社団法人 人工知能学会
Japanese Society for
Artificial Intelligence

人工知能学会研究会資料
JSAI Technical Report
SIG-PPAI-9303-1 (3/9)

A Parallel Chart-based Parser for Analysing Ill-formed Inputs

Thanaruk Theeramunkong   Hozumi Tanaka
Department of Computer Science,
Tokyo Institute of Technology
2-12-1, O-okayama, Meguro-ku, Tokyo 152, Japan
Tel : +81-3-3726-1111 (Ext. 4188)
Fax : +81-3-3720-4925
e-mail: ping@cs.titech.ac.jp

Abstract

This paper describes a parallel parsing method for analysing ill-formed inputs with consideration of full syntactic context. The method is developed to handle ill-formed inputs using tree searching scheme which is flexible to utilize any kind of task distribution to gain parallelism. We choose dynamic task distribution as a core mechanism to control distributing tasks during the parsing process. Our parser is implemented on a parallel inference machine, named PIM[1]. The result of the experiments demonstrates that using 256 processors, our parser is 60-170 times faster than a serial version in the case of long ill-formed inputs with multiple errors.

1 Introduction

In decades, there have been many attempts to develop parallel algorithms for time-consuming tasks in several areas. Especially, for natural language area, several works tried to introduce both shared-memory and loosely-coupled parallel machines to improve the speed of parsing grammatical inputs such as in [4][7][11]. However, in real situation (e.g., daily conversation), natural language is frequently used ungrammatically. Several reports showed that people usually use their own language ungrammatically, for instance, in Thompson's extensive study [1] with a database query system, 33% of inputs that users made, was unparsable due to vocabulary problem, punctuation errors, ungrammaticality and spelling errors. Owing to these phenomena, a practical system should have potential to deal with such ill-formedness instead of rejecting the analysis.

Many early works on parsing ill-formed inputs were based on different formalisms, such as ATNs[8][9], chart parsing[2][10], GLR[3]. Among these formalisms, left-right parsing like ATNs or GLR, deals with ill-formedness in two different means: (1) trying to analyse the input from left to right and then performing a special mechanism to apply some types of extra-condition rules to allow making progress in the parsing state at the interrupted point (2) hypothesizing some errors at each stages of parsing and then processing all of them. However, the systems of type (1) might fail to indicate minimal error due to the left-right bias which cannot consider right context to handle ill-formedness, and ones of type (2) faces with the problem that parsing grammatical inputs may be slowed down.

Recently, Mellish[2] introduced a new chart-based technique in which both left and right syntactic contexts could play a role to determine the best interpretation of ill-formedness. However, Mellish also pointed out that his method lacks the control for avoiding duplication of effort to recover ill-formedness, and remains to be seen how the performance will scale up for a real grammar and parser. Generally, to parse ill-formed input, some particular mechanisms to detect the cause of ill-formedness and correct it, have to be embedded to treat the ill-formedness. This causes an ill-formed input to take much more time to be parsed than a grammatical input. There are several previous works showing the advantage of parallel processing on parsing grammatical inputs, however it could be expected that more advantage is gained when ill-formed inputs are parsed.

This paper focuses on parsing ill-formed input using loosely-coupled parallel environment. The parsing algorithm proposed in this paper stands on the following objectives: (1) the consideration of full syntactic context during parsing (2) the approach firstly tries to parse the input according to the given grammar (a context free grammar) and then starts the recovery process when the input cannot be recognized as a grammatical one. This approach will cause the system not to be slowed down in any way when a grammatical input is parsed.

This paper describes a technique to construct a parsing method by using tree searching scheme. This scheme is flexible to allow any kind of task distribution to place on the parsing method to gain parallelism. We choose dynamic task distribution as a core mechanism to control distributing tasks among processors. We also report here a series of experiments to determine that parsing grammatically ill-formed input with the consideration of the full syntactic context can be successfully sped up on loosely-coupled environment (PIM) with 256 processing elements(PEs). The result shows our parser could per-

[1] A loosely-coupled system developed under the Fifth Generation Computer Project (ICOT) in Japan and the language used on the machine is a concurrent logic programming language named KL1

form 60-170 times faster than the serial version in the case of long inputs with multiple errors. Finally, the limitations on our method are also discussed.

