PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT.

A Parallel Cooperation Model for Natural Language Processing

Shigeichiro Yamasaki, Michiko Turuta, Ikuko Nagasawa, Kenji Sugiyama

Fujitsu LTD.
1015, Kamikodanaka Nakahara-Ku, Kawasaki 211, Japan

Abstract

This paper describes the result of a study of a natural language processing tool called "Laputa", which is based on parallel processing. This study was done as a part of the 5th generation computer project. The purpose of this study is to develop a software technology which integrates every part of natural language processing: morphological analysis, syntactic analysis, semantic analysis and so on, to make the best use of the special features of the Parallel Inference Machine.

To accomplish this purpose, we propose a parallel cooperation model for natural language processing that is constructed from a common processor which performs every sub-process of natural language processing in the same way. As a framework for such a common processor, we adopt a type inference system of record-like type structures similar to Hassan Aït-Kaci's psi-term [Aït-Kaci 86], Gert Smolka's sorted feature logic [Smolka 88] or Yasukawa and Yokota's object-tem [Yasukawa 90].

We found that we can utilize parallel parsing algorithms and their speed-up technology to construct our type inference system, and we then built a type inference system using an algorithm similar to a context-free chart parser. As a result of experimentation to evaluate the performance of our system on Multi-PSI, the simulator of the Parallel Inference Machine, we have been able to achieve a speed-up of a factor 13 when utilizing 32 processors of Multi-PSI.

1 Introduction

With the advance of semiconductor technologies, computers can be made smaller and cheaper, so that we can increase the value of a computer by giving it multiprocessor abilities. However, the software application technology for a parallel machine is still at an unsatisfactory level except for some special cases.

The Parallel Inference Machine which is being developed in the 5th generation computer project has some special features such as an automatic synchronization mechanism and a logic programming language allowing declarative interpretations.

Such features make complicated parallel processing tasks, that used to be practically impossible, possible to realize. Knowledge Information Processing is one such application which needs lots of computational power and consists of very complicated problems, but we expect that the Parallel Inference Machine will make these problems amenable to parallel processing. The purpose of this study is to propose a parallel cooperation model which makes natural language processing more natural by making use of the parallel inference machine features.

In this paper, we will discuss the schema of the parallel cooperation model, as well as its realization and show the experiment results of the model capacity evaluations on the Multi-PSI, the simulator of the parallel inference machine.

2 Parallel cooperation model

The advantages of using the Parallel Inference Machine lie not only in the processing speed, but also in the problem solving techniques. We were able to find more natural ways of solving a problem by looking at it from the parallel processing point of view.

In recent years, system integration has often been suggested in the field of natural language processing. This involves the integration of morphological analysis, syntactic analysis, semantic analysis and speech recognition, and the integration of analysis and generation. The implication is that the various natural processing mechanisms at every stage must be linked to each other in order to understand natural language entirely. [Hishida 91]

As the basis of this way of thinking, it is emphasized that our information processing has been carried out under "partialness of information", in other words, incompleteness of information. From the above, we can derive that a system which aims at integrating natural language processing could adopt parallel processing because it disregards the processing sequence.

We adopted a mechanism which integrates all natural language analysis processing stages and makes them cooperate in parallel as the fundamental processing model.

Also we have added a priority process as an extension, in order to improve the processing efficiency. This priority process is the combination of both load balance and the parallel priority control. We call this process 'competition' and we call the extended parallel cooperation model the "model of cooperation and competition". However, we shall not discuss 'competition' in this paper.

3 Realization of automatic parallel cooperation

It is well known that the integration of natural language processing and parallel cooperation is a natural model. However, very few systems based on this model are reported to have been actually built. One of the main problems has been modularity.

Various research projects in natural language processing have been achieving good results in the fields of morphological analysis and syntactic analysis. However, these systems were designed as independent modules and very often their interfaces are very restricted and internal information is normally invisible from the outside. To carry out efficient parallel cooperation, all processes must be able to exchange all of their information with each other. Therefore construction of methods of information exchange between the various modules and the control of these exchanges will be serious and complicated problems.

