A HARDWARE ALGORITHM FOR HIGH SPEED MORPHEME EXTRACTION AND ITS IMPLEMENTATION

Toshikazu Fukushima, Yutaka Ohyama and Hitoshi Miyai
C&C Systems Research Laboratories, NEC Corporation
1-1, Miyazaki 4-chome, Miyamae-ku, Kawasaki City, Kanagawa 213, Japan
(fuku@tsl.cl.nec.co.jp, ohyama@tsl.cl.nec.co.jp, miya@tsl.cl.nec.co.jp)

ABSTRACT

This paper describes a new hardware algorithm for morpheme extraction and its implementation on a specific machine (*MEX-I*), as the first step toward achieving natural language parsing accelerators. It also shows the machine's performance, 100–1,000 times faster than a personal computer. This machine can extract morphemes from 10,000 character Japanese text by searching an 80,000 morpheme dictionary in 1 second. It can treat multiple text streams, which are composed of character candidates, as well as one text stream. The algorithm is implemented on the machine in linear time for the number of candidates, while conventional sequential algorithms are implemented in combinational time.

1 INTRODUCTION

Recent advancement in natural language parsing technology has especially extended the word processor market and the machine translation system market. For further market extension or new market creation for natural language applications, parsing speed-up as well as improving parsing accuracy is required. First, the parsing speed-up directly reduces system response time required in such interactive natural language application systems as those using natural language interface, speech recognition, Kana-to-Kanji [1] conversion, which is the most popular Japanese text input method, and so on. Second, it also increases the advantage of such applications as machine translation, document proofreading, automatic indexing, and so on, which are used to treat a large amount of documents. Third, it realizes parsing methods based on larger scale dictionary or knowledge database, which are necessary to improve parsing accuracy.

Until now, in the natural language processing field, the speed-up has depended mainly on performance improvements achieved in sequential processing computers and the development of sequential algorithms. Recently, because of the further speeded-up requirement, parallel processing computers have been designed and parallel parsing algorithms (Matsumoto, 1986) (Haas, 1987) (Rytter, 1987) (Fukushima, 1990b) have been proposed. However, there are many difficult problems blocking efficient practical use of parallel processing computers. One of the problems is that access conflicts occur when several processors read or write a common memory simultaneously. Another is the bottle-neck problem, wherein communication between any two processors is restricted, because of hardware scale limitation.

On the other hand, in the pattern processing field, various kinds of accelerator hardware have been developed. They are designed for a special purpose, not for general purposes. A hardware approach hasn't been tried in the natural language processing field yet.

The authors propose developing natural language parsing accelerators, a hardware approach to the parsing speed-up (Fukushima, 1989b) (Fukushima, 1990a). This paper describes a new hardware algorithm for high speed morpheme extraction and its implementation on a specific machine. This morpheme extraction machine is designed as the first step toward achieving the natural language parsing accelerators.

2 MACHINE DESIGN STRATEGY

2.1 MORPHEME EXTRACTION

Morphological analysis methods are generally composed of two processes: (1) a morpheme extraction process and (2) a morpheme determination process. In process (1), all morphemes, which are considered as probably being used to construct input text, are extracted by searching a morpheme dictionary. These morphemes are extracted as candidates. Therefore, they are selected mainly by morpheme conjunction constraint. Morphemes which actually construct the text are determined in process (2).

The authors selected morpheme extraction as the first process to be implemented on specific hardware, for the following three reasons. First is that the speed-up requirement for the morphological analysis process is very strong in Japanese

[1] Kana characters are combined consonant and vowel symbols used in written Japanese. Kanji characters are Chinese ideographs.