2 Parsing Ill-formed Input Considering Full Context

Mellish [2] showed how to combine the advantages of bottom-up chart parser and top-down chart parser to deal with ill-formed input. The basic algorithm is to run firstly bottom-up parser with no top down filtering and if it fails, then to execute top-down parser that recovers existing errors exploiting the edges generated during bottom-up parsing and constructs the interpretations of the input. The advantages of this strategy are: (1) the recovery process (top-down parsing) is executed after bottom-up parsing fails to find a complete parse of the input, without causing existing work to be repeated or affecting the original parsing (bottom-up parsing) to be slowed down in any way, (2) it allows the full syntactic context to be considered to detect ill-formedness in the input.

Thereafter, Kato [10] proposed a modified version of Mellish's method. Kato introduced an intermediate step between bottom-up and top-down parser, named *edge completion phase* which exploited more edges generated in bottom-up fashion to reduce searching space in top-down parsing.

However, these methods have an important problem in the computation time when it is scaled up to be used in realistic situation; especially, when a large grammar is used or when a long ill-formed input with multiple errors is analysed.

3 Our Parallel Parsing Algorithm

In this section, we describe a parallel chart-based parsing algorithm that can inherit the advantages of Mellish's method[2] and Kato's method[10]. The chart parsing paradigm has properties suitable for parsing natural language in both views of ill-formed analysis and parallelisation. In particular, it has two excellent following properties:

- It occupies a book-keeping mechanism that the results of partial analysis of an input will be kept in the form of edges (the 'chart'). When the parser fails to analyze the input(due to ill-formedness), this mechanism allows the possibility of utilizing the 'chart' already created, during the ill-formed analysis.
- It allows the analysis to be occurred at any word in the input. That is, unlike the left-right parsing where the analysis is occurred in the order from left to right, it is possible for the parsing process to be done individually for each word of the input in parallel.

Our parallel parser is composed of three constituents: (1) parallel bottom-up parser (P-BU), (2) parallel extended top-down parser (P-ETD), (3) parallel edge completion process (P-EC), which is the intermediate process between P-BU and P-ETD. P-BU finds all complete parses of the input. If there is no possible interpretation of the input, the input is recognized to be ill-formed. Then, P-EC starts generating all edges that were blocked by P-BU due to its left-right characteristic. At the same time, P-ETD runs to find all possible interpretations of ill-formed input exploiting the information (in the form of a set of edges) generated by P-BU and P-EC. Our parallel parser handles three types of errors: extra (unknown/known) word error, omitted word error, substituted (unknown/known) word error.

3.1 Parallel Bottom Up Parser

P-BU is a parallel version of conventional bottom-up chart parsing without top-down filtering. The parser tries to construct complete parses of the input, and even if the input is ill-formed, edges generated by this parser can be utilized later in the recovery process to avoid duplication of existing work. We have explored the approach of distributing the chart among the processors in several implementations on a loosely-coupled system, named PIM. As the result, we found out that the way to distribute the chart among the processors on a vertex[2] by vertex basis, appeared to be the better way than the others.

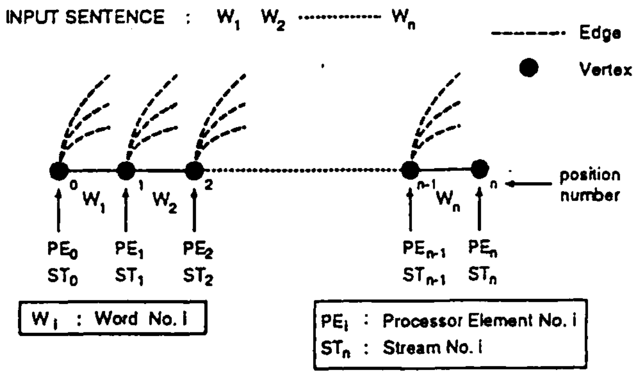

Figure 1: Processor distribution

Figure 1 shows the assignment of processors and streams to vertices. Here, each vertex is assigned a processor and a stream. The communication between processors is done through stream communication. In chart parsing, the basic actions are extending the edge and creating the edge and the two data structures are active edges and inactive edges. Let the sets of active edges and inactive edges which have i as their starting positions, be denoted by $AEDGE_i$ and $IEDGE_i$ respectively. PE_i[3] executes the two actions which are related to $AEDGE_i$ and $IEDGE_i$. More precisely, each edge in $AEDGE_i$ is a process in PE_i while $IEDGE_i$ is a set of the data which the processes of PE_i produce and it is put into the stream. Each process in PE_i consume the data in streams at every j th vertex.(where, $j > i$).