One way to solve this problem is to make an abstraction of the processing framework, so that analysis phases such as morphological analysis, syntactic analysis etc. are carried out by one single processing mechanism. One such approach is Hashida's Constraint Transformation [Hasida 90]. We have adopted an approach similar to that of among others Hashida, in the sense that all levels of processing are carried out by one and the same processing mechanism. Our processing framework, however, does not utilize Constraint Transformation, but rather Type Inferencing with respect to record-like type structures, which is comparable to Hassan Ait-Kaci's LOGIN , Gert Smolka's Feature Logic, or the Object Terms in Yasukawa and Yokota's QUIXOTE.

In our system, the usage of type inferencing can be seen to have two aspects: it works as a framework for analysis processing as well as for cooperation. Analysis processing employs a vertical kind of type judgment, as exemplified by the cooperation between morphological analysis and syntactic analysis. In morphological analysis characters are considered to be objects, and morphemes are to be taken as types; but when we perform syntactic analysis, morphemes are considered to be objects.

The usage of type inferencing as a framework for cooperation, the second aspect of this usage mentioned above, is as a means for exchanging information between objects and types and for structuring the contents of this information. Here both objects and types are represented as typed record structures containing shared or common variables, and information exchange is implemented through the unification of shared variables in two typed record structures representing an object and a type.

This unification mechanism of typed record structures has a mechanism to judge the types of objects that are instantiated to field elements through communication, and this is what was called the vertical type judgment mechanism.

Parallel cooperation between syntactic and semantic processing is expressed through the unification mechanism of typed record structures.

Even if we treat all phases of natural language processing as similar in kind, it is still natural, for the sake of ease of grammar development and debugging, to do the development of the distinct processing phases separately. For this reason, we have structured our system so that concept organization rules for morphological, syntactic and semantic processing can be developed separately. Parallel cooperation is then realized automatically by merging these diverse rules and definitions.

4 The realization of parallel analysis processing

4.1 Type judgment mechanism

Efficient algorithms exist for morphological and syntactic processing, and we cannot afford to ignore such knowledge in developing a practical system, even in the case of an integrated natural language processing system. Luckily we have found that there is a strict correspondence between our vertical type judgment and known syntactic analysis methods. Matsumoto's parallel syntactic analysis system PAX [Matsumoto 86] performs syntactic processing in parallel through a method called the "layered stream method", which is an efficient processing mechanism for search problems involving parallel logic programming languages.

PAX employs what is basically a chart parsing algorithm. Our vertical type judgment processing formalism involves a reversal of the relationship between process and communication data in PAX. A syntactic analysis system using a similar processing method to ours is being considered by Icot's Taki [Sato 90]. Whereas PAX is strongly concerned with the clause indexing mechanism of logic programming languages, our method concentrates on increasing OR-parallellism and reducing the amount of data communication in parallel execution.

How we interpret phrase structure rules, using the type ordering relation "<" and type variables, is shown below.

```
s <- np,vp
```

This is rewritten based on the rightmost element as follows.

```
vp < (np -> s)
```

Here the ordering relation "<" expresses a superordinate-subordinate relationship between types. Intuitively this means that the object that is judged to be a subordinate type can also be judged to be a superordinate type. It follows that the meaning of this rule is that the object that can be judged to be the vp, can also be judged to be a function of type np to s.

```
s <- advp,np,vp
```

In a case like this one, we embed functions to produce the following.

```
vp < (np -> (advp -> s))
```

When there are several possibilities, this is expressed in a direct sum format as follows.

```
vp < (np -> ((advp -> s) + s))
```

The dictionary is a collection of type declarations as follows.

```
(in,the,end):advp
love:np
wins:vp
```

Analysis is executed as a process of type judgment of a word string. In other words, analysis is the execution of the judgment of a type assignment such as the one below.

```
(in,the,end,love,wins):s
```

The execution is bottom-up. First the type of every word is looked up among the type declarations. The words then send these type judgments to their right adjacent element. If these types again have superordinate types, then they are treated as follows.