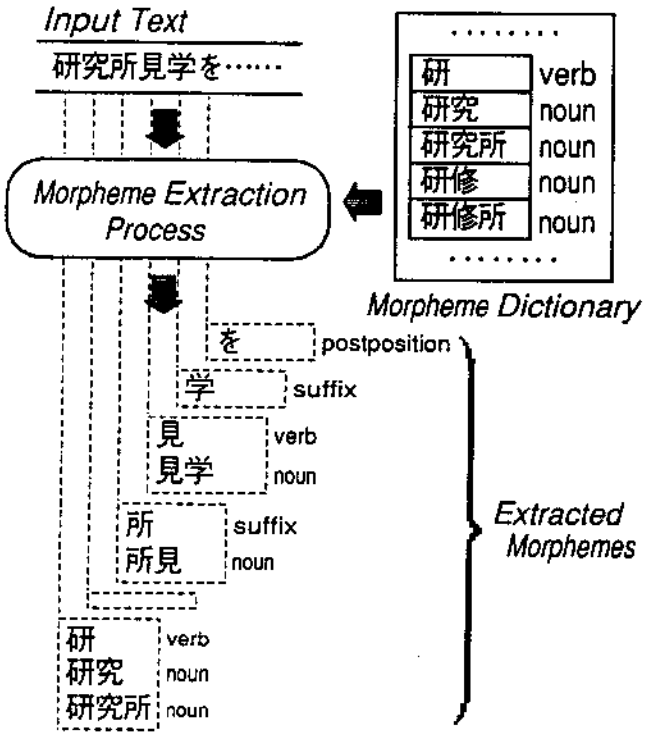

Figure 1: Morpheme Extraction Process for Japanese Text

text parsing systems. This process is necessary for natural language parsing, because it is the first step in the parsing. However, it is more laborious for Japanese and several other languages, which have no explicit word boundaries, than for English and many European languages (Miyazaki, 1983) (Ohyama, 1986) (Abe, 1986). English text reading has the advantage of including blanks between words. Figure 1 shows an example of the morpheme extraction process for Japanese text. Because of the disadvantage inherent in reading difficulty involved in all symbols being strung together without any logical break between words, the morpheme dictionary, including more than 50,000 morphemes in Japanese, is searched at almost all positions of Japanese text to extract morphemes. The authors' investigation results, indicating that the morpheme extraction process requires using more than 70 % of the morphological analysis process time in conventional Japanese parsing systems, proves the strong requirement for the speed-up.

The second reason is that the morpheme extraction process is suitable for being implemented on specific hardware, because simple character comparison operation has the heaviest percentage weight in this process. The third reason is that this speed-up will be effective to evade the common memory access conflict problem mentioned in Section 1.

2.2 STRATEGY DISCUSSION

In conventional morpheme extraction methods, which are the software methods used on sequential processing computers, the comparison operation between one key string in the morpheme dictionary and one sub-string of input text is repeated. This is one to one comparison. On the other hand, many to one comparison or one to many comparison is practicable in parallel computing.

Content-addressable memories (CAMs) (Chisvin, 1989) (Yamada, 1987) realize the many to one comparison. One sub-string of input text is simultaneously compared with all key strings stored in a CAM. However, presently available CAMs have only a several tens of kilobit memory, which is too small to store data for a more than 50,000 morpheme dictionary.

The above mentioned parallel processing computers realize the one to many comparison. On the parallel processing computers, one processor searches the dictionary at one text position, while another processor searches the same dictionary at the next position at the same time (Nakamura, 1988). However, there is an access conflict problem involved, as already mentioned.

The above discussion has led the authors to the following strategy to design the morpheme extraction machine (Fukushima, 1989a). This strategy is to shorten the one to one comparison cycle. Simple architecture, which will be described in the next section, can realize this strategy.

3 A HARDWARE ALGORITHM FOR MORPHEME EXTRACTION

3.1 FUNDAMENTAL ARCHITECTURE

A new hardware algorithm for the morpheme extraction, which was designed with the strategy mentioned in the previous section, is described in this section.

The fundamental architecture, used to implement the algorithm, is shown in Fig. 2. The main components of this architecture are a dictionary block, a shift register block, an index memory, an address generator and comparators.

The dictionary block consists of character memories (i.e. 1st character memory, 2nd character memory, ..., N-th character memory). The n-th character memory ($1 \leq n \leq N$) stores n-th characters of all key strings in the morpheme dictionary, as shown in Fig. 3. In Fig. 3, "献本", "研", "研究", "研究所", "研修", and so on are Japanese morphemes. As regarding morphemes shorter than the key length N, pre-defined remainder symbols fill in their key areas. In Fig. 3, '*' indicates the remainder symbol.

The shift register block consists of character registers (i.e. 1st character register, 2nd character register, ..., N-th character register). These registers

