Analysing Ill-formed Inputs with Parallel Chart-based Techniques

20－5

Thanaruk Theeramunkong*
Hozumi Tanaka

Dept. of Computer Science, Tokyo Institute of Technology

In this paper, we describe a parallel implementation of chart-based parser which performs robustness to analyse grammatical ill-formed inputs in loosely-coupled environment. The parser is composed of parallel bottom-up process which parses the inputs under the grammar rules and parallel top-down process which tries to recover the existing ill-formedness when the bottom-up process fails to find a complete parse. The top-down process resembles a resolution of tree searching problem, utilizing the intermediate parsing information generated in the bottom-up process. We propose a method to use a 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. Using 256 processors, it ran 60-170 times faster than a serial version in the case of long ill-formed inputs with multiple errors.

1 Introduction

Recently, 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 PAX, parallel chart parsers, PLR. However, several reports showed that people usually use their own language ungrammatically.

Among the previous works on parsing ill-formed inputs, Mellish[2] introduced a chart-based parsing method in which both left and right syntactic contexts could play a role to determine the best interpretation of ill-formedness. The method is superior to other early researches in the following points: (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. Then, this approach will cause the system not to be slowed down in any way when a grammatical input is parsed. However, Mellish's 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 as described in [2]. Moreover, in order to parse an 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.

This paper proposes a parallel parser based on Mellish's chart-based techniques in the loosely-coupled environment. The parser is composed of parallel bottom-up process which parses the input under the grammar rules and parallel top-down process which tries to recover the existing ill-formedness when the bottom-up process fails to find a complete parse. The recovery process(top-down) resembles a resolution of tree searching problem, utilizing the intermediate parsing information generated in the bottom-up process. We introduce on-demand dynamic task distribution as a core mechanism to control distributing tasks during the parsing process. We implement the parallel parser in a loosely-coupled parallel machine named PIM. We also showed some experimental results of our parser's efficiency. With 256 processors the parallel parser could perform 60-170 times faster than the serial version in the case of long inputs or inputs with multiple errors.

2 Chart-based Parsers for Ill-formed Input Analysis

In [2], Mellish shows how to combine the advantages of bottom-up and top-down chart parsers for dealing ill-formed input. The basis algorithm is to run bottom-up parser with no top down filtering, over the input and if it fails, then to execute top-down parser to find existing errors and construct the interpretations of the input.

*Dept. of Computer Science, Tokyo Institute of Technology, 2-12-1 Ookayama Meguro-ku Tokyo 152 Tel: (03)3734-2837 Fax: (03)3734-2915

The advantages of this strategy are: (1) that the recovery process (top-down parsing) executes after a standard parser (bottom-up parsing) fails to find a complete parse of the input (when it is ill-formed), without causing existing work to be repeated and the standard parser to be slowed down in any way (2) that it allows the full syntactic context to be considered in the interpretation of ill-formed input. Thereafter, Kato [1] proposed an improved version of Mellish's parsing method. Kato introduces the intermediate step between bottom-up and top-down parser, named ***edge completion phase*** which exploits more edges generated in bottom up style by violating the constraint of left-right order to reduce searching space of top-down parser. Kato's method occupies a simple algorithm instead of using complicated searching controls which applied in [2], and also dissolves the duplication of effort to recover the ill-formedness.

3 The Parallel Parser

Our parallel parser is based on the serial algorithm proposed in [1]. The parser is composed of three constituents: parallel left corner bottom-up parser (P-LC-BU), parallel non-left corner bottom-up process (P-NLC-BU) and parallel extended top-down parser (P-ETD). P-LC-BU executes to find all complete parses of the input. If there is no possible interpretation of the input found, the input is recognized to be ill-formed and P-NLC-BU gets start to generate all edges that were blocked in P-LC-BU due to left-to-right characteristic of the parser and simultaneously P-ETD runs to find all of the possible interpretations of ill-formed input using the information (a set of edges) generated in P-LC-BU and P-NLC-BU. Our parallel parser can handle three types of errors: extra (unknown/known) word error, omitted word error and substituted (unknown/known) word error.

3.1 Parallel Bottom Up Parser

The P-LC-BU is a left corner bottom-up chart parser without top-down prediction. Among early works on parallel bottom-up parsing, the way to construct a left corner bottom-up chart parser on loosely-coupled system is firstly explored by Henry[3]. Henry tried to implement a parser on the Intel Hypercube architecture but his system could not take encouragement due to slow networks but fast processors. The parallelisation of chart parsing can be viewed as the way to distribute the chart among the processors. Following Henry's work, we explored the approach of distributing the chart in some implementations on PIM. Similar to Henry's system, the result shows that the way to distribute the chart among the processors on a vertex[1] by vertex basis, seemed to be the better way than the others. For instance, ith processor will hold a set of edges starting at i position in the sentence. By this method, once a new edge is delivered to its 'home' processor, that processor has all the edges required to execute the fundamental rule with respect to that new edge. Communication between processors is limited to be one-to-many type. Therefore, the communication cost decreases.

