人工知能学会誌 Vol. 10 No. 4 (Jul. 1995) 〔別刷〕

論文 Technical Papers

A Parallel Chart-Based Parser for Analyzing Ill-Formed Inputs

Thanaruk Theeramunkong* Hozumi Tanaka*

* Dept. of Computer Science, Tokyo Institute of Technology, Tokyo 152, Japan.

Received March 16, 1994

Keywords: chart parsing, ill-formed input analysis, parallel inference machine (PIM), static and dynamic load balancing.

Summary

When a natural language processing system encounters unparsable inputs, the analysis should not be rejected. Instead, the system should attempt to detect the cause of ill-formedness and generate a set of possible interpretations. However, parsing ill-formed inputs suffers from large computation time due to extra mechanisms for detecting the existing ill-formedness. This indicates the importance of developing a parallel robust parsing algorithm. The goal of this research is to develop an effective parallel algorithm for parsing an ill-formed input under a loosely-coupled hardware environment. The parallel parser is implemented on PIM/m with 256 processors. Through the experiments, we point out that parsing ill-formed inputs with the proposed parser can acquire a satisfactory result in its performance.

1. Introduction

When people use language spontaneously, they do not always pay attention to their grammar structure. According to an extensive study by Thompson [Thompson 80], 33% of inputs in his query system were unparsable due to vocabulary problems, punctuation errors, ungrammaticality and spelling errors. A system should not reject the analysis of the inputs but should find and fix the ill-formedness. Coping with such ill-formedness is a challenge in natural language parsing.

In the past decades, there have been several attempts to modify existing parsing algorithms to handle ill-formed inputs based on ATN [Kwasny 81, Weischedel 83], Chart parsing [Kato 91, Mellish 89] and GLR [Saito 88, Tomita 86]. However, as Mellish [Mellish 89] points out, handling such ill-formed inputs takes a tremendous amount of time. According to our preliminary experiments, parsing an ill-formed input including more than two errors may be 10 000 times slower than parsing a grammatically correct input with the same length. Such slowness causes many problems when developing a practical query system that needs real time response to realize smooth human-machine communication. Thus, it is urged to develop a natural language processing system that can not only handle ill-formed inputs but also parse them at a reasonable speed. The latter emphasizes the importance of parallel parsing algorithm for ill-formed inputs.

In general, there is no universal parallel algorithm that is applicable to every kind of problems and the effectiveness of parallel algorithms varies in individual cases. Considering specific characteristics of each problem, some carefulness is needed in tuning the parallel algorithm [Nitta 92]. However, for certain problems it is not the case that we can get full gain even after carefully tuning the parallel (then, there are n streams for the input with n-word length). The messages in ith stream are inactive edges starting at i.

If we have an AEP $A \rightarrow A_1 A_2 \cdot A_3 A_4$ from i to j (in a certain processor), this process will try to match an inactive edge of category A_3 in the jth stream. If there is such an inactive edge, a new active edge $A \rightarrow A_1 A_2 A_3 \cdot A_4$ will be generated as a new AEP and then distributed to another processor. Later, if there is also an inactive edge of category A_4, an inactive edge of category A is generated and then pushed in the ith stream. At this point, if $X \rightarrow AY$ is a production, an AEP $X \rightarrow A \cdot Y$ will be generated and distributed to a certain processor.

The static version distributes all the active edges, starting at i, to the ith processor. The number of processors in use correspond to the length of the input sentence. At first glance, this version seems to gain less parallelism than the dynamic version. However, by this method, all messages (inactive edges) in each stream are generated by the AEPs in the same processor and there will be no need to distribute AEPs to different processors any more. Then, compared with the dynamic approach, we can expect less interprocessor communication cost in the static approach. In chap. 4, we will show an experimental comparison between these two versions.

3・2 Parallel Non-left-corner Bottom-up Process

When analyzing ill-formed inputs, the restriction of left-rightness of the P-LC-BU is not appropriate because it would suppress several subparses that are useful for hypothesizing the existing errors. The parallel non-left-corner bottom-up process (P-NLC-BU) relaxes this restriction and generates some other active edges. These newly generated edges are helpful not only for hypothesizing errors, but also for reducing the searching space in the error recovery process [Kato 91]. In order to realize the P-NLC-BU, the bottom-up rule and the fundamental rule of traditional chart parsing have to be modified to allow operating from arbitrary positions in the RHS of a grammar rule or in the undeterminated portion of an active edge.

Non Left Corner Bottom up Rule :
for all $\{i, j, C_1, [\,]\}$ in ST_i
  for all $C \rightarrow Cs_1, C_1, Cs_2 \in G$, s.t. $Cs_1 \neq [\,]$
    if $Cs_2 = [\,]$ then generate a process,
      $\{*, j, C, [(*, i, Cs_1)]\}$
    else generate a process,
      $\{*, *, C, [(*, i, Cs_1), (j, *, Cs_2)]\}$
Non Left Corner Fundamental Rule :
for all processes,
  $\{S, E, C, [\ldots, (s_1, e_1, [Cs_1, C_1, Cs_2])]\}$
  for all $\{S_1, E_1, C_1, [\,]\}$ in ST_{S_1}
    if $s_1 \leq S_1$ or $s_1 = *$ and $E_1 \leq e_1$ or $e_1 = *$
      then generate a process,
        $\{S, E, C, [\ldots, (s_1, S_1, Cs_1), (E_1, e_1, Cs2)]\}$

where ST_i is the ith stream, G is the set of grammar rules, C_i is a category and Cs_j is a sequence of categories.