If the superordinate type is a function, then a process is generated which checks the possibility that the typed object received from the left is appropriate. If it is not a function (in which case it is atomic), then this type judgment formula is sent to the right adjacent element, and also it is checked whether it has a superordinate type. When the result of a superordinate type or function appropriateness is a direct sum, then this result is handled in OR-parallel form. Repeating this kind of processing over and over, we get as answers all the combinations of elements from the leftmost to the rightmost that satisfy "s".

One of the special features of this processing formalism is that, when sending an object of atomic type, the pointer to the position of the leftmost of the elements that make up this object is sent along as the "exit of communication path". Hereby the partial tree that is constructed upon reception of this object in fact is capable of including all atomic-type objects that are structured to the left of the received object. If we translate this to structure sharing in sequential computation, we see that we can avoid unnecessarily repeating the same computation while retaining the computational efficiency of a chart parsing algorithm for context-free grammars.

Below we give the KL1 program for the fundamental part of vertical type judgment. Note however that the notation we have used above is transformed to KL1 notation, in the manner explained directly below.

```
direct sum
     type+,...,+type ==> [type,...,type]
type declaration
     object:type  ==> type(object,T) :-
                                 true| T=type.
type ordering
    type < type   ==> upper(type,T) :-
                                 true| T=type.
input format
  (in,the,end,love,wins):s
   ==>judgment([in,the,end,love,wins],s,R).
```

Note: R will contain the result of computation

Also we use "*" for the operator that constructs the pair of the sending atomic type and the stream, and the atom "Leftmost" as an identifier for the leftmost position of the input.

```
judgment(Objects,Type,Result) :- true|
        objects(Objects,'Leftmost',R),
        judged_as(R,Type,Result).

objects([],L,R) :- true|L=R.
objects([Word|Z],L,R) :- true|
        type(Word,Type),
        sum_type(Type,L,R1),
        objects(Z,[Word|R1],R).

sum_type([],L,R) :- true|L=R.
sum_type([Type -> Type2|Z],L,R) :- true|
        function_type(Type ->Type2,L,R1),
        sum_type(Z,L,R2),
        merge({R1,R2},R).
otherwise.
sum_type([Type|Z],L,R) :- true|
        atomic_type(Type,L,R1),
        sum_type(Z,L,R2),
        merge({R1,R2},R).

function_type(Type -> Type2,[],R) :-
        true|R=[].
function_type(Type -> Type2,'Leftmost',R) :-
        true|R=[].
function_type(Type-> Type2,[Type *L1|L],R) :-
```

```
        true|
        sum_type(Type2,L1,R1),
        function_type(Type -> Type2,L,R2),
        merge({R1,R2},R).
otherwise.
function_type(Type -> Type2,[_|L],R) :-
        true|
        function_type(Type -> Type2,L,R).

atomic_type(Type,L,R) :-
        true|
        upper(Type,Upper_Type),
        R=[Type*L|R1],
        sum_type(Upper_Type,L,R1).

judged_as([],Type,Result) :-
        true|Result=[].
judged_as([Type*'Leftmost'|L],Type,Result) :-
        true|
        Result=[Type|R],
        judged_as(L,Type,R).
otherwise.
judged_as([_|L],Type,Result) :- true|
        judged_as(L,Type,Result).

% Example of dictionary:

type(love,Type) :- true|
        Type=[np].
type(wins,Type) :- true|
        Type=[vp].
type(end,Type) :- true|
        Type=[the -> [in -> [advp]]].

% Example of grammar:

upper(vp,Upper_Type) :- true|
       Upper_Type=[np -> [advp -> [s], s]].
```

4.2 Unification mechanism of the record-like type structure

A record-like type structure is a pair of a sort symbol and a description. A sort symbol denotes the sort to which the type belongs. A description is a so-called record structure formed by pairs of feature names and their values. The feature value is also a description or an object. However a description is unlike an ordinary record structure in that its feature and value pairs are not always apparent. Indeed, the purpose of this structure is to obtain incremental precision from partial information, just like the feature structures used in unification grammar formalisms such as LFG.

