

Brill Tagging on the Micron Automata Processor

Keira Zhou; Jeffrey J. Fox; Ke Wang; Donald E. Brown, Fellow IEEE; Kevin Skadron, Fellow IEEE

University of Virginia
Charlottesville, VA 22904 USA
{qz4aq, jjf5x, kewang, brown, skadron}@virginia.edu

Abstract – Semantic analysis often uses a pipeline of Natural Language Processing (NLP) tools such as part-of-speech (POS) tagging. Brill tagging is a classic rule-based algorithm for POS tagging within NLP. However, implementation of the tagger is inherently slow on conventional Von Neumann architectures. In this paper, we accelerate the second stage of Brill tagging on the Micron Automata Processor, a new computing architecture that can perform massive pattern matching in parallel. The designed structure is tested with a subset of the Brown Corpus using 218 contextual rules. The results show a 38X speed-up for the second stage tagger implemented on a single AP chip, compared to a single thread implementation on CPU. This speed-up is linear with the number of rules, thus making large and/or complex rule sets computationally practical. This paper introduces the use of this new accelerator for computational linguistic tasks, particularly those that involve rule-based or pattern-matching approaches.

Keywords–Part-of-speech tagging; Brill tagging; Automata Processor; Natural Language Processing

I. INTRODUCTION

Semantic analysis often uses a pipeline of Natural Language Processing (NLP) tools such as part-of-speech (POS) tagging [14]. POS tagging makes assignments of a tag to input tokens, such as, nouns, verbs, adjectives, etc [1]. Accelerating these tools may be a promising approach for increasing the speed and potentially the sophistication of semantic analysis. We evaluate acceleration of POS tagging as a case study.

Brill Tagging is a classic rule-based POS tagging algorithm [9]. The algorithm is one of the most widely used rule-based approaches [6]. However, the computational time is slow for both training and tagging. Specifically, it may require RKn elementary steps to tag an input of n words with R contextual rules with at most K tokens of context [8].

The Micron Automata Processor (AP) [10] is a novel non-Von Neumann architecture that can be programmed to run thousands of Non-deterministic Finite Automata (NFA), i.e. regular-expression rules, in parallel to identify patterns in a data stream. The work described here shows that the AP’s parallelism can significantly

reduce the tagging time of Brill Tagging compared to implementation on a single-core CPU.

II. BACKGROUND AND RELATED WORK

A. POS tagging

Manually tagged corpora provide training data for automated taggers. POS tagging algorithms can be categorized into two groups: rule-based and stochastic (statistical) approaches. State-of-art stochastic based approaches include conditional random field models [7] and maximum entropy Markov models [12]. Brill tagging is one of the first and most widely used rule-based approaches [6]. When performing a POS tagging task, choosing a standard tagset is important. Two commonly used tagsets for POS tagging are the Brown (87 tags) [4] and Penn Treebank (36 tags) [5] tagsets.

The Brill algorithm proceeds in training and tagging steps. During the training, it identifies the most frequent tag for all recorded tokens as well as contextual rules for updating the tags. After training, a two-stage process tags new untagged corpora. The first stage assigns the most frequent tag to the new corpora. In the second stage, the initial tags are updated based on the contextual rules. Our work focuses on reducing the computational time of the second stage of the tagging by exploiting the fact that rules can be easily implemented in parallel as regular expressions on the AP.

III. AUTOMATA PROCESSOR

A physical embodiment of the AP is not yet available; however, we do have access to Micron’s simulator (SDK) of the AP. This allows us to design automata and simulate the on-chip processes and performance.

There are three major components on the AP: State-Transition-Element (STE), Counter Element, and Boolean Elements, among which STE is the core component. The Counter and Boolean elements are not needed for Brill tagging. 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 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

This work was supported by the Army Research Laboratory under grant number W911NF-10-2-0051; C-FAR, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA; and a grant from Micron Technology.

This is the authors' version of the final manuscript. The authoritative version can be found in the Proceedings of the 2015 IEEE International Conference on Semantic Computing, Feb. 2015.

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. Table I provides illustrations of basic STE functions.

One AP chip contains a total of 49,152 STEs, among which 6144 can report. There are expected to be 32 chips on a PCI Express AP board, which thus has 1,572,864 STEs that can all operate in parallel [3]. The CPU initializes the AP chips, sends data, and fetches results using PCIe transactions. Given the AP chips' 7.5ns clock period, one chip can process data at 128 M symbols/sec; higher processing rates are possible by sharing the AP resources among multiple streams. Our design easily fits inside a single AP chip, so multiple input streams or much larger rulesets could easily be supported.