Fig. 1 Rules in P-NLC-BU.

Fig. 1 shows two modified rules applied in this process: the *non-left-corner bottom-up* rule (NLC-BU rule) and the *non-left-corner fundamental* rule (NLC-F rule). In these rules, an edge is generalized and represented in the form of $\{SP, EP, Cat, Unparsed\}$ [*1], where SP (EP) specifies the starting (ending) position of the edge in the chart. *Cat* is the category of the edge and *Unparsed* is the unparsed part of the edge. Both SP and EP are denoted by an integer for determined position, or by '$*$' for undetermined position. The NLC-BU rule provokes the pre-existing inactive edges in the communication streams to generate new active edges by matching the inactive edges with arbitrary RHS elements of the grammar rules (other than the leftmost ones [*2]). The NLC-F rule provokes the existing active edge process to generate some new active edges by making the completion between inactive edges and arbitrary positions of undetermined parts of that active edge.

3・3 Parallel Extended Top-down Parser

The error recovery is to run a parallel extended top-down parser (P-ETD), exploiting the information (a set of the edges) generated by the P-LC-BU and P-NLC-BU. In this section, we first describe a way to recover the possible interpretations of an ill-

* 1 A brief form of the notation defined by Mellish [Mellish 89].
* 2 The leftmost ones have been already matched during the P-LC-BU.

formed input in top-down fashion (top-down search) and then provide a method to construct a parallel parser. The top-down search starts with the assumption that all words in the input are finally covered by the start symbol (e.g., sentence). To illustrate this, the following data structure is introduced for representing each state during the search. This notation is analogous to the one used in [Kato 91].

$$\langle \text{hole}: N \ \text{err}: M \ [(S_1, E_1, CatList_1), \cdots \cdots, (S_k, E_k, CatList_k)] \rangle$$

where N is the total number of categories in $CatList_1 \cdots CatList_k$; M is the number of errors detected before reaching this state; $S_1, E_1, \cdots, S_k, E_k$ are positions in the chart; $CatList_i$ is a set of categories needed between S_i and E_i.

The initial searching state is $\langle$hole : 1 err : 0 [(0, n, [S])]$\rangle$, where S is the start symbol (goal category) and n is the final position in the input. The extended top-down parser applies three special rules (garbage rule, empty category rule and unknown word rule), in addition to two rules*3 of the original top-down parsing, to handle three kinds of primitive errors (extra word error, omitted word error, and unknown/substituted word error). The description of the rules is illustrated in Fig. 2. These rules are applied to refine the state during the search. In the refinement process, the parser may reach a state $\langle$hole : 0 err : Err [ ]$\rangle$. By this time, one possible interpretation of the ill-formed input can be obtained. In our implementation, a threshold is set to limit the searching space ($N+M \leq Threshold$).

〔1〕 An Example

To illustrate the extended top-down parsing, let us consider a simple CFG and an ill-formed input, '*Jumbo is ap elephant*', as shown in Fig. 3. The inactive edges (1)~(5) and the active edges (6)~(8) are generated by the P-LC-BU. The active edges (9)~(10) are generated by the P-NLC-BU. Top-down searching (parsing) process begins with the state $\langle$hole : 1 err : 0 [(0, 4, [S])]$\rangle$. First, the *Active Edge Fundamental* rule (defined in Fig. 2) refines the initial state by applying the active edge (6). Then, by using the same rule, the resultant state $\langle$hole : 1 err : 0 [(1, 4, [VP])]$\rangle$ is refined by applying the active edges (7) and (8). From this step, two possible states will be derived. These two states are later refined again by the active edges (10) and (9), respectively. Finally, both of the possibilities are refined by *Unknown Word* rule to the final states, $\langle$hole : 0 err : 1 [ ]$\rangle$. The error detected is that the word '*ap*' is an unknown word which has either preposition (p) or determiner (det) as its category. This searching progress is shown in the right half of Fig. 3.

(CS = Current State, GR = Grammar Rule,
IE = Inactive Edge, AE = Active Edge,
NS = New generated State)

Top Down Activation Rule :
CS : $< \text{hole}{:}N \ \text{err}{:}M \ [(s_1,e_1,[C_1,C_2,\ldots]),\ldots \ldots,(s_n,e_n,Cs_n)] >$
GR: $C_1 \rightarrow RHS$
NS : $< \text{hole}{:}(N+(\text{length of RHS})-1) \ \text{err}{:}M \ [(s_1,e_1,[\text{RHS},C_2,\ldots]),\ldots,(s_n,e_n,Cs_n)] >$