In systems like Ait-Kaci's psi-term, Smolka's sorted feature structure or Yasukawa & Yokota's object term, the value of a feature is also a record-like type structure. However in our system, a value of a feature is not a record-like type structure but a description and only the terminal nodes of a feature structure tree are typed objects. This is to improve the efficiency of calculation.

In our system an object is represented as a pair of a record-like type structure and an identifier of the object. The value of a feature can be a variable. However the unification of descriptions involves merging feature structures rather than instantiating variables.

Variables of feature value play the role of a tag for the merging point in feature unification. In our system such variables also play the role of communication pass to exchange information for our parallel cooperation. Sometimes a variable can be assigned a type. When a variable is assigned a type such a typed variable must be instantiated by an object.

Below we give an example of a record-like type structure.

```
{human,[parents=[father={human,211,[name=taro]},
                 mother=X:{human,[]}]]}
```

This example shows a type which is sorted "human" and satisfies some constraints as a description. The description has a feature "parents" and the value of the feature is also a description that contains the feature of "father" and feature of "mother". The value of the feature "father" is an object that is of sort "human" and named "taro" and its object identifier is "211". The value of feature "mother" is a typed variable. The type of the variable is sorted "human" and its description has no information.

The unification mechanism for the record-like type structure is realized as the addition of information to the table of the pairs of the tag and the structure to which the tag referred. The unification process is the merging process to construct the details of the record-like type structure. When the typed variable is instantiated by an object, the type judgment process is invoked.

This is in concreto how our parallel cooperation mechanism for semantic analysis and syntactic analysis works.

4.2.1 Parallel cooperation and record-like type structure

A type can be seen as a program which can process an object. This implies that there is a close relation between merging of information using record-like type structures on the one hand, and the "living link" between objects or programs on the other. As an example, imagine that a graph object which was created by a spread sheet program is passed on to an object which is a word processor document. If we want such a graph object to be a "living" object, re-computable by the creator program, then it must be annotated by its creator as a data type in the record structure of the word processor document. Now

when the data is re-computed, the system will invoke its creator application program automatically. The type theory of record-like type structures can be viewed as a framework for this kind of cooperation of different application programs.

Laputa's principle of automatic parallel cooperation is a parallel version of this "live linking". Vertical type judgment of morphological analysis and syntactic analysis is an application program in this sense.

We can extend this live linking even further by using variables that are shared between objects and types, so that we can propagate information to objects within objects. For example, if a graph object from a spread sheet is pasted to a word processor document, and some of the data within the graph is shared with a part of the document text, then re-computation of the spread sheet program will happen when that part of the document text is modified. In our system, such re-computation is realized by communication of processes.

5 The grammar and lexicon for parallel cooperation

In this section we explain the syntax and description method for the grammar and lexicon for our system. Grammar rules and lexical items are described as a type definition or an ordinal relation of types. Our parallel processing mechanism treats morphological data and syntactic data in a uniform way. However it is not efficient to use exactly the same algorithm for morphological processing as for syntactic processing, because morphological processing examines only immediately adjacent items and therefore does not need context-free grammar. Our processing mechanism treats characters and morphemes in a slightly different way. For this reason characters and morphemes are distinguished as data types. Another, more essential reason for this is the problem posed by morphemes that consist of only one character. If there is no difference between a character and a morpheme, then our type judgement process will never be able to stop.

5.1 Some examples of grammatical and lexical description

5.1.1 Dynamic determination of semantic relation of subject and object

The semantic categories of subject and object are not determined only by the verb with which they belong. In many, if not most cases, the adequacy of the semantic category of the object changes according to the subject. Because of this, the required semantic categories of subject and object should not be fixed in the lexical description of the verb.

