectangle) to process the '.' and parenthesis, which do not exist in family name formats.

The design may produce false negatives if the AP stores a shorter form first. For example, if AP stores '*J.*' first, it will not report a match when it reads '*Janet*', the full form of '*J.*'. This problem causes most of the inaccuracy in our ER outcomes (Section V-C). In this paper, we consider '.' symbol as a character within a string. However, abbreviations such as '*J.*' often are meant to indicate *J* followed by 0 or more of any character. One possible solution is to use an STE accepting any character when reading '.', but this is left for future work.

E. Hybrid Version

With these automata designs, we roughly evaluated the approach and found that the STE capacity is the bottleneck. To reduce STE consumption, we use a hybrid version (Figure 5) of automata for family name and first name in Figure 4. We

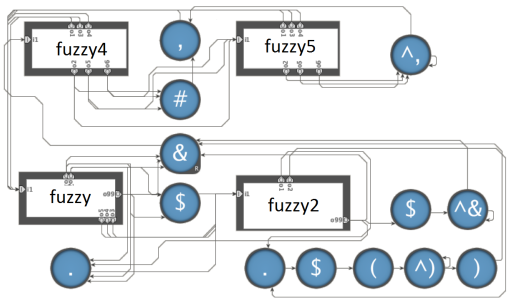

Figure 5. Automata design for the whole name in SNAC ('&' is the delimiter of different names).

recognize the names using one single automaton in order to save STEs. The hypothesis is that if the family name is a match, one can compare fewer characters within the first name. The hybrid technique also allows us to use two fuzzy macros instead of three for family name and first name, because it is unlikely that one finds a wrong match for both sub-names. An STE pointing to itself is used to accept all the remaining characters, further reducing STEs consumed. This self-pointing technique is also used when generalizing the AP approach for other ER problems. With all these techniques, we reduce the STEs consumed from 174 to 99 for one name.

F. Generalizing to Other String-based ER Problems

In this section, we discuss how to generalize the AP approach to other string-based ER problems. As shown in the above discussion, we can build macros that allow different degrees of fuzziness. For example, we can extend the macro in Figure 3 to support different string lengths or different Hamming distances. We can also build macro structures for other distances, such as edit distance [14]. Users can generalize the AP approach with these different automata designs to solve their specific ER problems.

First, we show how to generalize the AP approach to solve the ER problem in DBLP. The workflow is the same as in Section IV-B. Figure 6 shows the automaton design for recognizing similar names in DBLP and middle name is used because most names in DBLP have a middle name. Macro *fuzzy* and *fuzzy1* are used to match first name and family name. These two macros adopt the structure in Figure 3 with different lengths (10). The middle part is used to recognize middle name, which is mostly the abbreviation of the full-length middle name. We use three STEs to store characters in middle name. If the middle name is longer, we ignore the remaining part; if the middle name is shorter, similar to what we have discussed in Section IV-D, we use '+' to pad the position. The fourth self-pointing STE is used to accept all the characters before the 'space' character. The automaton is similar to the one in Figure 5 for SNAC, and they share the same macro structures.

Secondly, we show how to identify the same restaurant from Zagat's and Fodor's restaurant guides. The workflow is still the same and the automata design is shown in

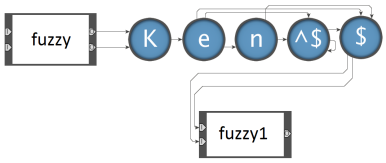

Figure 6. Automata design for DBLP ('$' is the delimiter of sub names).

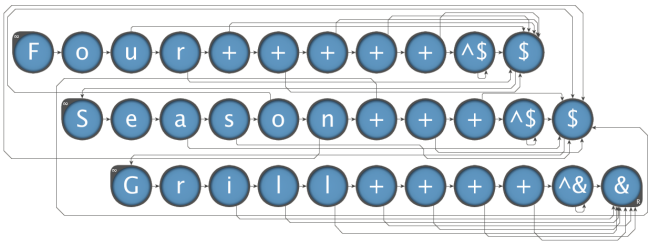

Figure 7. Automata design for identifying the same restaurant ('$' is the delimiter of different parts inside a restaurant name and '&' is the delimiter between restaurant names).