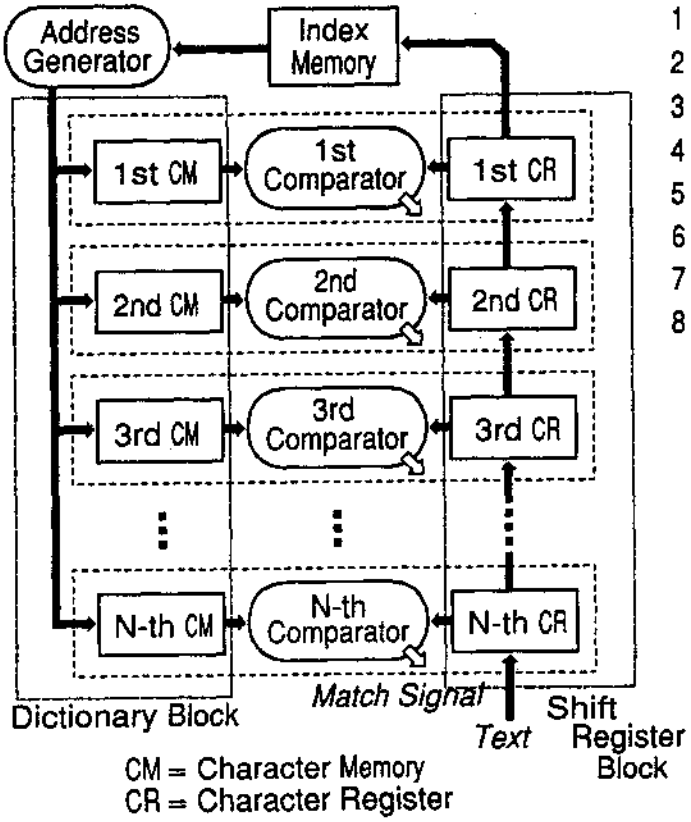

Figure 2: Fundamental Architecture

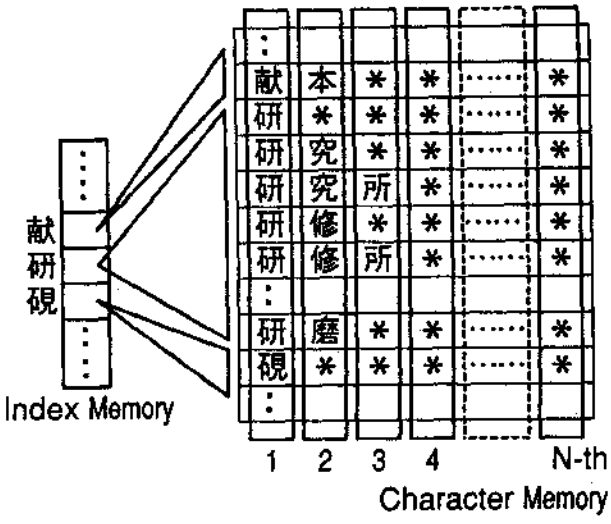

Figure 3: Relation between Character Memories and Index Memory

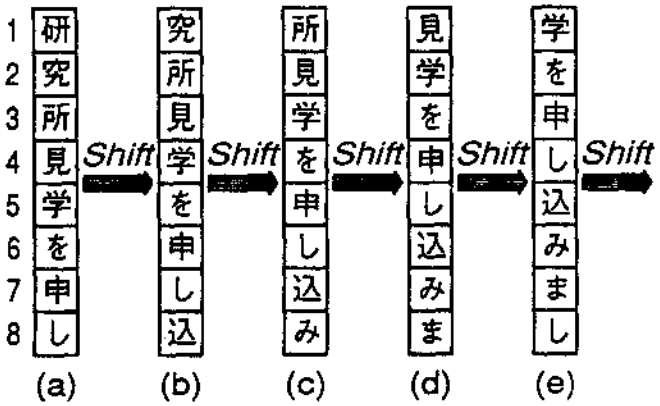

Figure 4: Movement in Shift Register Block

store the sub-string of input text, which can be shifted, as shown in Fig. 4. The index memory receives a character from the 1st character register. Then, it outputs the top address and the number of morphemes in the dictionary, whose 1st character corresponds to the input character. Because morphemes are arranged in the incremental order of their key string in the dictionary, the pair for the top address and the number expresses the address range in the dictionary. Figure 3 shows the relation between the index memory and the character memories. For example, when the shift register block content is as shown in Fig. 4(a), where '研' is stored in the 1st character register, the index memory's output expresses the address range for the morpheme set {"研", "研究", "研究所", "研修", "研修所", ..., "研磨"} in Fig. 3.

The address generator sets the same address to all the character memories, and changes their addresses simultaneously within the address range which the index memory expresses. Then, the dictionary block outputs all characters constructing one morpheme (key string with length N) simultaneously at one address. The comparators are N in number (i.e. 1st comparator, 2nd comparator, ..., N-th comparator). The n-th comparator compares the character in the n-th character register with the one from the n-th character memory. When there is correspondence between the two characters, a match signal is output. In this comparison, the remainder symbol operates as a wild card. This means that the comparator also outputs a match signal when the n-th character memory outputs the remainder symbol. Otherwise, it outputs a *no match* signal.

The algorithm, implemented on the above described fundamental architecture, is as follows.

- **Main procedure**

  Step 1: Load the top N characters from the input text into the character registers in the shift register block.

Step 2: While the text end mark has not arrived at the 1st character register, implement **Procedure 1**.