In this approach, there is only one message producer per each stream while there are multiple consumers that consume the messages(edges) and use them to extend the edges. Thus communication cost can be dramatically reduced. With our several experiments on PIM, the result indicated that the approach did well on PIM's architecture, while the early work done by Henry[4] showed that

[2] A vertex is a position in the input, as shown in Figure 1.
[3] i th processor element

the approach gained no speed-up in the Intel Hypercube architecture due to slow network but fast processors. This indicates that the achievement of the approach also depends on the system architecture.

In this bottom-up parsing, one possible alternative decomposition of the parsing task would be to distribute active edges into different processors. These active edges are processes that consume streams of inactive edges and makes progress in the parsing state. However, in this approach, the communication between processors becomes large. According to our experiments, the result showed that there was no speed-up gained by this approach.

3.2 Parallel Edge Completion Process

P-EC, the intermediate process, relaxes the restriction of left-right order of bottom-up parser and generates some active edges. For instance, toward the grammar ($C \to C_0, C_1, C_2$), an active edge ($C \to C_0[C_1]C_2$) is generated by this process if C_1 is found, and ($C \to [C_0], C_1, [C_2]$) is generated if there is an active edge ($C \to [C_0], C_1, C_2$) and C_2 is found. The category in parenthesis means that it has already been analysed. These edges are helpful for hypothesizing errors during parsing ill-formed inputs and also useful to reduce searching space in P-ETD. P-EC runs after P-BU had determined that the input was ill-formed. Each processor of P-EC generates some other active edges which are not generated before by P-BU by relaxing left-right restriction. To denote what is done in each processor, the following notation is introduced to represent an edge:

$$\{SP, EP, Cat, Unparsed\}$$

where, SP is an integer number or a variable (denoted by '$*$') specifying the starting position of the edge in the sentence, EP is an integer number or a variable specifying the ending position of the edge in the sentence, Cat is the category of the edge which is being parsed, $Unparsed$ is the unparsed part of the edge. Unlike items in Early's algorithm [5], in this notation, parsing state is represented by two elements, Cat and $Unparsed$, instead of a dotted rule. Cat is LHS of the rule while $Unparsed$ is a list showing the unparsed part of RHS. Each element of $Unparsed$ is denoted by (S_i,E_i,$CatList_i$), where $CatList_i$ is a list of categories; S_i,E_i are positions in the input(sentence); and (S_i,E_i,$CatList_i$) means $CatList_i$ is needed between S_i and E_i. Here, an inactive edge occupies a null list as its $Unparsed$. The following rules show what is done in processor i.

Processor i :

- Modified Bottom-up Rule:
  If there is an edge, $\{i,j,C_1,[\,]\}$ and a rule,
  $C \to ...Cs_1,C_1,Cs_2...$ (..Cs_1 is not empty)
  then generate an edge,
  $\{*,E_2,C,[(*,i,...Cs_1),(j,E_2,Cs_2)]\}$. Here, if Cs_2 is empty then $E_2=j$ otherwise $E_2=*$.
- Modified Fundamental Rule:
  If there is an active edge,
  $\{S,E,C,[...,(s_1,e_1,[...,Cs_1,C_1,Cs_2])]\}$
  in the processor and an inactive edge
  $\{S_1,E_1,C_1,[\,]\}$ in other processors,
  then generate an active edge,
  $\{S,E,C,[...,(s_1,S_1,Cs_1),(E_1,e_1,Cs2)]\}$.
  Here, $s_1 \le S_1$ or $s_1=*$ and $E_1 \le e_1$ or $e_1=*$

3.3 Parallel Extended Top Down Parser

The backbone of ill-formedness recovery is parallel augmented top down parser(P-ETD). This parser exploits the information(the set of edges) generated by P-BU and P-EC, and tries to find the interpretation of the ill-formed input in top down style by starting from the assumption that the input is a sentence.