The example grammar rules below show how the adequacy of the semantic category of the grammatical object can vary dynamically depending on the subject.

```
{np,[sem=Ob]} < ({vt,VT} -> {vp,VT=[obj=Ob]})
{vp,VP} < ({np,[sem=Ag]} -> {s,VP=[agent=Ag]})
```

In this example the first grammar rule shows that the superordinate type of the type "np" is a function of type "$vt- > vp$". This rule means that an object which is judged as the type "np" is also a function which, if applied to an object of type "vt", results in an object of type "vp". In this rule every description of "vt" is merged to "vp" and the value of the feature "sem" of "np" is unified with the value of feature "obj" of the type "vp".

The next grammar rule means that an object of type "vp" is also a function of type "$np- > s$". This rule means that the value of the feature "sem" of subject "np" is unified with the value of the feature "agent" of "vp".

Now we also show some lexicon entries to go with these rules.

```
eats:{vt,[agent=Ag:{animal,[eat_obj=Ob]},obj=Ob]}
john:{np,[sem={human,Id,[name='John']}]}
the_tiger :{np,[sem={tiger,Id,[]}]}
```

In the lexicon the object "eats" has type "vt" and complex description. In the description the value of the feature "agent" is a typed variable and the type of the variable is sorted "animal" and the value of the feature of "eat-obj" is unified with the value of the feature of "obj" of type "vp".

The rules specifying semantic categories look as follows.

```
{tiger,[]} < {animal,[eat_obj=E:{animal,[]}]}
{human,[]} < {animal,[eat_obj=E:{food,[]}]}
```

These rules mean that a tiger is an animal which eats animals and a human is an animal which eats food, respectively.

Although under these rules of grammar, lexicon and semantic categories 'john' and 'the_tiger' are both animals, the judgment (the_tiger,eats,john):s succeeds but (john,eats,the_tiger):s fails, because John is a human and a human is an animal which eats food but a tiger cannot be judged as food from the rules governing semantic categories.

5.1.2 Subcategorization

The next example is the lexical entry for the Japanese verb "hanasu"(to speak). This verb is subcategorized by 3 "np"s which are marked for case by the particles "ga", "wo" and "ni".

```
hanasu:{vp,[subcat=Case:{ga,wo,ni},
            predicate=[ga=[gram_rel=subj,
                           sem=G],
                       wo=[gram_rel=comp,
                           sem=W],
                       ni=[gram_rel=obj,
                           sem=N],
            sem=[rel=speak,
                 agent=G:{human,[]},
                 object=N:{human,[]},
                 topic=W:{event,[]}]]]}.
```

In addition to the above, suppose that we also have the following lexical entries and grammar rules.

```
ga:{noun,N} -> {np,N=[case_marker=ga]}
ni:{noun,N} -> {np,N=[case_marker=no]}
wo:{noun,N} -> {np,N=[case_marker=wo]}

{vp,VP=[subcat=Case:SUB]} <
       {np,[case_marker=Case]} ->
              {vp,VP=[subcat=New:SUB-{Case}]}
```

In that case the type judgments for the sentences below will be successful.

```
(john,ga,mary,ni,anokoto,wo,hanasu):{vp,[]}
(mary,ni,anokoto,wo,john,ga,hanasu):{vp,[]}
```

5.1.3 Example of the conceptual system rules

The conceptual system rules are sets of rules which determine superordinate and subordinate relations of concepts. The semantic analysis of Laputa uses these conceptual rules when it performs semantic judgemant.

```
{object,[]} < {'Top',[]}
{event,[]} < {'Top',[]}
{concrete-object,[]} < {object,[]}
{creature,[]} < {concrete-object,[]}
{human,[]} < {creature,[]}
{student,[]} < {human,[]}
```

6 An experiment using Laputa

6.1 Conditions of the experiment

computer Multi-PSI 32PE construction

OS PIMOS 3.0.1

The size of the grammar and dictionary

grammar rules	651
words	14,613
morphemes	8,268
concepts	770

We used the syntactic grammar and morphological grammar which were developed by Sano of ICOT's 6th Laboratory [Sano 91]. We made the conceptual system rules in accordance with the conceptual system of the Japan Electronic Dictionary Research Institute EDR.