- **Procedure 1**

  Step 1: Obtain the address range for the morphemes in the dictionary, whose 1st character corresponds to the character in the 1st character register. Then, set the top address for this range to the current address for the character memories.

  Step 2: While the current address is in this range, implement **Procedure 2**.

  Step 3: Accomplish a shift operation to the shift register block.

- **Procedure 2**

  Step 1: Judge the result of the simultaneous comparisons at the current address. When all the comparators output match signals, detection of one morpheme is indicated. When at least one comparator outputs the *no match* signal, there is no detection.

  Step 2: Increase the current address.

For example, Fig. 4(a) shows the sub-string in the shift register block immediately after **Step 1** for **Main procedure**, when the input text is "研究所見学を申し込みました...". **Step 3** for **Procedure 1** causes such movement as (a)→(b), (b)→(c), (c)→(d), (d)→(e), and so on. **Step 1** and **Step 2** for **Procedure 1** are implemented in each state for (a), (b), (c), (d), (e), and so on.

In state (a) for Fig. 4, the index memory's output expresses the address range for the morpheme set {"研", "研究", "研究所", "研修", "研修所", ..., "研磨"} if the dictionary is as shown in Fig. 3. Then, **Step 1** for **Procedure 2** is repeated at each address for the morpheme set {"研", "研究", "研究所", "研修", "研修所", ..., "研磨"}.

Figure 5 shows two examples of **Step 1** for **Procedure 2**. In Fig. 5(a), the current address for the dictionary is at the morpheme "研究所". In Fig. 5(b), the address is at the morpheme "研修所". In Fig. 5(a), all of the eight comparators output match signals as the result of the simultaneous comparisons. This means that the morpheme "研究所" has been detected at the top position of the sub-string "研究所見学を申し". On the other hand, in Fig. 5(b), seven comparators output match signals, but one comparator, at 2nd position, outputs a *no match* signal, due to the discord between the two characters, '修' and '究'. This means that the morpheme "研修所" hasn't been detected at this position.

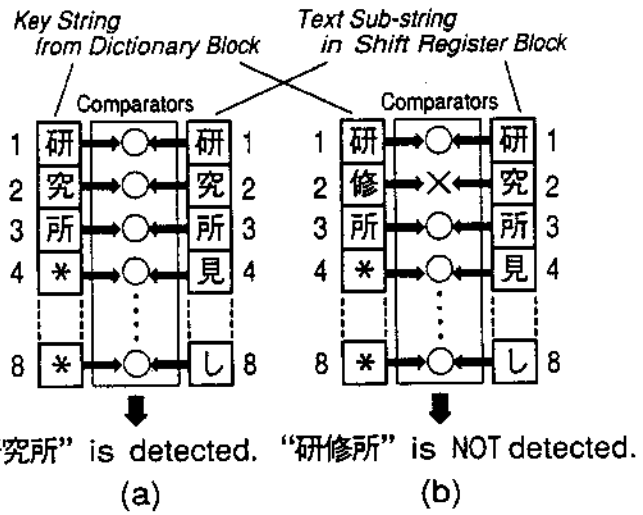

Figure 5: Simultaneous Comparison in Fundamental Architecture

3.2 EXTENDED ARCHITECTURE

The architecture described in the previous section treats one stream of text string. In this section, the architecture is extended to treat multiple text streams, and the algorithm for extracting morphemes from multiple text streams is proposed.

Generally, in character recognition results or speech recognition results, there is a certain amount of ambiguity, in that a character or a syllable has multiple candidates. Such multiple candidates form the multiple text streams. Figure 6(a) shows an example of multiple text streams, expressed by a two dimensional matrix. One dimension corresponds to the position in the text. The other dimension corresponds to the candidate level. Candidates on the same level form one stream. For example, in Fig. 6(a), the character at the 3rd position has three candidates: the 1st candidate is '析', the 2nd one is '折' and the 3rd one is '所'. The 1st level stream is "砿究析見...". The 2nd level stream is "研突折貝...". The 3rd level stream is "切空所目...".