[1]A vertex is a position in the input

When the P-LC-BU cannot find out a complete parse of the input due to some ill-formedness, the P-NLC-BU relaxes the restriction of left-right order of P-LC-BU parser and generates some active edges. For instance, toward the grammar ($C \rightarrow C_0.C_1.C_2$), an active edge ($C \rightarrow C_0[C_1]C_2$) is generated by this process if C_1 is found, and ($C \rightarrow [C_0].C_1.[C_2]$) is generated if there is an active edge ($C \rightarrow [C_0].C_1.C_2$) and C_2 is found. The category in parenthesis means that the category has already been analysed. These edges are dramatically useful for hypothesizing errors during parsing ill-formed inputs and also helpful to reduce searching space in P-ETD parser. In the P-NLC-BU, the processors in the P-LC-BU are reactivated to generate more edges as described above.

3.2 Parallel Extended Top Down Parser

The backbone of ill-formedness recovery is parallel extended top down parser(P-ETD). This parser exploits the information(the set of edges) generated by P-LC-BU and P-NLC-BU, 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. The top-down parsing is viewed as a searching process using a set of active and inactive edges generated in P-LC-BU and P-NLC-BU. To illustrate this parser, we introduce the following notation for representing each searching state.

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

where $CatList_i$ is a list of categories; $S_1.E_1,..., S_k.E_k$ are positions in the input(sentence) : $(S_i.E_i.CatList_i)$ means $CatList_i$ is needed between S_i and E_i ; 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 ill-formed input is found when the parser reaches the state, $<$ hole:0 err:Err [ ] $>$. The restriction, $N + M \le Limit$, is used to limit searching space. P-ETD parser occupies five objects (shown in Fig.1) to find errors in the input. The first two objects, Top Down and Active Edge Fundamental, are for refining the unsatisfied portions, and the remaining three objects are for determining 3 kinds of primitive errors: extra word error, omitted word error and unknown/substituted word error:

3.2.1 An Example

To illustrate top down parsing, let's consider an example grammar with the ill-formed input, '*We bought ap car*' shown in Fig. 2. Inactive edges (1)-(5) and active edges (6)-(8) are edges generated in P-LC-BU parser. Active edges (9)-(10) are edges generated in P-NLC-BU process. Top down parsing corresponds to a searching process starting from the initial state, $<$ *hole*:1 *err*:0 [(0.4.[S])] $>$, using existing edges. Firstly, ***Active edge fundamental*** object refines the initial state by applying the edge (6). Moreover, with the same object, the result state, $<$ *hole*:1 *err*:0 [(1,4.[VP])] $>$ is refined by

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

Top Down Activation Rule :

CS : < hole:N 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 : < hole:(N+(length of RHS)-1) err:M
$[(s_1,e_1,[\mathrm{RHS},C_2,\ldots]),\ldots,(s_n,e_n,Cs_n)]$ >

Active Edge Fundamental Rule :

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

Extra Word Finder Rule :

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

Empty Category Finder Rule :

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

Substituted Word Finder Rule :

CS : < hole:N 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 terminal category
NS : < hole:(N-1) err:(M+1) $[(s_1+1,e_1,[C_2,\ldots]),$
$\ldots]$ >

Fig. 1: Five Rules in P-ETD

applying the edges (7) and (8). Note that there are conflicts occurred at this point. Finally, both of possibilities are refined by *Substitute Word Finder* object to the final state, < hole:0 err:*Err* [ ] > and then the word 'ap' is detected to be an unknown word with either preposition or determiner as its category.

3.2.2 Dynamic Task Distribution

During top down parsing, a search tree, where each node represents a parsing state, is provided. Operations in the tree are five rule objects (defined in Fig.1). 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 some 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 that 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 in next step.