Figure 7. We first work on the record formats, finding that most of the variances come from different abbreviations of the same word, like 'deli' vs. 'delicatessens'. This feature makes the automata look more like the exact match design in Figure 2 but it can recognize the same word with different lengths. Most of the restaurants' names have fewer than three parts, so we only use three rows to represent different parts inside one name. If the name has more than three parts, we only consider the first three; otherwise, we fill the automaton using the latter rows first. For example, for restaurant 'Carneigie Deli', we store 'Carneigie' in the second row and 'Deli' in the third row. As for the first row, we use '+' as a placeholder as in Section IV-D. The second-to-last STE in each row activates itself until it reads the 'space' symbol as discussed in Section IV-E.

These two examples show how we generalize the AP approach for SNAC in other string-based ER problems. They all use the same workflow in Section IV-B. Even though we need to modify the automata designs to solve the specific problem, the structures are similar and we can re-use many design ideas and techniques, such as macro structures, self-pointing STEs, and using '+' as placeholders.

However, this is still not a universal method. There are some applications for which the proposed method is not suitable. The AP approach did not work well when we tried to resolve consumer-electronics products from online shopping websites. This is because the various representations of products are not due to different spellings; instead, they usually have semantic meanings of words or abbreviations or different descriptions, such as 'PlayStation4' vs. 'PS4' or 'black headphone' vs. 'headphone in black'. In such situations, it is difficult for the AP approach to identify these records, because too many variations are required, and a dictionary of all possible relationships is needed.

V. Evaluation

A. Experiment Setup

To evaluate the suitability of the prototype of the AP approach, we compare both execution time and resolution accuracy of the AP approach with other conventional methods. The experiments are executed on a server with AMD Opteron 4386 Cores (3100MHz). We use an AP simulator to derive the execution time for the AP approach until the real hardware is available. The data used in the following experiments are sampled from different databases, including SNAC [2], DBLP [15], Fodor's and Zagat[16]. We use a random selector to select records from these databases.

B. Performance

We first compare the performance of the AP approach with conventional methods, including Apache Lucene, a sorting-based method as suggested in [1], a suffix-tree-based method, and a hashing-based method. Apache Lucene is a widely used searching library and supports advanced query types, such as proximity queries, which enables us to execute fuzzy matching [12]. The sorting-based method first sorts the names and then compares the Hamming distance of neighboring names. The suffix-tree-based method builds a suffix tree for the names and searches names against the suffix tree. The hashing-based method builds a hash table for the names and searches the exact names inside the table. We only execute exact matching with these two methods in this paper. There are some methods which can execute fuzzy matching using suffix tree and hashing-table, but it takes much more time than exact matching.

The matching time is used as the primary metric to evaluate these methods. Because using Apache Lucene involves some other overhead like building indexes for databases, we only collect the time of the search function that executes matching operations. We evaluate these methods with both small datasets and large datasets sampled from SNAC. The results for small datasets are shown in Figure 8. When the database is smaller than 10,000, the suffix-tree-based method is the least effective, yet the matching time increases slower than Apache Lucene as the database size increases. The sorting-based method works as well as the hashing-based method for these datasets. But the sorting-based method achieves better result quality, and the details will be discussed in Section V-C. The AP approach runs faster than all the other four methods for these datasets. The matching time of the AP approach increases almost linearly as the database increases. The slope of the AP approach is nearly flat because the AP can process a new input character every clock cycle. However, the AP can only hold 14,000[1] names at a time; if a dataset is larger than 14,000, a symbol replacement operation is needed to load the next 14,000

[1]The AP capacity in the number of names or other records is a function of the expected record size and complexity (e.g. Hamming distance) of matching.