Figure 6(b) shows an example of the morphemes extracted from the multiple text streams shown in Fig. 6(a). In the morpheme extraction process for the multiple text streams, the key strings in the morpheme dictionary are compared with the combinations of various candidates. For example, "研究所", one of the extracted morphemes, is composed of the 2nd candidate at the 1st position, the 1st candidate at the 2nd position and the 3rd candidate at the 3rd position. The architecture described in the previous section can be easily extended to treat multiple text streams. Figure 7

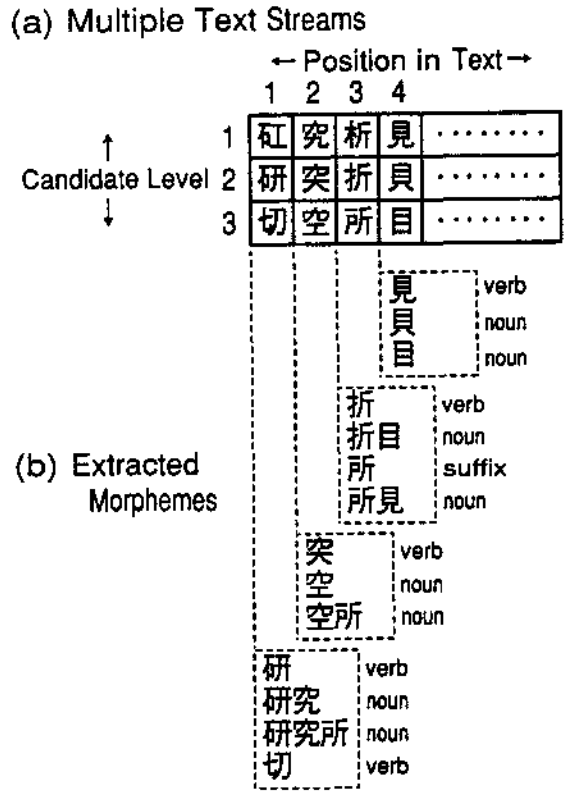

Figure 6: Morpheme Extraction from Multiple Text Streams

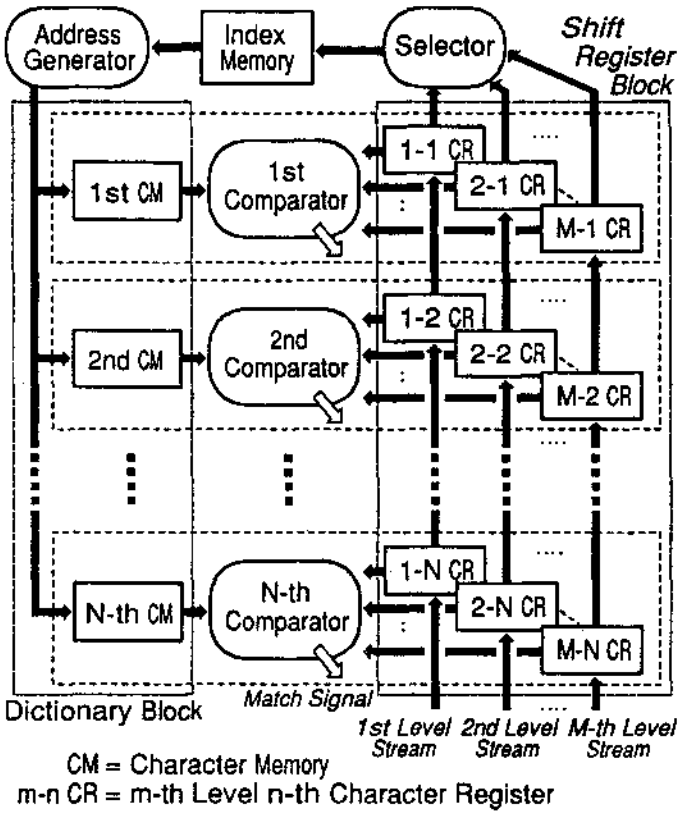

Figure 7: Extended Architecture

shows the extended architecture. This extended architecture is different from the fundamental architecture, in regard to the following three points.

First, there are M sets of character registers in the shift register block. Each set is composed of N character registers, which store and shift the sub-string for one text stream. Here, M is the number of text streams. N has already been introduced in Section 3.1. The text streams move simultaneously in all the register sets.

Second, the n-th comparator compares the character from the n-th character memory with the M characters at the n-th position in the shift register block. A match signal is output, when there is correspondence between the character from the memory and either of the M characters in the registers.

Third, a selector is a new component. It changes the index memory's input. It connects one of the registers at the 1st position to sequential index memory inputs in turn. This changeover occurs M times in one state of the shift register block.

Regarding the algorithm described in Section 3.1, the following modification enables treating multiple text streams. **Procedure 1** and **Procedure 1.5**, shown below, replace the previous **Procedure 1**.

- **Procedure 1**

  Step 1: Set the highest stream to the current level.

  Step 2: While the current level has not exceeded the lowest stream, implement **Procedure 1.5**.

  Step 3: Accomplish a shift operation to the shift register block.

- **Procedure 1.5**

  Step 1: Obtain the address range for the morphemes in the dictionary, whose 1st character corresponds to the character in the register at the 1st position with the current level. Then, set the top address for this range to the current address for the character memories.