Active Edge Fundamental Rule :
CS : $< \text{hole}{:}N \ \text{err}{:}M \ [(s_1,e_1,[C_1,C_2,\ldots]),\ldots] >$
AE: $\{s',E,C_1,[(S_1,E_1,Cs_1),\ldots,(S_n,E_n,Cs_n)]\}$, here, $s'=s_1$ or $s'=*$
NS : $< \text{hole}{:}(N+\Sigma(\text{length of } Cs_i)-1) \ \text{err}{:}M \ [(S_1,E_1,Cs_1),\ldots,(S_n,E_n,Cs_n),(E,e_1,[C_2,\ldots]),\ldots] >$

Garbage Rule :
CS : $< \text{hole}{:}N \ \text{err}{:}M \ [(s_1,e_1,[C_1,C_2,\ldots]),\ldots] >$
IE : $\{S_1,E_1,C_1,[\,]\}$, where C_1 is a lexical category
NS : $< \text{hole}{:}(N-1) \ \text{err}{:}(M+(S_1-s_1)) \ [(E_1,e_1,[C_2,\ldots]),\ldots] >$

Empty Category Rule :
CS : $< \text{hole}{:}N \ \text{err}{:}M \ [(s,s,Cs_1),(s_2,e_2,Cs_2),\ldots] >$
NS : $< \text{hole}{:}(N-(\text{length of } Cs_1)) \ \text{err}{:}(M+(\text{length of} Cs_1)) \ [(s_2,e_2,Cs_2),\ldots] >$

Unknown Word Rule :
CS : $< \text{hole}{:}N \ \text{err}{:}M \ [(s_1,e_1,[C_1,C_2,\ldots]),\ldots] >$
If No Edge : $\{s_1,s_1+1,C_1,[\,]\}$, where C_1 is a lexical category
NS : $< \text{hole}{:}(N-1) \ \text{err}{:}(M+1) \ [(s_1+1,e_1,[C_2,\ldots]),\ldots] >$

Fig. 2 Rules in P-ETD.

〔2〕 Parallelization of Top-down Parser

The top-down parser functions as a resolution of a tree searching problem. Parallelization of this parser corresponds to the way of distributing nodes (subtasks) in the searching tree among the processors. The subtasks are mutually independent of

*3 Top-down rule and fundamental rule.

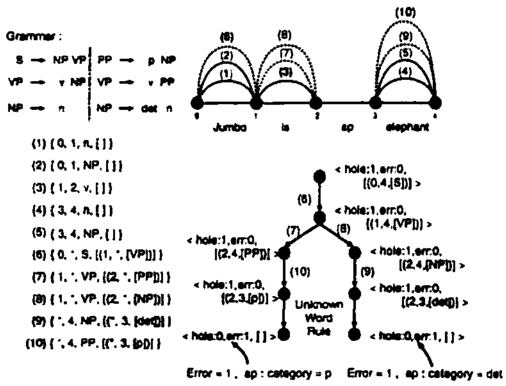

Fig. 3 An example of the extended top-down parsing.

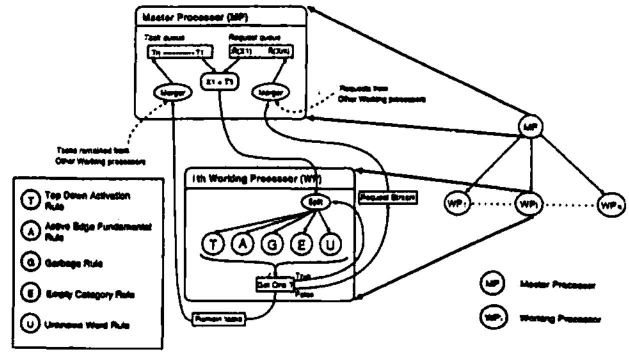

Fig. 4 On-demand load balancing scheme in P-ETD.

each other and equivalent to the refinement of a searching state under the five rules defined in Fig. 2.

In certain tasks (such as the bottom-up parsing) where communication patterns and dependency of tasks can be estimated before the execution, the best load balancing method can be decided statistically (static load balancing). However, a top-down search is not such a case since the size of a task (the number of subtasks) can not be determined unless the search is finished. Another aspect of the top-down search is that each subtask can be executed independently without exchanging information with other subtasks. According to these properties, the appropriate method for the top-down search would be the on-demand dynamic load balancing. Its basic technique is to dispatch a request for another task to other processors, if a processor has no task to execute in the next step.

Fig. 4 illustrates an overview of our load balancing method. From a set of processors, a master processor (MP) is selected to control load balancing by using a task queue and a request queue. Other processors, called the *working processors* (WPs), occupy the five basic routines corresponding to the rules defined in Fig. 2. Initially, all inactive and active edges are transferred to all WPs. Each WP dispatches a request for a task to the MP, where the requests are kept in the request queue. By utilizing data flow synchronous mechanism, when there are some tasks in the task queue of the MP, the task in the top of the queue will match the request in the top of the request queue. Then, the task is automatically transferred to the WP, which possesses that request. After receiving the task from the MP, the WP performs five routines (defined by the rules in Fig. 2). As the result of these routines, some new states (tasks) may be generated. One of the new states (a task) is executed in the processor while the remainders are transferred back to the MP and restored in the task queue. Unless there is any new state generated, the current WP will dispatch a request to the MP for a task in the next step. This request is kept in the request queue of the MP until it matches the task in the task queue. This procedure occurs recursively until no task remains (the task queue is empty and no more tasks exist in WPs). In this load balancing method, more parallelism is gained when the search is spread out, though the system may gain a little parallelism at the beginning of the search due to small number of states (nodes) at the top level of the searching space.