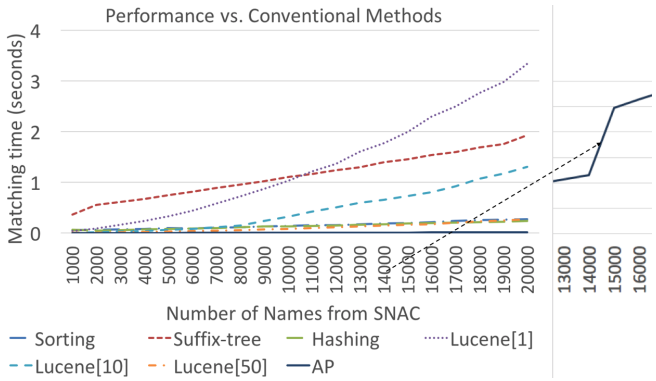

Figure 8. Performance vs. conventional methods for small SNAC databases. (X axis represents the number of names, ranging from 1,000 names to 20,000 names.)

names. This shows up in Figure 8 as steps in the AP curve. We also compare the AP approach with Apache Lucene with its multiple search support. Apache Lucene supports simultaneously searching multiple records (90 at most), but this does not always work for all datasets, and searching 50 names is the largest number that works for all datasets. The average speedup is calculated over all samples in a given dataset. For example, samples of different numbers of names from SNAC (1,000 to 20,000 names in Figure 8) are used to calculate the average speedups against conventional methods (in Table II). Though searching multiple names helps to reduce the matching time, the AP approach still achieves 20.3x speedup compared with searching 50 names simultaneously and up to 373x speedup compared with the suffix-tree-based method.

	Speedup
Sorting	47.9
Suffix-tree	373
Hashing	45.8
Lucene[1]	248
Lucene[10]	75.7
Lucene[50]	20.3

Table II
AVERAGE SPEEDUPS FOR SMALL SNAC DATABASES.

To evaluate how the AP approach works for large databases, we then work on larger datasets (from 14,000 names to 140,000 names) from SNAC. Figure 9 shows that even when the number of names exceeds the capacity and several reconfigurations are needed, the AP approach still runs at least 8.5x faster than other methods. On average, the proposed method achieves 17x, 33.4x, and 16.9x speedup compared with the sorting-based, suffix-tree-based, and hashing-based methods.

Furthermore, we evaluate the AP approach for DBLP. We can store 45,000 names with the design in Figure 6 on one AP board. We collect the matching time for 1 million to 10 million names, since DBLP has a much larger database. Because the number is larger than the capacity of the current board, we need to replace the symbols and re-stream the input. Figure 10 shows that even for much

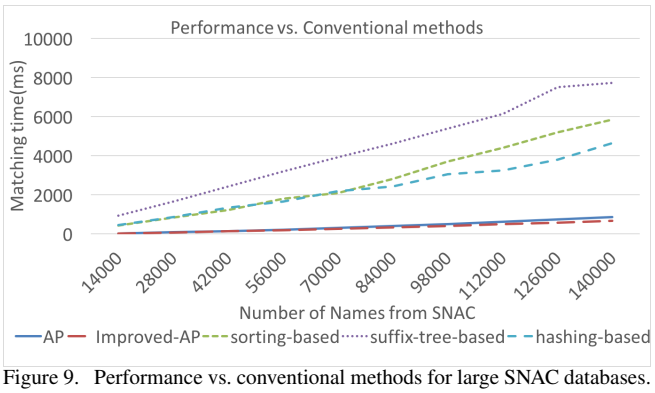

Figure 9. Performance vs. conventional methods for large SNAC databases. (X axis represents the number of names, ranging from 14,000 names to 140,000 names. Y axis represents the matching time.)

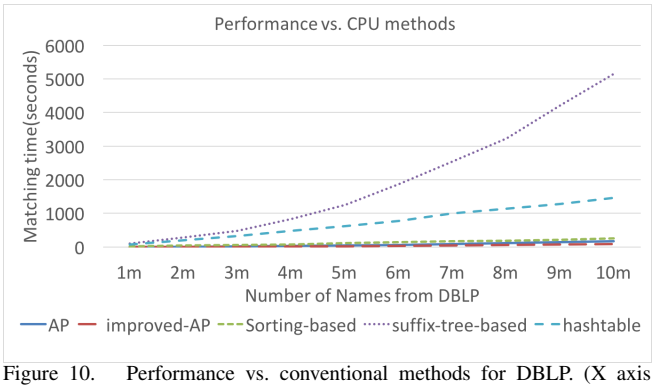

Figure 10. Performance vs. conventional methods for DBLP. (X axis represents the database size, ranging from 1 million names to 10 million names. Y axis represents the matching time.)