IV. DESIGN ON THE AUTOMATA PROCESSOR

A. Brill Tagging initial steps

Brill tagging is implemented in C and source code is openly available [2]. This implementation uses the Penn Treebank tagset and contains 218 contextual rules. We use this implementation to run the first stage tagging and write the intermediate result into a separate file that serves as the input for the second stage tagging.

B. Design on the AP


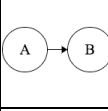
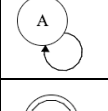

The input data for the AP consists of a stream of the words and initial tags. The rules are implemented on the AP as described below. All of the rules operate in parallel, inspecting the input stream for a potential match. Among the 218 rules, there are 19 different structures. The simplest rule structure matches a sequence of two tagged words, while the most complex structures look for patterns in sequences of up to seven tagged words. Fig. 1 provides an example design of an automaton with a simple structure that matches a sequence of two tags. We use “_” to represent white space. The reporting element ID contains a rule ID as well as the update tag. This information is needed for post processing (Section IV.C.)

We illustrate the design with an example input:

... to/TO conflict/NN with/IN...

The starting STE for the rule has a symbol “*”. This means that the STE will be *matched* by any input symbols. The symbol on the second STE is “^/”. Thus the second STE will be *matched* on any symbols that are *NOT* a “/”. This *activates* both the third STE and itself. The self-activating function will keep the second STE *activated* and accepting input symbols until a “/” appears. This ignores the actual words; the Brill rules are based on patterns of tags. When the “/” is seen, the third STE will be *matched* and *activate* the fourth STE; the next characters, up to a “_”, constitute a tag.

TABLE I. GRAPHIC ILLUSTRATIONS OF BASIC STE FUNCTIONS

	A starting STE: It can either be <i>start-of-data</i> , which only matches the first symbol of the input and matches against A; or <i>all-input-start</i> , which accepts every symbol from the input and matches against Symbol A
	Matching - Activating: Symbol A is matched on the first STE, and the second STE is activated; a “*” means to match any input symbols; “^” means a negation of the symbol
	Self-activating: The STE activates itself when Symbol A is matched
	A reporting STE: Reports when the symbol A is matched

This design assumes that every word is followed by a “/” and then its tag. Each word/tag pair ends with a “_”. If, at any point, a rule no longer matches the input, the “*” STE keeps the rule actively processing subsequent input.

C. Post processing

As the input data stream is being inspected by the rules on the AP, reporting elements generate output whenever a match is found. The output consists of an offset number that specifies the input symbol cycle on which one or more reporting elements fired, as well as the ID of the reporting elements. A post processing step is needed to use this output to update the tags and thus complete the second stage of Brill Tagging. We modified the openly available Brill code to conduct the post processing. In the original code, a word and a tag array are created when reading in the first-stage-tagged file. Our post processing requires another array that matches each character in the file with the word to which the character belongs. Table II shows this array for the example sentence we used previously.

After all the arrays are created, the baseline CPU code then reads in the contextual rule file that contains 218 rules and applies one rule at a time to the entire corpus. For the AP, we modified this part of the code. Instead of reading in the contextual rule file, the code now reads the output file from the AP Emulator and performs post-processing. Both rule ID and the update tag information can be obtained from the ID of the reporting STEs. The offset number indicates the character position at which an entire rule is matched. Using the offset number, we can use the character position array to look up the index of the word that needs to be updated. An example report we get from the AP looks like:

Offset 28 Reporting Element ID: rule2_VB

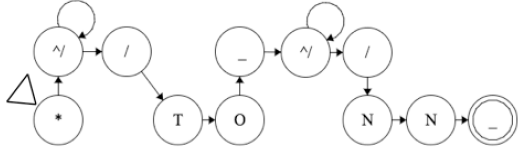


Figure 1. Example design of a simple automaton structure for the rule described in Section IV.B.

From the character position array, we find that the word associated with the 28th character is the 6th word. We thus update the tag of that word in the original corpus.

Fig. 2 shows the steps involved for the CPU and AP implementation. The **bold-italicized** parts are the execution time we included in the comparison. For the original Brill code on the CPU, for the most direct comparison with the AP, we only count the step for reading the rules, matching the rules, and updating tags. For the AP implementation, we include the estimate of the matching time on the AP (see Fig. 3, last column), and the pre/post processing time on the CPU to: 1) create the character position array; 2) read the file with the reported rules and 3) update the tags. The steps we ignore are identical in both the CPU and AP implementations.