4. Experimental Results and Discussion

Our parallel parser is implemented on PIM/m, a loosely-coupled MIMD parallel processor with 256 processor elements (PEs). The efficiency of the parser was investigated by using the grammar with 393 CFG rules (the same as in [Tomita 87]). We can evaluate the parallel parser through three experiments. The first experiment is carried out to investigate the effectiveness of the parser when a grammatical input is analyzed. The second one is conducted to measure the ratio of computation time between *edge-generation* phase (P-LC-BU+P-NLC-BU) and *error recovery* phase (P-ETD). The last experiment is to investigate the effectiveness of the parser in the case of ill-formed inputs.

In case of grammatical inputs, the P-LC-BU succeeds and thus the P-NLC-BU and P-ETD will never get start. As described in **sec. 3•1**, we implement two versions of the P-LC-BU, in which the static and dynamic load balancing methods are applied. The dynamic version is to assign each active edge to a processor, while the static version is to distribute all active edges starting at i to the i th processor. Consequently, the number of processors used for the former version is 256 and it is correspondent to the length of the input for the latter version. The result speed-ups are shown in Fig. 5, where the graph plots the length of inputs versus the true speed-up (the speed-up relative to the serial version of the parser). The experimental inputs are shown in **Appendix A**.

The graph in Fig. 5 shows that the static version is superior to the dynamic version in all lengths (2~30 words) of grammatical inputs. Though the dynamic version seems to be the finer-grained algorithm, it faces the problem of much interprocessor communication, and therefore the overhead caused by the communication is a dominant factor during parsing. On the other hand, the static version has relatively less communication overhead. Therefore, we select the static version for the P-LC-BU of our parallel parser. It should be noted that the P-LC-BU can not gain large speed-up (around an increase of 20%) due to the high dependency of partitioned subtasks.

For the ill-formed inputs, we consider the following types of errors: extra-known-word (E-K), extra-unknown-word (E-U), substituted-known-word (S-K), substituted-unknown-word (S-U) and omitted-word (O-W). In addition to the P-LC-BU, the P-NLC-BU and P-ETD are activated to find all possible interpretations. In order to examine the source of parallelism between *edge-generation* phase (P-LC-BU+P-NLC-BU) and *error recovery* phase (P-ETD) in our parser, we conduct the second experiment to measure the computation time of these phases by using a single processor. The principle behind this experiment is that if a phase takes more time, the parallelization of that phase will have more influence in the total analytical time.

This experiment is carried out for single-error ill-formed inputs with different lengths (2~18 words) (cf. **Appendix B**). For each length, five sentences corresponding to the five types of errors are considered. The graph in Fig. 6 plots the length of inputs versus the ratio of the computation time for *edge-generation* and *error recovery* phases. The results indicate that this ratio tends to increase along with the length of inputs. In other words, for the longer ill-formed input, the computation time of the *error recovery* phase becomes more dominant than that of the *edge-generation* phase. This indicates that the parallelism of the *edge-generation* phase may influence the speed-ups of shorter inputs, while the parallelism of the error recovery phase may influence the speed-ups of longer inputs.

In the error recovery phase (P-ETD), the parallelism gained depends on the number of states

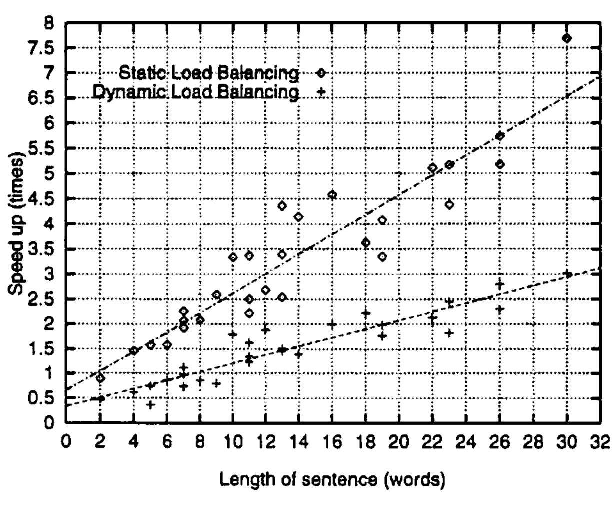

Fig. 5 Speed-up rate gained when inputs are grammatical (P-LC-BU only).

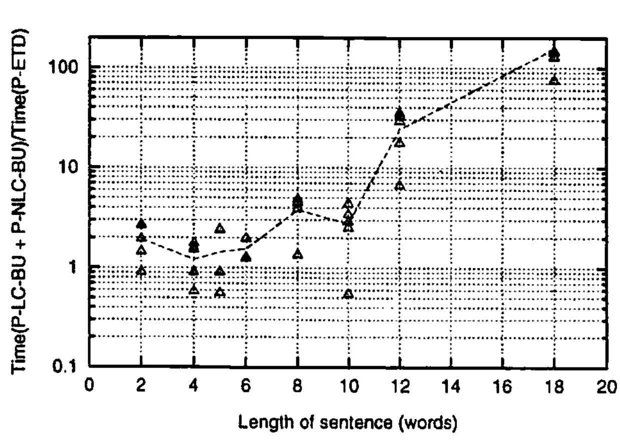

Fig. 6 The computation time ratio between *edge-generation* phase and *error recovery* phase, using single processor.