  Step 2: While the current address is in this range, implement **Procedure 2**.

  Step 3: Lower the current level.

Figure 8 shows an example of **Step 1** for **Procedure 2**. In this example, all of the eight comparators output the match signal as a result of simultaneous comparisons, when the morpheme from the dictionary is "研究所". Characters marked with a circle match the characters from the dictionary. This means that the morpheme "研究所" has been detected.

When each character has M candidates, the worst case time complexity for sequential morpheme extraction algorithms is $O(M^N)$. On the other hand, the above proposed algorithm (Fukushima's algorithm) has the advantage that the time complexity is $O(M)$.

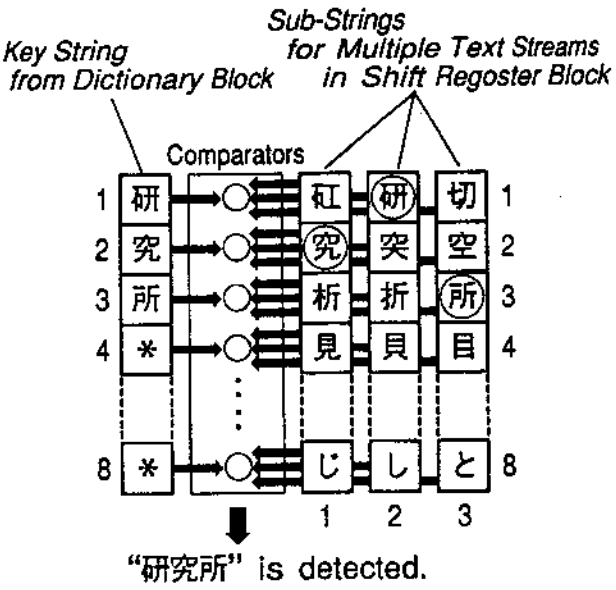

Figure 8: Simultaneous Comparison in Extended Architecture

Hamaguchi's hardware algorithm (Hamaguchi, 1988), proposed for speech recognition systems, is similar to Fukushima's algorithm. In Hamaguchi's algorithm, S bit memory space expresses a set of syllables, when there are S different kinds of syllables ($S = 101$ in Japanese). The syllable candidates at the same position in input phonetic text are located in one S bit space. Therefore, Hamaguchi's algorithm shows more advantages, as the full set size of syllables is smaller and the number of syllable candidates is larger. On the other hand, Fukushima's algorithm is very suitable for text with a large character set, such as Japanese (more than 5,000 different characters are computer readable in Japanese). This algorithm also has the advantage of high speed text stream shift, compared with conventional algorithms, including Hamaguchi's.

4 A MORPHEME EXTRACTION MACHINE

4.1 A MACHINE OUTLINE

This section describes a morpheme extraction machine, called *MEX-I*. It is specific hardware which realizes extended architecture and algorithm proposed in the previous section.

It works as a backend machine for NEC Personal Computer PC-9801VX (CPU: 80286 or V30, clock: 8MHz or 10MHz). It receives Japanese text from the host personal computer, and returns morphemes extracted from the text after a bit of time.

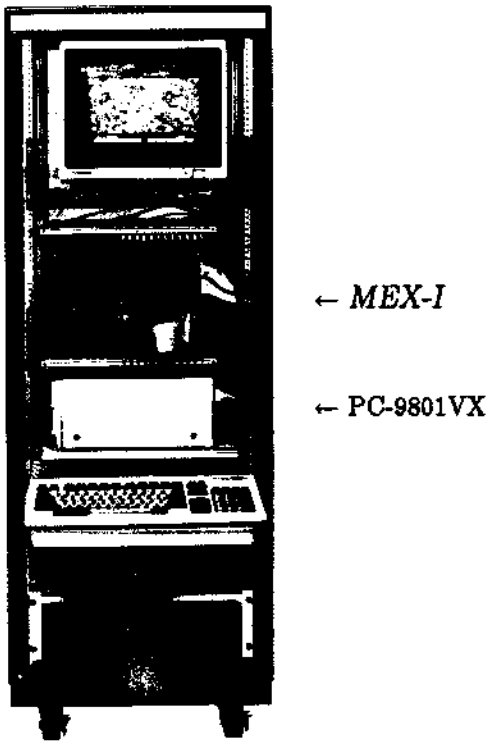

Figure 9: System Overall View

Figure 9 shows an overall view of the system, including *MEX-I* and its host personal computer. *MEX-I* is composed of 12 boards. Approximately 80 memory IC chips (whose total memory storage capacity is approximately 2MB) and 500 logic IC chips are on the boards.