The experiment We used 22 test sentences and examined 3 types of cooperation pattern: (1) syntactic analysis only, (2) cooperation of morphological analysis and syntactic analysis, and (3) cooperation of morphological analysis, syntactic analysis and semantic analysis.

We checked the relation between the number of processor elements utilized and the number of reductions and processing time for each of these 3 cases.

All the tests have been performed three times, and the measurements given here are the averages computed from these three processing runs.

Example of analysis result To indicate the level of processing of this experiment, I will show the result of analysis of a example sentence.

Example sentence

「彼が父の事業を継いだ」

(He inherited his father's business.)

Analysis result

```
vp(1,[subcat=SUB:[],
      infl=u_ga,
      predicate=[
           lex=継,
           soa=[ga=[sem={man,1,[]},
                    gram_rel=subj],
                wo=[sem={job,6,
                         [of_type={man,2,[]}]},
                    gram_rel=comp],
                sem={tugu,8,
                     [agent={man,1,[]},
                      object={job,6,
                              [of_type=
                                  {man,2,[]}]}]}
                tenseless=action],
           polarity_of_soa=true,
           judgment=affirmation,
           aspect=not_continuous],
      mood=finished,
      recognition=[modality=descriptive,
                   acceptance=affirmative]])
```

6.2 Outcome of the experiment

The the following graph shows, for the analysis of example sentence 12, how the speed-up ratio changes as the number of processors is increased from 1 to 32.

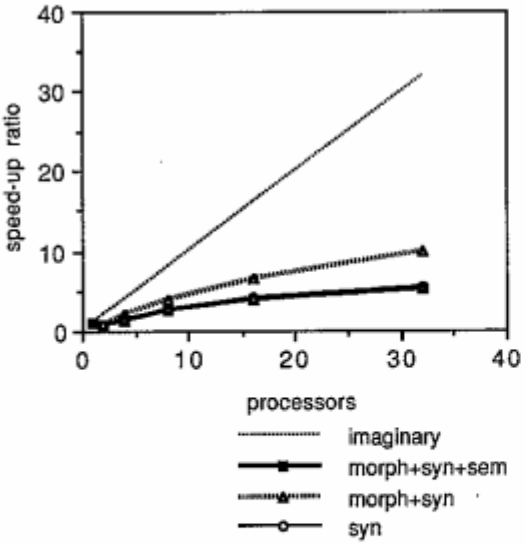

Figure 1: Example 12 processors and speed up ratio

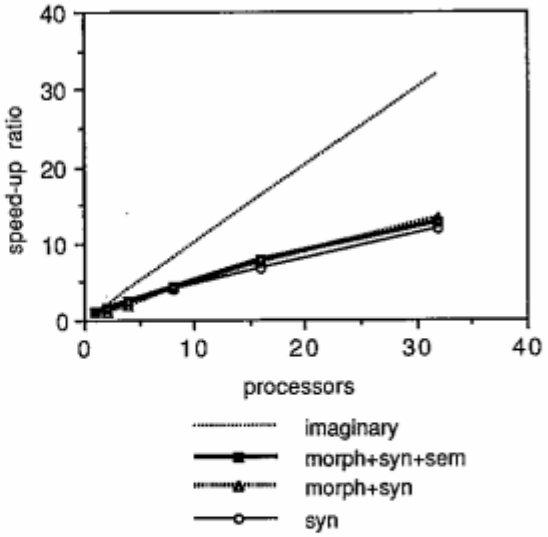

Figure 2: Example 14 processors and speed up ratio

The behavior of the cooperative process of morphological, syntactic and semantic analysis is almost identical to that of syntactic analysis alone, while the cooperation of just morphological and syntactic analysis shows a much better speed up ratio relatively. This might lead one to think that cooperation of just morphological and syntactic analysis makes for a better speed-up ratio. However examples involving a greater amount of calculation do not show this difference. Figure 2 is the result of the analysis experiment on example sentence 14.