(nodes) in the searching space. In general, when the number of states becomes larger, we can expect a larger amount of parallelism. The number of states increases in accord with the length of the input and the number of errors because of the increasing ambiguity by the errors' positions.

We carry out the last experiment to investigate the effectiveness of the parallel parser for ill-formed inputs. In this experiment, we use short inputs (the original sentence's length is 7) with one/two errors and long inputs (the original sentence's length is 18) with one error (cf. **Appendix C**). The remaining three figures, Fig. 7, Fig. 8, and Fig. 9, show the resulting speed-ups for the short inputs with one error, two errors, and the long inputs with one error, respectively. All graphs in these figures plot the number of processors versus the speed-up.

In each figure, the legend (e.g., 1, E-K [932/2187] (83.5 → 6.0 sec) in Fig. 7) denotes the number and type of errors, followed by the number of edges and states in the form of [Number-of-edges/Number-of-states] and finally followed by the computation time for single processor and for 256 processors. In principle, the computation time of the edge-generation phase is proportional to the number of edges, whereas the computation time of the error recovery phase is proportional to the number of states in the searching space.

For all the inputs, the computation time and speed-up gained come out in the following order : E-K, E-U, S-K, S-U and O-W. In Fig. 7, we find out that the number of edges is larger than that of states. This means the edge-generation phase has a significant effect on the speed-ups of the total analysis in the case of short inputs with a single error. In this case, only small speed-ups were obtained. However, in E-K type, the error recovery phase seems to have more influence on the analysis time (due to the large number of states compared with the number of edges) and a relatively larger speed-up is gained. This indicates that the error recovery phase may gain a lot of parallelism.

The result of short inputs with two errors is shown in Fig. 8. In this case, the error recovery phase seems to have more effect on the speed-ups. In comparison with the case of single error, the case

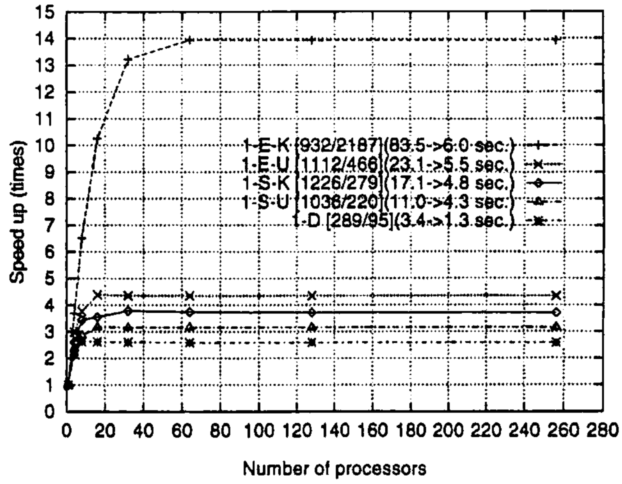

Fig. 7 Speed-up for the analysis of short ill-formed inputs with a single error (original sentence length=7).

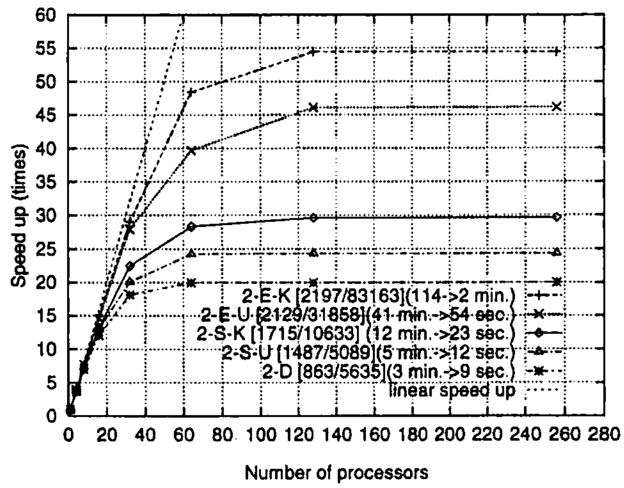

Fig. 8 Speed-up for the analysis of short ill-formed inputs with two errors (original sentence length=7).

of two errors has a larger number of states, and hence, there will be more opportunity for parallelism and more speed-ups gained for every type of error.

Fig. 9 indicates the result of long sentences with a single error. In this case, the parallelism is nearly gained from the error recovery phase (large number of states but small number of edges as shown in Fig. 9). The computation time in this case is remarkably larger than the previous two cases of short inputs for all types of errors, and conspicuously larger speed-ups are gained. According to the above-mentioned results, we notice that the speed-ups depend on the ratio of the communication and computation time in the error recovery phase (P-ETD), especially in the cases of two errors and long inputs, where the error recovery phase is the main analysis.