larger datasets, the AP approach still works the best among all the methods. On average, it achieves 3x, 29.7x, and 15x speedup against the sorting-based, the suffix-tree-based, and the hashing-based methods. The speedups are not as high as small datasets when compared with the sorting-based method in SNAC, but we achieve much better resolution accuracy (Section V-C). To achieve similar accuracy as the AP approach does, conventional methods need longer time. For example, we can improve the accuracy of the sorting-based method by increasing neighboring group size. However, the time consumed increases linearly as the size increases. We will discuss how to further improve the AP approach performance for large databases in Section V-D.

In general, the prototype of the AP approach achieves promising speedups compared with conventional methods for both small and large datasets, even when many reconfigurations are needed. The speedups come from the massive parallelism when comparing the records using the AP. We can store a large number of records (14,000 for SNAC and 45,000 for DBLP) on one AP board and compare these records simultaneously.

C. Resolution Accuracy

Results quality is also important to study the suitability of the approach; therefore, we use two different metrics (correct pair numbers and generalized merge distance(GMD)) to evaluate the accuracy of the AP approach.

	Correct #	Pct	Merge	Split	Total
Apache Lucene	262	80.6%	51	3	54
Sorting	233	71.7%	63	0	63
Hashing	213	65.6%	72	0	72
Suffix-tree	213	65.6%	72	0	72
AP	292	89.8%	30	1	31
Manual	325	100%	0	0	0

Table III
RESOLUTION ACCURACY RESULTS FOR SNAC.

The first metric is the number of correct pairs (Table III). If there are more than two records in one group, every two records inside the group are counted as one correct pair. The AP approach finds 9.2%, 18.1%, 24.2%, and 24.2% more correct pairs than Apache Lucene, sorting-based, hashing-based and suffix-tree-based methods respectively.

The second metric is GMD [17], which is based on the elementary operations of merging and splitting the records group to correct results. We use a simple version of GMD, where the costs of merging and splitting are equivalent (Table III). The AP approach method only needs 31 operations, while other methods need at least 54. The AP approach needs 50% fewer operations compared with the best conventional method. We observe that most GMD of the AP approach comes from merging operations, which implies that the AP approach may miss some names that should be grouped together instead of recognizing wrong names. To further reduce GMD cost, we can design more complex fuzzy macros in order to identify more similar names. But this may consume more STEs.

We then evaluate the AP approach (Table IV) using a subset from DBLP provided by [15]. The AP approach finds 17% more correct pairs and uses 66% fewer GMD operations than the best conventional method.

	Correct Pair #	Percentage	GMD
Correct	675	100	0
AP	615	91.4	62
sorting	502	74.4	183
suffix-tree	484	71.7	212
hashing-table	484	71.7	212

Table IV
RESOLUTION ACCURACY RESULTS FOR DBLP.

Lastly, we evaluate the design for identifying the same restaurant from Fodors and Zagat. The dataset is achieved from [16] and the matching results are provided. 112 matched tuples should be found and the AP approach finds 105 (93.75%) correct results, while the other three methods only find around 90 (80.36%) correct pairs.

For all three datasets, the AP approach achieves better results quality. This is because the differences in these datasets are usually caused by typos, mis-spellings or different abbreviations, and the fuzzy macros we present in the previous section can recognize these differences and identify the same entities yet with small differences.

D. Improved AP Approach

When the name number is larger than the AP capacity, we re-stream the whole database. However, we do not need to re-stream all records after reconfigurations. For example, for the ER problem in SNAC, 14,000 names are compared in each round and these 14,000 names do not need to be streamed in the next round since they have already been compared. We improve the algorithm by deleting unnecessary processed names, thus reducing the total number of comparisons on the AP. Figure 11 shows that the speedups using the improved algorithm increase from 1.3x to 1.8x compared to the simple AP approach. For the ER problem in DBLP, we also implement the improved algorithm and the results are shown in Figure 10 (solid line) and it works 2x faster than the streaming the whole database.