The algorithm parameters in *MEX-I* are as follow. The key length (the maximum morpheme length) in the dictionary is 8 (i.e. $N = 8$). The maximum number of text streams is 3 (i.e. $M = 1, 2, 3$). The dictionary includes approximately 80,000 Japanese morphemes. This dictionary size is popular in Japanese word processors. The data length for the memories and the registers is 16 bits, corresponding to the character code in Japanese text.

4.2 EVALUATION

MEX-I works with 10MHz clock (i.e. the clock cycle is 100ns). **Procedure 2**, described in Section 3.1, including the simultaneous comparisons, is implemented for three clock cycles (i.e. 300ns). Then, the entire implementation time for morpheme extraction approximates $A \times D \times L \times M \times 300ns$. Here, D is the number of all morphemes in the dictionary, L is the length of input text, M is the number of text streams, and A is the indexing coefficient. This coefficient means the average rate for the number of compared morphemes, compared to the number of all morphemes in the dictionary.

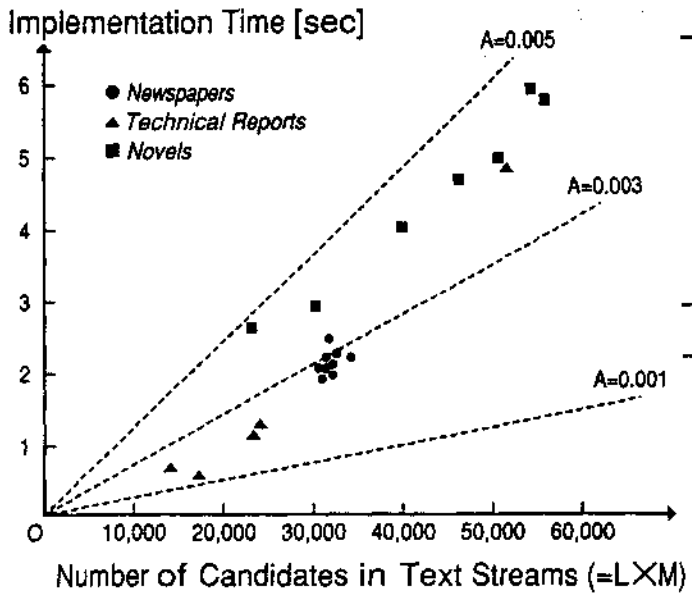

Figure 10: Implementation Time Measurement Results

Morpheme Extraction Methods	Text1	Text2	Text3	Text4
Programs Based on Sequential Algorithms				[sec]
• Binary Search Method (Knuth, 1973)	564	642	615	673
• Binary Search Method Checking Top Character Index	133	153	147	155
• Ordered Hash Method (Amble, 1974)	406	440	435	416
• Digital Search Method (Knuth, 1973) with Tree Structure Index	52	56	54	54
MEX-I	0.56	0.50	0.51	0.44

Figure 11: Implementation Time Comparison for 5,000 Character Japanese Text

The implementation time measurement results, obtained for various kinds of Japanese text, are plotted in Fig. 10. The horizontal scale in Fig. 10 is the $L \times M$ value, which corresponds to the number of characters in all the text streams. The vertical scale is the measured implementation time. The above mentioned 80,000 morpheme dictionary was used in this measurement. These results show performance wherein *MEX-I* can extract morphemes from 10,000 character Japanese text by searching an 80,000 morpheme dictionary in 1 second.

Figure 11 shows implementation time comparison with four conventional sequential algorithms. The conventional algorithms were carried out on NEC Personal Computer PC-98XL^2 (CPU: 80386, clock: 16MHz). Then, the 80,000 morpheme dictionary was on a memory board. Implementation time was measured for four different Japanese text samplings. Each of them forms one text stream, which includes 5,000 characters. In these measurement results, *MEX-I* runs approximately 1,000 times as fast as the morpheme extraction program, using the simple binary search algorithm. It runs approximately 100 times as fast as a program using the digital search algorithm, which has the highest speed among the four algorithms.

5 CONCLUSION

This paper proposes a new hardware algorithm for high speed morpheme extraction, and also describes its implementation on a specific machine. This machine, *MEX-I*, is designed as the first step toward achieving natural language parsing accelerators, which is a new approach to speeding up the parsing.

The implementation time measurement results show performance wherein *MEX-I* can extract morphemes from 10,000 character Japanese text by searching an 80,000 morpheme dictionary in 1 second. When input is one stream of text, it runs 100–1,000 times faster than morpheme extraction programs on personal computers.