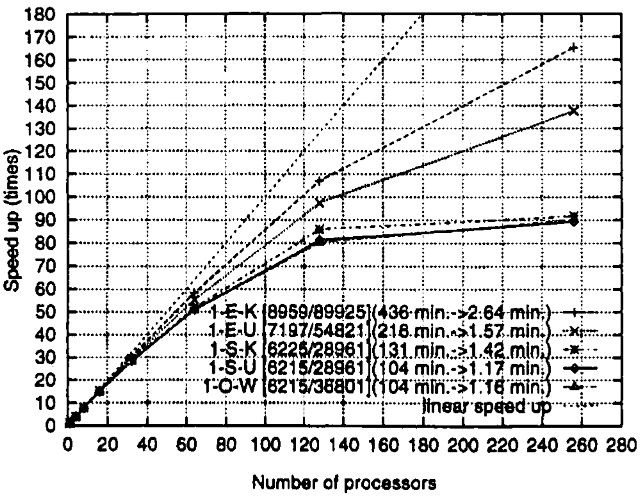

Fig. 9 Speed-up for the analysis of long ill-formed inputs with a single error (original sentence length=18).

In this phase, the interprocessor communication time is proportional to the number of states (tasks), while the computation time is proportional to the number of edges.

In comparison with the case of short inputs with two errors (Fig. 8), the number of states in searching space is larger in the case of long inputs (Fig. 9). For long inputs, both communication and computation time is larger. However, when we focus on one state, the computation time is larger (owing to the larger number of edges) but the communication time is not different (one state transferred). This makes the case of long inputs gain more parallelism and speed-ups. In addition, for the case of short inputs with a single error (Fig. 7), the number of states (tasks) is small, so the total idle time of processors becomes obviously longer and, thus, only a little parallelism is obtained.

The above experimental result shows that our parser can obtain a lot of gain when an ill-formed input is analyzed, especially in the cases of two errors and long inputs. However, we expect that our parallel parser can achieve more parallelism and gain in case of longer inputs with multiple errors*4.

As a limitation, our parallel parser still cannot parse ill-formed inputs in real time. Based on the above results, the parser takes 1~2 minutes to analyze the 18-word inputs with a single error. We expect that this problem can be solved by using faster networks and processors. However, we have shown through the experiments that the introduction of parallel processing to the task of parsing ill-formed inputs can succeed in parsing speed-up.

5. Conclusion

This paper proposes a parallel parsing method for analyzing ill-formed input under loosely-coupled hardware environment. Several pre-existing studies in parsing grammatical inputs indicate that the introduction of parallel execution provides only minimal advantages and none of those studies deals with the parallel performance of the ill-formed inputs. By our proposed method, we show that parallel parsing of ill-formed input can be improved to a satisfactory level of speed-up. The method, based on chart parsing algorithm, is composed of parallel bottom-up parsing (edge-generation phase) and parallel top-down parsing (error recovery phase). The bottom-up parsing generates the partial parses of the ill-formed input, while the top-down parsing exploits these partial parses to find and fix the existing errors and generates possible interpretations of the input.

We construct two parallel versions for the bottom-up parsing, in which static and dynamic load balancing methods are applied. Through a preliminary experiment, the static version seems to be more effective than the dynamic one, since it has less communication cost. The top-down parsing resembles a tree-searching framework, where on-demand dynamic load balancing seems to be more suitable. The parallel parser was implemented and tested for its efficiency on PIM/m, a loosely-coupled system. Based on several experimental results with 256 processors, the execution time of our parser is 2~14 times faster than the serial version in the case of short single-error inputs and up to 60~170 times faster in the cases of short two-error inputs and long single-error inputs.

In our present research, the parser finds all interpretations of inputs. However, the following issues should be considered for future research: (a) how to choose the best interpretation, (b) how to cut off the

* 4 Due to hardware and software limitations, we can not conduct this experiment.

distinctly useless interpretations, and (c) how to control the parallel parser to accommodate both (a) and (b).

Acknowledgement

The authors are grateful to our colleagues and friends for their assistance and support throughout our research duration. Without them, our research would have never been accomplished. First of all, we wish to thank Dr. K. Fuchi and Dr. S. Uchida, the previous and current ICOT directors, for allowing us to get access to Parallel Inference Machine (PIM) and offering us an ample opportunity to conduct this research. K. Susaki is especially helpful in setting up the machine and in providing us her constructive comments. We are indebted to M. Koshimura for his technical assistance during the implementation of our parser. We also would like to thank T. Tokunaga and our lab's members for their useful comments and suggestions. W. Udomkitwanit offers us all his skillful assistance in preparing many experiments. K. G. Suresh and P. Kitiarsa help us to proofread a draft of this paper.

Finally, we especially appreciate all referees for reading an earlier version of this paper. Without their critiques and tolerance, this paper would have never been possible. Although our colleagues and friends provide us their assistance and support, errors and misinterpretations appearing in this paper are solely our own responsibility.

◇ References ◇

[Grishman 88] Grishman, R. and Chitrao, M.: Evaluation of a Parallel Chart Parser, *2nd Conf. on Applied Natural Language Processing*, pp. 71-76, Austin, Texas, 1988, Association for Computer Linguistics (1988).