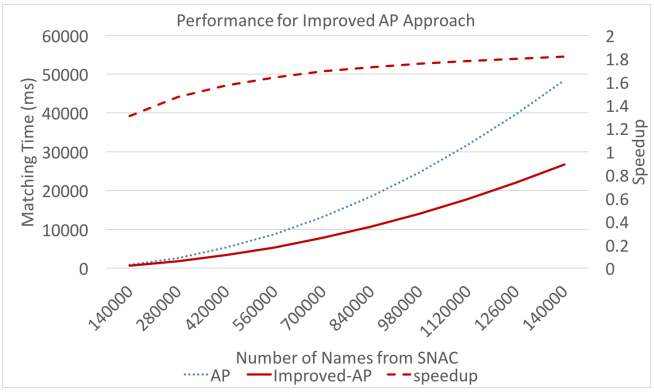

Figure 11. Performance for improved AP approach. (X axis represents the database size, ranging from 14,000 names to 140,000 names. Y axis on the left represents the matching time and Y axis on the right represents speedup against original algorithm.)

E. Scalability

For the current generation of the AP, we can compare 14,000 names for SNAC and 45,000 names for DBLP in one pass. The number may not be large enough for some applications, where databases are much larger, like DBLP. There are several possible ways to address this problem.

The STE capacity is the current bottleneck of the AP approach. Therefore, if we can increase the STE capacity, we can store more records and reduce the number of reconfigurations, thus improving the matching time. In Figure 12, we estimate how the matching time varies when STE number increases using the datasets from DBLP. Results show that the speedup increases almost linearly as STE number increases. We achieve 1.97x, 4.91x, and 9.56x speedups if we can have 2x, 5x, and 10x more STEs on one AP board.

The method we use for larger datasets introduces the cost of replacing the symbols and re-streaming the input. We estimate the performance if we can reduce the symbol replacement time or increase the input rate. For relatively small datasets, reducing symbol replacement time helps more to reduce the matching time than reducing re-streaming time, because symbol replacement time takes most of the matching time. Figure 13 shows how the matching time in SNAC varies when reducing the symbol replacement time. For the ER problem in SNAC, when the number of names is fewer than 230,000, the symbol replacement consumes more

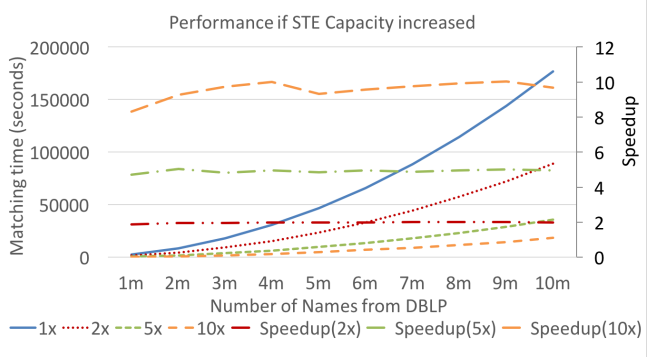

Figure 12. Performance if STE capacity increased. (X axis represents the database size, ranging from 1 million names to 10 million names. Y axis on the left represents the matching time and Y axis on the right represents speedup against original performance.)

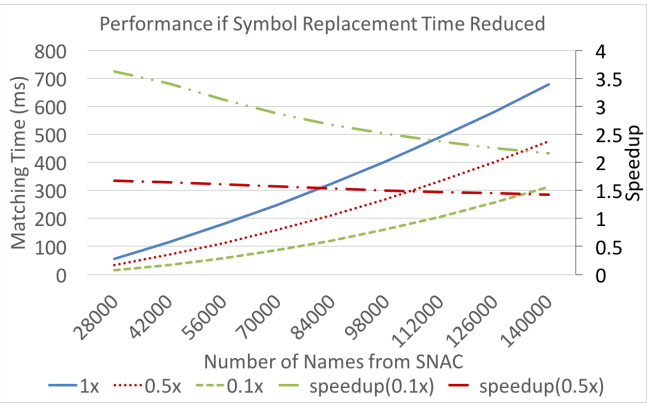

Figure 13. Performance if the symbol replacement time is reduced. (X axis represents the database size, ranging from 28,000 names to 140,000 names. Y axis on the left represents the matching time and Y axis on the right represents speedup against original performance.)