3.3.1 The Algorithm

We introduce a strategy to reconstruct top down parsing as tree searching process exploiting the information (the set of edges) generated by P-BU and P-EC. This strategy has the advantage that it allows existing parallelisation methods for any tree-searching problem to be applied directly in the parsing algorithm. To illustrate our augmented top down parsing in detail, we utilize the following notation for each searching state:

$$< \text{hole:}N\ \text{err:}M\ [(S_1, E_1, CatList_1), \ldots$$
$$\ldots, (S_k, E_k, CatList_k)] >$$

where N is the total number of categories in $CatList_1 \ldots CatList_k$; M is the number of errors detected before reaching this state.

The initial searching state is < hole:1 err:0 [(0,n,[S])] >, where n is the final position in the input sentence (sentence length). One possible interpretation of the ill-formed input is found when the parser reaches the state, < hole:0 err:Err [ ] >. P-ETD occupies five following objects to find errors in the input. The first two objects, *Top Down* and *Active Edge Fundamental*, are for refining the unsatisfied portions[4], and the remaining three objects are for determining 3 kinds of primitive errors: extra (known/unknown) word error, omitted word error and substituted (known/unknown) word error.

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

Top Down Rule Activation Object :

CS : < hole:N err:M [(s_1,e_1,[C_1,C_2, ...]), ...
...,(s_n,e_n,Cs_n)] >
GR: $C_1 \to RHS$
NS : < hole:(N+(length of RHS)-1) err:M
[(s_1,e_1,[RHS,C_2, ...]), ...,(s_n,e_n,Cs_n)] >

Active Edge Fundamental Object :

CS : < hole:N err:M [(s_1,e_1,[C_1,C_2, ...]), ...] >
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 : < hole:(N+Σ(length of Cs_i)-1) err:M
[(S_1,E_1,Cs_1), ..., (S_n,E_n,Cs_n),
(E,e_1,[C_2, ...]), ...] >

Extra Word Finder Object :

CS : < hole:N err:M [(s_1,e_1,[C_1,C_2, ...]), ...] >
IE : $\{S_1,E_1,C_1,[\,]\}$
NS : < hole:(N-1) err:(M+(S_1-s_1))
[(E_1,e_1,[C_2,...]), ...] >