V. TEST DATA AND RESULT

A. Test data

To test our design we use a subset of the Brown Corpus [4] downloaded from the NLTK website [7]. We combined the files into 5 different sizes: 40KB, 60KB, 79KB, 99KB to test the impact of input size on execution time for both the CPU and AP implementations. We also tested the largest file (99KB) with different numbers of rules ranging from 20 to 218 rules to test the impact of the number of rules on the execution time.

TABLE II. THE CHARACTER POSITION ARRAY

Characters	...	t	o	/	T	O	-
Index of the Char Position Array	...	11	12	13	14	15	16
Array Content (Index of Word Array)	...	4	4	4	5	5	5
Characters	c	o	n	f	l	i	e
Index	17	18	19	20	21	22	23
Array Content	5	5	5	5	5	5	5
Characters	t	/	N	N	-	w	i
Index	24	25	26	27	28	29	30
Array Content	5	5	5	5	5	6	6
Characters	t	h	/	I	N	-	...
Index	31	32	33	34	35	36	...
Array Content	6	6	6	6	6	6	...

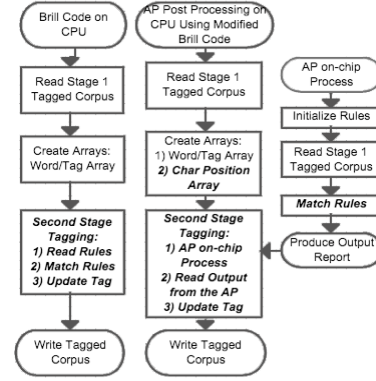


Figure 2. The CPU and AP Processes

B. Execution environment

For the CPU implementation, we used the published C code [2] and ran it on an Intel Core i5 machine with a single thread. The execution time recorded for the CPU implementation is the wall clock time for the step of updating the tags measured in microseconds.

For the AP implementation, one character is processed per clock cycle, so on-chip time is a direct function of the input size. Post processing is conducted on the same CPU. The execution time included for the post processing is the wall clock time for the steps of creating the extra character position array and updating the tags based on the output report from the AP.

C. Results

1) Execution time for different input data size

Table IV shows the execution time of the CPU and AP implementation for different sizes of input data (Time in microsecond). The speed-ups are within the range of 38.3X to 41.0X. This suggests that the size of the input does not have a significant impact on the speed-up.

2) Execution time for different number of rules

Fig. 3 shows the speed-up in relation to number of rules. We can see that there is an approximately linear growth of the speed-up with the number of rules. Although we only tested with up to 218 rules, the largest number of rules mentioned in the literature is 1729 [12]. Based on the regression line ($y = 0.1577x + 3.3794$), we estimate that the potential speed-up for such a rule set could be 276.0X.

TABLE III. EXECUTION TIME FOR DIFFERENT SIZES OF INPUT

	Time in microsecond	40 KB	60 KB	79 KB	99 KB
	CPU	56130	86545	112289	141810
A P	On chip	298	453	594	741
	Create Array	372	596	875	1031
	Post process	747	1063	1462	1935
	Total	1417	2112	2931	3707
	Speed-up	39.6X	41.0X	38.3X	38.3X