It can treat multiple text streams, which are composed of character candidates, as well as one stream of text. The proposed algorithm is implemented on it in linear time for the number of candidates, while conventional sequential algorithms are implemented in combinational time. This is advantageous for character recognition or speech recognition.

Its architecture is so simple that the authors believe it is suitable for VLSI implementation. Actually, its VLSI implementation is in progress. A high speed morpheme extraction VLSI will improve the performance of such text processing applications in practical use as Kana-to-Kanji conversion Japanese text input methods and spelling checkers on word processors, machine translation, automatic indexing for text database, text-to-speech conversion, and so on, because the morpheme extraction process is necessary for these applications.

The development of various kinds of accelerator hardware for the other processes in parsing is work for the future. The authors believe that the hardware approach not only improves conventional parsing methods, but also enables new parsing methods to be designed.

REFERENCES

Abe, M., Ooshima, Y., Yuura, K. and Takeichi, N. (1986). "A Kana-Kanji Translation System for Non-segmented Input Sentences Based on Syntactic and Semantic Analysis", *Proc. 11th International Conference on Computational Linguistics*: 280–285.

Amble, O. and Knuth, D. E. (1974). "Ordered Hash Tables", *The Computer Journal, 17(2)*: 135–142.

Bear, J. (1986). "A Morphological recognizer with Syntactic and Phonological Rules", *Proc. 11th International Conference on Computational Linguistics*: 272–276.

Chisvin, L. and Duckworth, R. J. (1989). "Content-Addressable and Associative Memory: Alternatives to the Ubiquitous RAM", *Computer*: 51–64.

Fukushima, T., Kikuchi, Y., Ohyama, Y. and Miyai, H. (1989a). "A Study of the Morpheme Extraction Methods with Multi-matching Technique" (in Japanese), *Proc. 39th National Convention of Information Processing Society of Japan*: 591–592.

Fukushima, T., Ohyama, Y. and Miyai, H. (1989b). "Natural Language Parsing Accelerators (1): An Experimental Machine for Morpheme Extraction" (in Japanese), *Proc. 39th National Convention of Information Processing Society of Japan*: 600–601.

Fukushima, T., Ohyama, Y. and Miyai, H. (1990a). "Natural Language Parsing Accelerators (1): An Experimental Machine for Morpheme Extraction" (in Japanese), *SIG Reports of Information Processing Society of Japan, NL75(9)*.

Fukushima, T. (1990b). "A Parallel Recognition Algorithm of Context-free Language by Array Processors" (in Japanese), *Proc. 40th National Convention of Information Processing Society of Japan*: 462–463.

Haas, A. (1987). "Parallel Parsing for Unification Grammar", *Proc. 10th International Joint Conference on Artificial Intelligence*: 615–618.

Hamaguchi, S. and Suzuki, Y. (1988). "Hardware-matching Algorithm for High Speed Linguistic Processing in Continuous Speech-recognition Systems", *Systems and Computers in Japan, 19(7)*: 72–81.

Knuth, D. E. (1973). *Sorting and Searching, The Art of Computer Programming, Vol.3.* Addison-Wesley.

Koskenniemi, K. (1983). "Two-level Model for Morphological Analysis", *Proc. 8th International Joint Conference on Artificial Intelligence*: 683–685.

Matsumoto, Y. (1986). "A Parallel Parsing System for Natural Language Analysis", *Proc. 3rd International Conference of Logic Programming, Lecture Notes in Computer Science*: 396–409.

Miyazaki, M., Goto, S., Ooyama, Y. and Shirai, S. (1983). "Linguistic Processing in a Japanese-text-to-speech-system", *International Conference on Text Processing with a Large Character Set*: 315–320.

Nakamura, O., Tanaka, A. and Kikuchi, H. (1988). "High-Speed Processing Method for the Morpheme Extraction Algorithm" (in Japanese), *Proc. 37th National Convention of Information Processing Society of Japan*: 1002–1003.

Ohyama, Y., Fukushima, T., Shutoh, T. and Shutoh, M. (1986). "A Sentence Analysis Method for a Japanese Book Reading Machine for the Blind", *Proc. 24th Annual Meeting of Association for Computational Linguistics*: 165–172.

Russell, G. J., Ritchie, G. D., Pulman, S. G. and Black, A. W. (1986). "A Dictionary and Morphological Analyser for English", *Proc. 11th International Conference on Computational Linguistics*: 277–279.

Rytter, W. (1987). "Parallel Time O(log n) Recognition of Unambiguous Context-free Languages", *Information and Computation, 73*: 75–86.

Yamada, H., Hirata, M., Nagai, H. and Takahashi, K. (1987). "A High-speed String-search Engine", *IEEE Journal of Solid-state Circuits, SC-22(5)*: 829–834.