[4]elements within the parenthesis [...] of a searching state

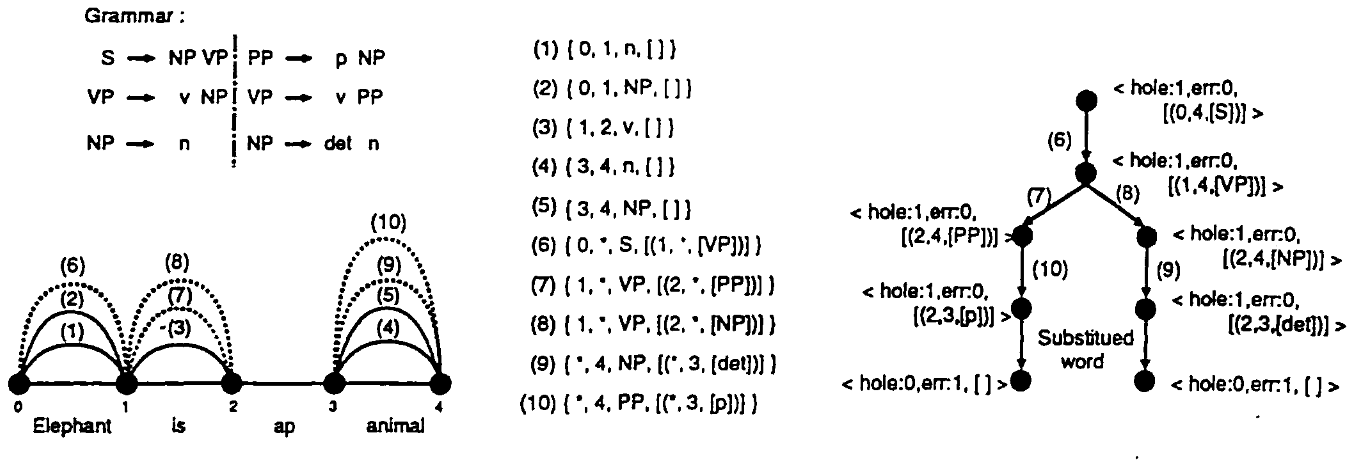
Grammar :
S → NP VP
PP → p NP
VP → v NP
VP → v PP
NP → n
NP → det n
(1) { 0, 1, n, [ ] }
(2) { 0, 1, NP, [ ] }
(3) { 1, 2, v, [ ] }
(4) { 3, 4, n, [ ] }
(5) { 3, 4, NP, [ ] }
(6) { 0, *, S, [(1, *, [VP])] }
(7) { 1, *, VP, [(2, *, [PP])] }
(8) { 1, *, VP, [(2, *, [NP])] }
(9) { *, 4, NP, [(*, 3, [det])] }
(10) { *, 4, PP, [(*, 3, [p])] }
Elephant
is
ap
animal
< hole:1,err:0, [(0,4,[S])] >
< hole:1,err:0, [(1,4,[VP])] >
< hole:1,err:0, [(2,4,[PP])] >
< hole:1,err:0, [(2,4,[NP])] >
< hole:1,err:0, [(2,3,[p])] >
< hole:1,err:0, [(2,3,[det])] >
< hole:0,err:1, [ ] >
< hole:0,err:1, [ ] >
Substitued word

Figure 2: An example of top down parsing

Empty Category Finder Object :

CS : < hole:N err:M [(s,s,Cs_1),(s_2,e_2,Cs_2), ...] >
NS : < hole:(N-(length of Cs_1))
err:(M+(length ofCs_1)) [(s_2,e_2,Cs_2), ...] >

Substituted Word Finder Object :

CS : < hole:N err:M [(s_1,e_1,[C_1,C_2, ...]), ...] >
If No Edge : {s_1,s_1+1,C_1,[ ]},
where C_1 is a terminal category
NS : < hole:(N-1) err:(M+1) [(s_1+1,e_1,[C_2, ...]), ...] >

3.3.2 An Example

To illustrate top down parsing, let's consider an example grammar and an ill-formed input, '*Elephant is ap animal*' shown in Figure 2. Inactive edges (1)-(5) and active edges (6)-(8) are generated by P-BU. Active edges (9)-(10) are generated by P-EC. Top down parsing corresponds to a searching process starting from the initial state, < *hole*:1 *err*:0 [(0,4,[S])] >, using the existing edges. Firstly, *Active Edge Fundamental* object refines the initial state by applying edge (6). In the next step, with the same object, the result state, < *hole*:1 *err*:0 [(1,4,[VP])] > is refined by applying edge (7) and edge (8). Note that there are conflicts occurred at this point and it is possible to process them individually in parallel. Finally, both of possibilities are refined by *Substituted Word* object to the final state, < hole:0 err:1 [ ] > and the error detected is that the word '*ap*' is an unknown word with either preposition or determiner as its category.

3.3.3 Dynamic Task Distribution

In this section, we describe how to parallelize the searching process efficiently in loosely-coupled systems. Since there is no general task distribution that works well in every task, it is necessary to construct a suitable task distribution for this framework. In certain tasks where communication patterns and dependency of tasks can be estimated before the execution, the best task distribution can be decided statistically. However, in this framework, it is not such a case and we consider that on-demand dynamic task distribution is a good method to balance this parsing task. The idea is that when each processor finishes the current task, it sends a demand for another task to do at the next step.