[Kato 91] Kato, T.: Yet Another Chart-based Technique for Parsing Ill-formed Input, *Natural Language Processing*, pp. 83-100, Information Processing Society of Japan, in Japanese (1991).

[Kwasny 81] Kwasny, S. and Sondheimer, N.: Relaxation Techniques for Parsing Grammatically Ill-formed Input in Natural Language Understanding Systems, *Am. J. Computational Linguistics*, Vol. 7, No. 2, pp. 99-108 (1981).

[Matsumoto 87] Matsumoto, Y.: A Parallel Parsing System for Natural Language Analysis, *New Generation Computering*, Vol. 5, No. 1, pp. 63-78 (1987).

[Meknavin 93] Meknavin, S., Theeramunkong, T. and Tanaka, H.: Parsing Ill-Formed Input with ID/IP rules, *4th Int. Workshop on Natural Language Understanding and Logic Programming (NLULP4)*, pp. 158-171 (1993).

[Mellish 89] Mellish, C.: Some Chart-based Techniques for Parsing Ill-formed Input, *Proc. 27th Annual Meeting of the ACL*, pp. 102-109 (1989).

[Nitta 92] Nitta, K., Taki, K. and Ichiyoshi, N.: Experimental Parallel Inference Software, *Int. Conf. on fifth Generation Computer Systems*, Vol. 1, pp. 166-190 (1992).

[Saito 88] Saito, H. and Tomita, M.: Parsing Noisy Sentences, *COLING-88*, No. 2, pp. 561-565 (1988).

[Susaki 89] Susaki, K., Sato, H., Sugimura, R., Akasaka, K., Taki, K., Yamazaki, S. and Hirota, H.: Implementation and Evaluation of parallel syntax analyzer PAX on the Multi-PSI, *Proc. Joint Parallel Processing Symposium (JSPP'89)*, pp. 342-350, In Japanese (1989).

[Taki 92a] Taki, K.: A Parallel Cooperation Model for Natural Language Processing, *Int. Conf. on fifth Generation Computer Systems*, Vol. 1, pp. 405-413 (1992).

[Taki 92b] Taki, K.: Parallel Inference Machine PIM, *Int. Conf. on fifth Generation Computer Systems*, Vol. 1, pp. 50-72 (1992).

[Thompson 80] Thompson, B.: Linguistic Analysis of Natural Language Communication with Computers, *Proc. 8th Int. Conf. on Computational Linguistics*, pp. 190-201, Tokyo, Japan (1980).

[Thompson 89] Thompson, H.: Chart Parsing for Loosely Coupled Parallel Systems, *Int. Workshop on Parsing Technologies*, pp. 320-328, Carnegie Mellon, Pittsburgh, PA (1989).

[Tomita 86] Tomita, M.: An Efficient Word Lattice Parsing Algorithm for Continuous Speech Recognition, *IEEE-IECEJ-ASJ Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP86)*, Tokyo (April, 1986).

[Tomita 87] Tomita, M.: An Efficient Augmented-Context-Free Parsing Algorithm, *Computational Linguistics*, Vol. 13, No. 1-2, pp. 31-46 (1987).

[Ueda 86] Ueda, K.: Guarded Horn Clauses, Wada, E. (ed.): *Logic Programming '85*, Lecture Notes in Computer Science 221, pp. 168-179, Springer-Verlag (1986).

[Ueda 90] Ueda, K. and Chikayama, T.: Design of the Kernel Language for the Parallel Inference Machine, *Computer Journal*, Vol. 33, No. 6, pp. 494-500 (1990).

[Weischedel 83] Weischedel, R. and Sondheimer, N.: Meta-rules as a Basis for Processing Ill-Formed Input, *Am. J. Computational Linguistics*, Vol. 9, No. 3-4, pp. 161-177 (1983).

〔担当編集委員：平川秀樹，査読者：木村和広〕

◇ Appendix ◇

A. Experiment No. 1

1. Do it .
2. I have a pen .
3. I must not do that .
4. Time flies like an arrow .
5. This is a book about language .
6. This is a book about human language .
7. It consists of symbolic commands called statements .
8. Its approach is motivated by two questions .
9. We look at language from a different perspective .
10. What knowledge must a person have to speak language .
11. It allows the use of symbolic codes to represent instructions .
12. It allows the use of symbolic codes to represent the instructions .
13. A program written in assembly language is called a source program .
14. In writing this book , I had several purposes in mind .
15. A model for estimating performance under heavy loads is included for completeness .
16. It includes exercises designed to help the student master a body of techniques .
17. It is a reference source with many pointers into the literature of linguistics .
18. How is the mind organized to make use of this knowledge in communicating .
19. How is the mind organised to make the use of this knowledge in communicating .
20. Each statement is written on a single line in computer and it consist of four entries .
21. Ethernet is a broadcast communication system for carrying digital data packets among computing stations and it is distributed .
22. Switching of packets to their destinations on the 'Ether' is distributed among the receiving stations using packet address recognition .
23. I have attempted to introduce a wide variety of material to provide newcomers with broad access to the field .
24. The source program is processed by the assembler to obtain a machine language program that can be executed directly by the 'CPU' .
25. Our study of the mental processes involved in language draws heavily on concepts that have been developed in the area called artificial intelligence .
26. In performing a mental task like deciding on a chess move , we are aware of going through a sequence of thought process .
27. Labels can be assigned to a particular instruction step in a source program to identify that step as an entry point for use in subsequent instructions .
28. The next chapter sets the computational approach into the context of other approaches to language by giving a brief history of the major directions in linguistics .
29. It is safe to say that much of the work in computer science has been pragmatic , based on a desire to produce computer programs that can perform useful tasks .