time than re-streaming the input. A reduction of 50% in the symbol replacement time leads to 1.42x to 1.67x speedup and a reduction of 90% leads to 2.16x to 3.63x speedup. The speedup decreases as the number of names increases, which implies that the symbol replacement time becomes less dominant for larger databases. For larger datasets like DBLP, the re-streaming time dominates the matching time. When the input size is 10 million, 94.7% of the matching time is used for re-streaming. In such a case, increasing input rate helps more to reduce the matching time. The speedup increases almost linearly as the input rate increases.

In summary, the AP shows advantages on both performance and result quality compared with different conventional methods. Note that the AP board we use to derive the performance is the first generation. Technology scaling projections and performance estimation suggest that, in the future, we may have a larger capacity and higher frequency, which could lead to even better performance.

VI. Summary and Future Work

In this paper, we propose a prototype using the AP to accelerate string-based ER problems. We present the design details of how to solve ER problems in several datasets. The proposed approach makes full use of the massive parallelism of the AP, and compares up to 14,000 names for SNAC and 45,000 names for DBLP in each pass. To evaluate the suitability of the AP approach, we measure both performance and resolution accuracy using various datasets from SNAC, DBLP, Fodors and Zagat. The AP approach achieves promising speeds and also enhances resolution accuracy (more correct pairs with less GMD cost) compared with conventional CPU methods. In summary, the AP shows great potential for accelerating string-based ER problems. Future work includes using the AP to process larger datasets, improve accuracy, and solve other string-based ER problems.

VII. Acknowledgements

This work is sponsored in part by C-FAR, a funded center of STARnet, a Semiconductor Research Corporation (SRC) program sponsored by MARCO and DARPA.

References

[1] Hernández, M. A. et al. The merge/purge problem for large databases. In *ACM SIGMOD Record*, 1995.

[2] *Social Networks and Archival Context*. http://socialarchive.iath.virginia.edu.

[3] Monge, A. et al. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the SIGMOD Workshop on Data Mining and Knowledge Discovery*, 1997.

[4] Whang, S. E. et al. Entity resolution with iterative blocking. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2009.

[5] Kolb, L. et al. Parallel entity resolution with dedoop. *Datenbank-Spektrum*, 2013.

[6] Benjelloun, O. et al. Swoosh: a generic approach to entity resolution. *The International Journal on Very Large Data Bases (VLDB)*, 2009.

[7] Dlugosch, P. et al. An efficient and scalable semiconductor architecture for parallel automata processing. *IEEE Transactions on Parallel and Distributed Systems*, 2014.

[8] Wang, K. et al. Association rule mining with the Micron Automata Processor. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2015.

[9] Roy, I. et al. Finding motifs in biological sequences using the Micron Automata Processor. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2014.

[10] Bo, C. et al. String kernel testing acceleration using the Micron Automata Processor. In *Proceedings of the Workshop on Computer Architecture for Machine Learning (CAMEL)*, 2015.

[11] Zhou, K. et al. Brill tagging on the Micron Automata Processor. In *Proceedings of the IEEE International Conference on Semantic Computing (ICSC)*, 2015.

[12] *Apache Lucene*. http://lucene.apache.org.

[13] Ley, M. The DBLP computer science bibliography: evolution, research issues, perspectives. In *Proceedings of the International Symposium on String Processing and Information Retrieval*. Springer, 2002.

[14] Tracy, T. II. et al. Nondeterministic finite automata in hardware-the case of the Levenshtein automaton. In *Proceedings of the Workshop on Archtectures and Systems for Big Data (ASBD)*, 2015.

[15] *Fuzzy document finding in Ruby*. https://github.com/brianhempel/fuzzy_tools.

[16] *Duplicate Detection, Record Linkage, and Identity Uncertainty: Datasets*. http://www.cs.utexas.edu/users/ml/riddle/data.html.

[17] Menestrina, D. et al. Evaluating entity resolution results. In *Proceedings of the VLDB Endowment*, 2010.