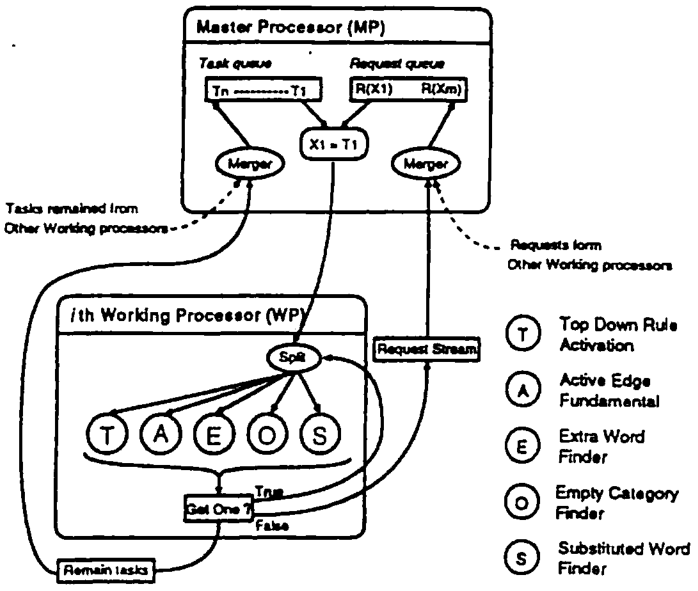

Figure 3: Task distribution scheme

Figure 3 shows the scheme of our task distribution. A master processor(MP) is established to control the task distribution by using a task queue. Other processors, called the *working processors*(WPs), have the five objects (as shown earlier in this section) as their basis routines. Each of WPs gets a task from the master processor and performs these five routines. 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 remainder of them is transferred back to the master processor and restored in the queue. If there is no new state generated, the current WP has no work to do in the next step and then it sends a request to the master processor to get another task(state) and then a task in the top of the queue of the master processor is distributed to that processor. This procedure occurs recursively until the queue is empty. In our implementation, the master processor also executes searching but the priority of performing task distribution is higher than the priority of executing searching.

4 Experimental Results and Discussion

The parser was implemented on Parallel Inference Machine(PIM). The efficiency of the parser was investigated in the environment of : (1) 256PEs multiprocessors (2) the grammar containing 393 CFG rules as described in [6] (3) the ill-formed inputs with the length ranging from 7-18 words. In all cases, the graphs plot the number of processors vs. the true speed-up (the speed-up relative to the serial version of the parser).

The tested inputs are of the following types: one and two extra-known-word(E-K), extra-unknown-word(E-U), substituted-known-word(S-K), substituted-unknown-word(S-U) and omitted-word(D). Figure 4 and 5 show speed-up rates of 7-word sentence with one error and two errors, respectively. Figure 6 shows speed-up rate of 18-word sentence with one error. In each figure, the key denotes the type of errors, following by the number of edges and states in the form of [Number-of-edges/Number-of-states] and finally following by computation time when utilizing one processor and 256 processors.

Figure 4 shows the speed-up results of a short sentence with one error. Among five types of errors, the extra-known-word case gains the highest speed-up since the number of states (tasks) is much larger and then much greater opportunity for parallelism is offered. Compared with extra-word cases, substituted-word cases gain less speed-up due to a less number of tasks. Figure 5 shows the result of a short sentence with two errors. The order of speed-up rates for five types of errors is the same as in the case of short sentence with one error. However, this case gains much more speed-up rates in every type of error than one error case. Figure 6 indicates the result of a long sentence with one error. In this case, computation time for every type of errors is larger than the two cases of a short sentence. However, the highest speed-up is obtained in this case due to the largest number of tasks and possibility for more parallelism.

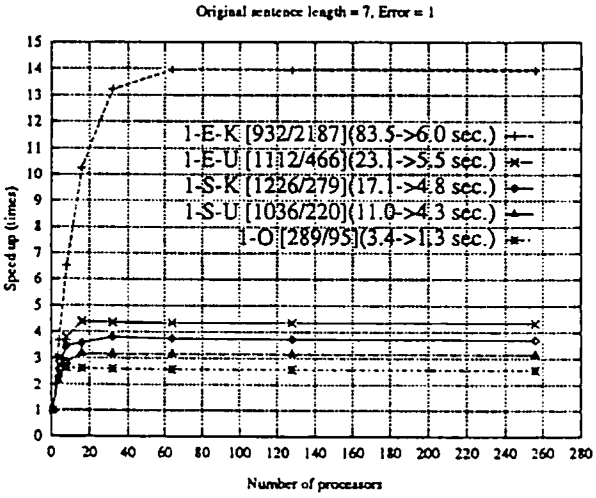

Figure 4: Speed-up for the analysis of an ill-formed input with one error (original sentence length=7)