B. Experiment No. 2

Error Type	Sentence
Original	Do it .
O-W	_ It .
E-K	Do it a .
E-U	Do it xxxx .
S-K	It it .
S-U	Do xxxx .
Original	I have a pen .
O-W	I have a _.
E-K	I have must a pen .
E-U	I have xxxx a pen .
S-K	I about a pen .
S-U	I xxxx a pen .
Original	I have a heavy pen .
O-W	I ___ a heavy pen .
E-K	I have must a heavy pen .
E-U	I have xxxx a heavy pen .
S-K	I about a heavy pen .
S-U	I xxxx a heavy pen .
Original	This is a book about language .
O-W	This is a book about ___.
E-K	This is a book must about language .
E-U	This is a book xxxx about language .
S-K	This is a book must language .
S-U	This is a book xxxx language .
Original	We look at language from a different perspective .
O-W	We look at ___ from a different perspective .
E-K	We look at language from a different a perspective .
E-U	We look at language from a xxxx different perspective .
S-K	We look at language from not different perspective .
S-U	We look at language from xxxx different perspective .
Original	The single computer look at language from a different perspective .
O-W	The single computer look at ___ from a different perspective .
E-K	The single computer look at language from a different a perspective .
E-U	The single computer look at language from a xxxx different perspective .
S-K	The single computer look at language from not different perspective .
S-U	The single computer xxxx at language from a different perspective .
Original	A model for estimating performance under heavy loads is included for completeness .
O-W	A model for estimating performance ___ heavy loads is included for completeness .
E-K	A model for estimating performance that under heavy loads is included for completeness .
E-U	A model for estimating performance xxxx under heavy loads is included for completeness .
S-K	A model for estimating performance is heavy loads is included for completeness .
S-U	A model for estimating performance xx heavy loads is included for completeness .
Original	Ethernet is a broadcast communication system for carrying digital data packets among computing stations and it is distributed .
O-W	Ethernet is a broadcast communication system for carrying digital data packets among computing ___ and it is distributed .
E-K	Ethernet is a broadcast communication system for carrying digital data packets among computing are stations and it is distributed .
E-U	Ethernet is a broadcast communication system for carrying digital data packets among computing xxx stations and it is distributed .
S-K	Ethernet is a broadcast communication system for carrying digital data packets among computing the and it is distributed .
S-U	Ethernet is a broadcast communication system for carrying digital data packets among computing xxx and it is distributed .

C. Experiment No. 3

Original	it consists of symbolic commands called statements
O-W	it ___ of symbolic commands called statements
E-K	it consists of of symbolic commands called statements
E-U	it xxx consists of symbolic commands called statements
S-K	at consists of symbolic commands called statements
S-U	xx consists of symbolic commands called statements
Original	it consists of symbolic commands called statements
O-W	it ___ of symbolic commands ___statements
E-K	it consists the of symbolic commands called statements the
E-U	it consists xxx of symbolic commands called statements yyy
S-K	at consists of symbolic commands called of
S-U	xxx consists of symbolic commands called yyy
Original	ethernet is a broadcast communication system for carrying digital data packets among computing stations and it is distributed
O-W	ethernet is a broadcast communication system for carrying digital data packets among computing ___ and it is distributed
E-K	ethernet is a broadcast communication system for carrying digital data packets among computing are stations and it is distributed
E-U	ethernet is a broadcast communication system for carrying digital data packets among computing xxx stations and it is distributed
S-K	ethernet is a broadcast communication system for carrying digital data packets among computing the and it is distributed
S-U	ethernet is a broadcast communication system for carrying digital data packets among computing xxx and it is distributed

Authors' Profile

Thanaruk Theeramunkong (Student Member)

He born in 1967. He received B.E., M.E. and Ph.D. from Tokyo Institute of Technology in 1990, 1992 and 1995 respectively. Now he works as a research assistant at Japan Advanced Institufe of Science and Technology, Hokuriku. His research interests are in natural language processing, machine learning, information retrieval and knowledge science. He is a member of Japanese Society for Artificial Intelligence and the Information Processing Society of Japan.

Hozumi Tanaka (Member)

He received the B. S., and M. S. degrees in faculty of science and engineering from Tokyo Institute of Technology in 1964 and 1966 respectively. In 1966 he joined in the Electro Technical Laboratories, Tsukuba. He received his Doctor of Engineering in 1980. In 1983 he joined as an associate professor in the faculty of Information Engineering in Tokyo Institute of Technology and he became professor in 1986. He has been engaged in Artificial Intelligence and Natural Language Processing research. He is a member of the Information Processing Society of Japan. etc.