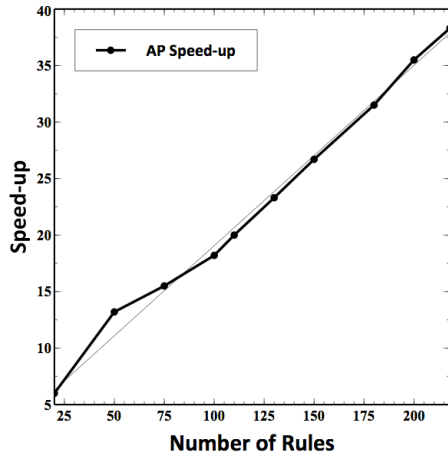


Figure 3. Speedup in Relation to Number of Rules

VI. ACCURACY AND DISCREPANCY

The Association for Computational Linguistics provides the accuracies of several POS tagging systems for the Wall Street Journal corpus [13]. The accuracy of Brill Tagging is listed at 96.5%, while Hidden Markov model is 96.96% and Maximum Entropy Markov Model is 97.32%. The implementation of Brill Tagging described here could make larger and more complex rule sets computationally practical. Such rule sets could allow Brill Tagging to achieve higher accuracy. On the other hand, the CPU implementation of Brill Tagging applies one rule at a time, while the AP implementation matches all rules in parallel, which can lead to differences in updating tags. For example, in the CPU version, for a given word, after rule 1 is applied and the tag is updated, rule 2 may not be triggered; while in the AP version, both rules will be triggered, thus the order of updating the tag will depend on the AP output. We used 4 different input files to estimate these discrepancies. We obtained the annotated Brown Corpus, which used the Brown tagset to determine the correct tag when the two methods disagreed. However, the 218 rules within the published software use the Penn Treebank tagset. Thus, the correct tags for a few tag differences were categorized as unknown. To set an upper bound for the decrease in accuracy, we count all unknown differences for the CPU implementation as correct and AP implementation as wrong. Table IV shows that such discrepancies are rare.

TABLE IV. TAGGING DISCREPANCIES FOR DIFFERENT SAMPLES

	ca01 (2242 words)	cb01 (2200 words)	cc01 (2415 words)	cd01 (2213 words)
CPU Correct	5	2	6	6
AP Correct	2	1	1	3
Both Wrong	0	1	1	1
Unknown	2	3	2	5
Difference in Accuracy	0.223%	0.182%	0.290%	0.362%
Average	0.264%			

VII. CONCLUSIONS AND FUTURE WORK

The Micron AP is a novel non-Von Neumann architecture that can test for thousands or even hundreds of thousands of patterns in parallel. This paper introduces an AP implementation of Brill tagging, and shows that the AP achieves a significant speed-up.

We plan to implement this design on hardware once the AP is available. In addition, we will more rigorously evaluate the accuracy of the CPU and AP designs relative to each other (combining rules may be one promising approach to reduce differences in tagging) and relative to state-of-the-art stochastic POS-tagging techniques.

This study suggests that the AP may be a promising platform for other semantic analysis tasks. We plan to explore other applications within NLP domain such as parsing and machine translation.

ACKNOWLEDGMENTS

The authors thank Mateja Putic for assistance in the initial setup, Micron Technology for providing access to the AP SDK, Matt Tanner for his insights into using the AP, and the anonymous reviewers for their helpful comments.

REFERENCES

- [1] Voutilainen, A. "Part-of-speech tagging." *The Oxford handbook of computational linguistics (2003)*: 219-232.
- [2] Brill, E. CIS, Univ. of Pennsylvania, and the Spoken Language Systems Group, Laboratory for Computer Science, MIT, 1994. http://www.tech.plym.ac.uk/soc/staff/guidbugm/software/RULE_BASED_TAGGER_V.1.14.tar.Z
- [3] Roy, I. and S. Aluru. "Finding Motifs in Biological Sequences Using the Micron Automata Processor." *IPDPS*, 2014.
- [4] Francis, W.N. and H. Kucera. *Brown Corpus Manual*, 1964.
- [5] Santorini, B. "Part-of-speech tagging guidelines for the Penn Treebank Project (3rd revision)." 1990.
- [6] Mohammad, S. and T. Pedersen. "Guaranteed pre-tagging for the brill tagger." *CICLing '03. Springer Berlin*, 2003. 148-157.
- [7] NLTK Corpora. *Natural Language Toolkit*. Web. 2013.
- [8] Roche, E. and Y. Schabes. "Deterministic part-of-speech tagging with finite-state transducers." *Computational linguistics 21.2 (1995)*: 227-253.
- [9] Brill, E. "Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging." *Computational linguistics 21.4 (1995)*: 543-565.
- [10] Dlugosch, P., D. Brown, P. Glendenning, M. Leventhal, and H. Noyes. "An efficient and scalable semiconductor architecture for parallel automata processing." *IEEE TPDS*, 2014.
- [11] Brill, E. "Unsupervised learning of disambiguation rules for part of speech tagging." *WVLC-3. Vol. 30. New Jersey: ACL*, 1995.
- [12] McCallum, A., D. Freitag, and F. CN Pereira. "Maximum Entropy Markov Models for Information Extraction and Segmentation." *ICML*. 2000.
- [13] "POS Tagging (State of the art)". *Wiki of the ACL*. Web. 2013.
- [14] Sheu, P., et al., eds. *Semantic Computing*. John Wiley & Sons, 2011.