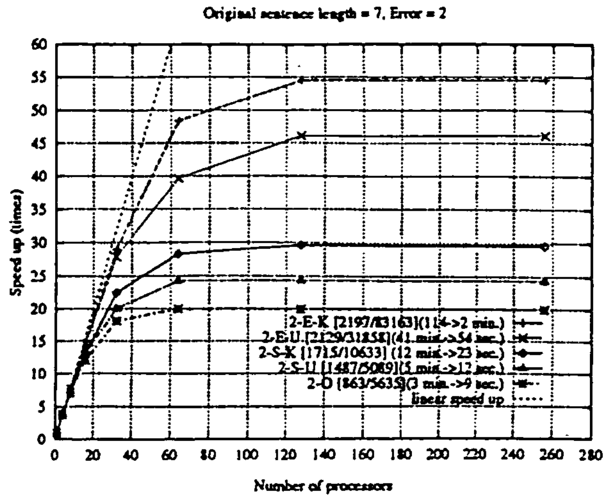

Figure 5: Speed-up for the analysis of an ill-formed input with two errors (original sentence length=7)

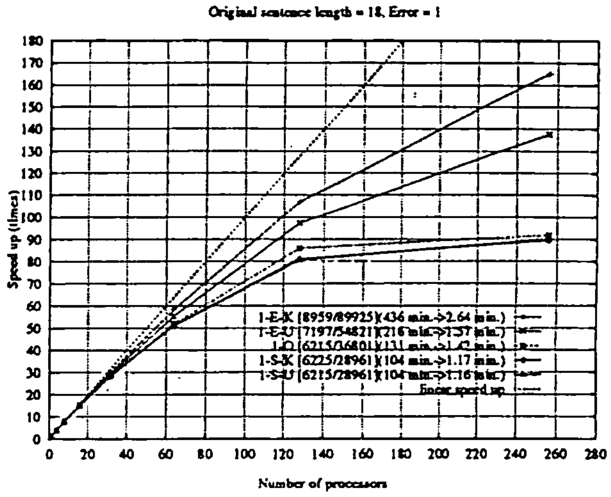

Figure 6: Speed-up for the analysis of an ill-formed input with one error (original sentence length=18)

According to the results mentioned above, we can observe that speed-up rate depends on the ratio of communication and computation time. Here, communication time is proportional to the number of states(tasks) while computation time is proportional to the number of edges. In the case of multiple error (Figure 5) or long sentence (Figure 6), the number of states in searching space is large and both communication and computation time is also large. However, when we focus on one state, we can observe that computation time is large (owing to the large number of edges) while communication time is constant(one state transferred). In addition, in the case of short sentence with one error, the number of states(tasks) is small, so the total idle time of processors becomes obviously long and then less parallelism is obtained.

Our experimental result convinces that the introduction of parallel approach to parsing ill-formed input with the consideration for full context is promising. Especially, in the case of a long sentence with multiple errors, we expect our parallel parser will gain tremendous (almost linear) speed-up in parsing time.

We also make a series of experiments for checking our parallel parsing when the input is grammatical. In this case, P-BU succeeds in analysing the input and thus P-EC and P-ETD never gets start. In P-BU, the number of used processors corresponds to the length of each input. The tested grammatical inputs had the length ranging from 2-30 words. The speed-up results are shown in Figure 7. In the figure, it indicates that more speed-up is gained when the analysed sentence is longer and it grows up with an increase of 10 %. Though the speed-up aim of parallel processing is that the analysis time when utilizing n processors should be n times faster than one of single processor, this aim cannot be overcome in the framework of parsing due to the serial operations of this framework and communication cost.