Fig. 3 shows the scheme of our task distribution. A master processor(MP) is established to control the task

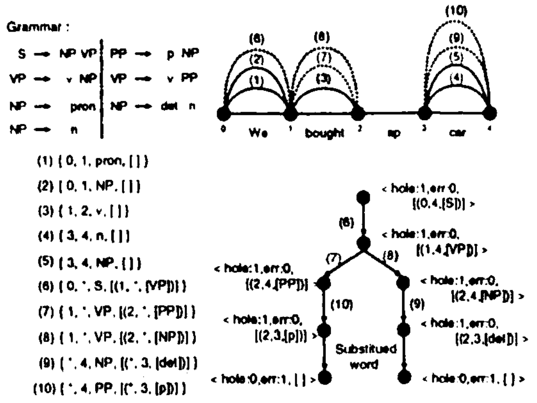
Grammar :
S → NP VP
VP → v NP
NP → pron
NP → n
PP → p NP
VP → v PP
NP → det n
(1) { 0, 1, pron, [ ] }
(2) { 0, 1, NP, [ ] }
(3) { 1, 2, v, [ ] }
(4) { 3, 4, n, [ ] }
(5) { 3, 4, NP, [ ] }
(6) { 0, *, S, [(1, *, [VP])] }
(7) { 1, *, VP, [(2, *, [PP])] }
(8) { 1, *, VP, [(2, *, [NP])] }
(9) { *, 4, NP, [(*, 3, [det])] }
(10) { *, 4, PP, [(*, 3, [p])] }
0
We
1
bought
2
ap
3
car
4
< 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

Fig. 2: An Example of Extended top down parsing

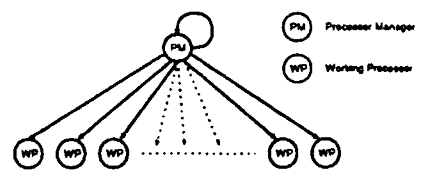

Figure 3: The scheme of task distribution

distribution by using a task queue. Other processors, called them *working processors*(WPs), have the five objects corresponding to rules in Fig.1 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), the processor 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. (3) the ill-formed inputs with the length ranging from 6-18 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/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). For the short tested input with one error, our parser ran 1-5 times faster than the serial version in average. As the case of multiple errors and long inputs, Fig. 4 and 5 show the speed-up rates of 6-word inputs with two errors and the speed-up rate of 18-word input 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.

Fig. 4 shows the speed-up results of a short sentence with two 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. This demonstrates that the extra-known-word-typed error is the most difficult to be parsed. Compared with extra-word cases, substituted-word cases gain less speed-up due to a less number of tasks. Fig. 5 indicates the result of a long sentence with one error. In this case, the computation time for every type of errors is larger than the case of the short sentence. However, the highest speed-up is obtained in this case due to the largest number of tasks and possibility for more parallelism.

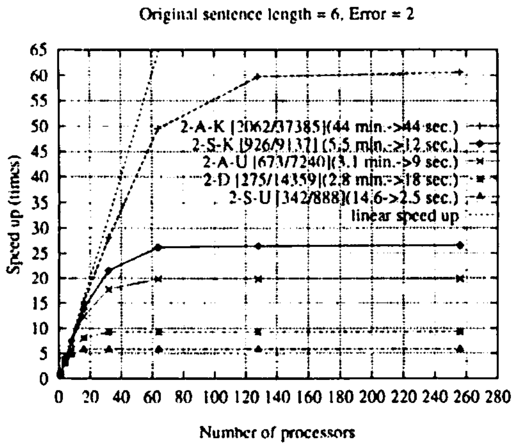

Fig. 4: Speed-up for the analysis of an ill-formed input with two errors (original sentence length=6)

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 (Fig. 4) or long sentence (Fig. 5), 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 the computation time is large (owing to the large number of edges) while the communication time is constant(one state transferred). In addition, in the case of the short sentence, 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.

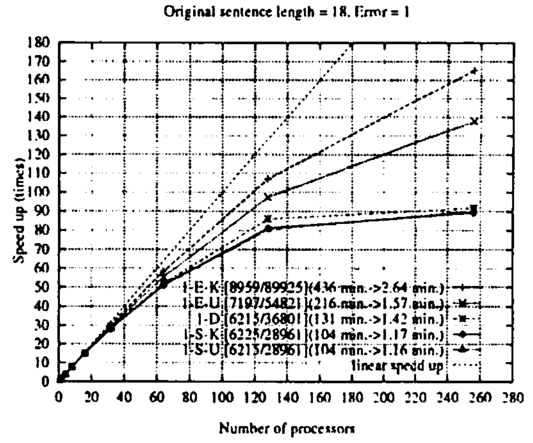

Fig. 5: Speed-up for the analysis of an ill-formed input with one error (original sentence length=18)

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 had a promising result in its performance. it ran up to 60-170 times faster in the case of long ill-formed inputs or inputs with multiple errors.

Currently, our parser discovers all interpretations of the input, however we are on the way to consider some issues concerned with how to choose the best interpretation among them, how to cut off useless interpretations, and how to control the parallel parser to firstly find out the appropriate solutions.

References

[1] 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.

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