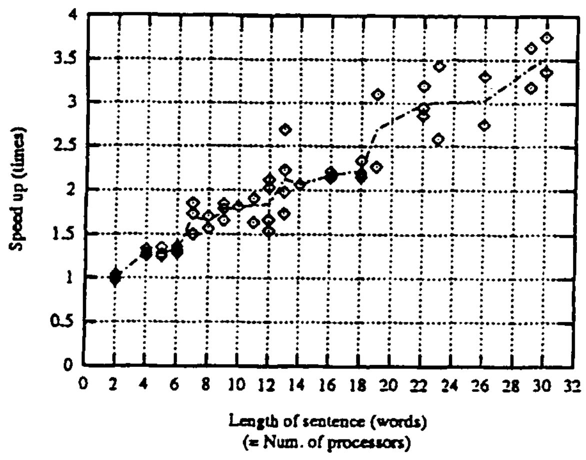

Figure 7: Speed-up rate gained when inputs are grammatical and their lengths are between 2 and 30 words

5 Conclusion

This paper proposes a new parallel parsing method for analysing ill-formed input with consideration of full syntactic context. Our parser generates all complete parse trees for a grammatical input and all possible interpretations of an ill-formed input. We introduced a method to reconstruct the ill-formedness recovery process (top down parsing) into a tree-searching framework and utilized dynamic task distribution to acquire parallelism. The parser was implemented and tested its efficiency on PIM, a loosely-coupled system. According to the results of several experiments, we found out that our parser acquired promising result in its performance. It ran 2-14 times faster than the serial version in the case of short ill-formed inputs and up to 60-170 times faster in the case of long ill-formed inputs or inputs with multiple errors.

However, more explorations in the efficiency of the parser have to be done with several other task distribution schemes. Currently, our parser discovers all interpretations of the input, however we are on the way to consider some issues concerned with (a) how to choose the best interpretation among them (b) how to cut off useless interpretations, and (c) how to control the parallel parser to accommodate (a) and (b).

Acknowledgement

We would like to thank the ICOT Director, Dr. K.Fuchi, for providing access to Parallel Inference Machine(PIM) and giving us the opportunity to conduct this research. We would like to thank K.Susaki who helped us in setting the machine and also gave us many useful comments. We also would like to thank M.Koshimura who helped me in some useful techniques during constructing the system. Finally, we would like to thank T.Tokunaga and our lab's members who gave us many useful comments.

References

[1] B.H. Thompson. Linguistic analysis of natural language communication with computers. In *Proc. of the 8th International Conference on Computational Linguistics*, pp. 190–201, Tokyo, Japan, 1980.

[2] C.S. Mellish. Some chart-based techniques for parsing ill-formed input. In *Proceeding of 27th Annual Meeting of the ACL*, pp. 102–109, 1989.

[3] H. Saito and M. Tomita. Parsing noisy sentences. In *COLING(2)*, pp. 561–565, 1988.

[4] Henry S. Thompson. Chart parsing for loosely coupled parallel systems. In *International Workshop on Parsing Technologies*, pp. 320–328, Carnegie Mellon, Pittsburgh,PA, 1989.

[5] J. Early. An efficient context-free parsing algorithm. *Comm. ACM*, Vol. 13, No. 2, pp. 94–102, 1970.

[6] M. Tomita. An efficient augmented-context-free parsing algorithm. *Computational Linguistics*, Vol. 13, No. 1-2, pp. 31–46, 1987.

[7] R. Grishman and M. Chitrao. Evaluation of a parallel chart parser. In *Second Conference on Applied Natural Language Processing*, pp. 71–76, Austin, Texas, 1988. Association for Computer Linguistics.

[8] Ralph M. Weischedel and Norman K. Sondheimer. Meta-rules as a basis for processing ill-formed input. *American Journal of Computational Linguistics*, Vol. 9, No. 3-4, pp. 161–177, 1983.

[9] S.C. Kwasny and N.K. Sondheimer. Relaxation techniques for parsing grammatically ill-formed input in natural language understanding systems. *American Journal of Computational Linguistics*, Vol. 7, No. 2, pp. 99–108, April-June 1981.

[10] T. Kato. Yet another chart-based technique for parsing ill-formed input. In *Natural Language Processing*, pp. 83–100. Information Processing Society of Japan, 1991. in Japanese.

[11] Y. Matsumoto. A parallel parsing system for natural language analysis. *New Generation Computering*, Vol. 5, No. 1, pp. 63–78, 1987.