This graph show that all three types of cooperation have the same speed-up ratio, which is a different result than that we deduced from example sentence 12.

We can interpret this difference as resulting from a difference in the amount of calculation. Both the syntax-only calculation and the cooperative syntactic and morphological analysis process for example sentence 12 simply do not involve enough computation to fully show the potential speed-up ratio. Sentence 14, on the other hand, requires enough computation for any of the three types of cooperation, so that we can more clearly see the speed-up ratio.

To verify this assumption, we plot the speed-up ratio against the amount of calculation for the three types of cooperation. As the graph shows, all three cooperation types show similar behavior for this relation. We can understand why this is so if we recall that in our system, all modes of processing employ the same basic processing mechanism.

In the graph, we can see that the speed-up ratio rises steeply while the number of reductions remains small, but gradually becomes saturated as the number of reductions grows.

In the bar graph of Figure 4, we can see that the number of reductions for sentence 12 in the case of cooperative morphological and syntactic analysis is about 1,200,000, while in the other two cases it is about half as much (approximately 600,000). Because the number of reductions as a whole is small, this difference is important. Example 14 on the other hand involves enough computation so that the effect is minimized and the similarities between the processing modalities are allowed to come out.

7 Conclusion

In this paper, we proposed a model for integrated natural language processing on a parallel inference machine. This model is realized by choosing similar processing schemes for morphological, syntactic and semantic analysis and having these cooperate in parallel.

Also, we have carried out an experiment to evaluate the practicality of our processing model.

As the result of our experiment we have been able to realize speed-up to a factor of about 13 when utilizing 32 processor elements.

The results also showed that the speed-up ratio is determined only by the amount of computation, and is not influenced by the configuration of cooperating analysis processes.

If our processing model is to be practical as a method for a real parallel inference machine, the object of analysis should require a great amount of calculation because when the amount of calculation is low we can not expect a satisfactory speed-up ratio.

We think that our processing model has the potential

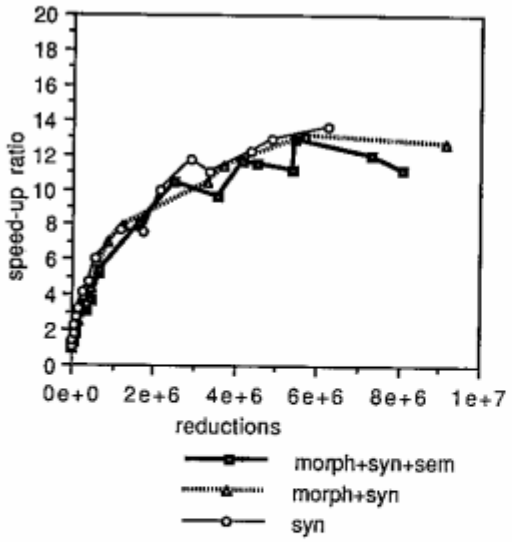

Figure 3: reductions and speed-up ratio

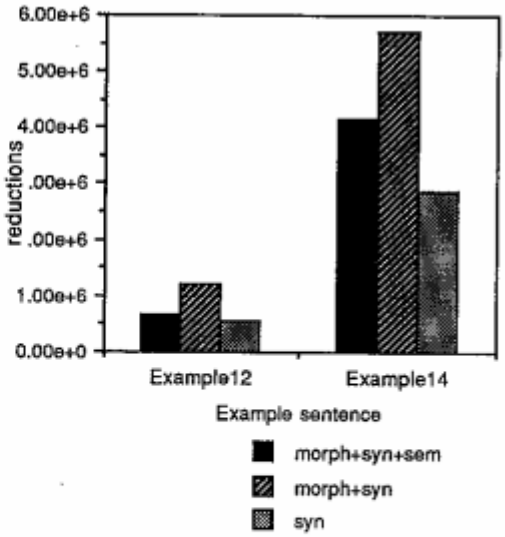

Figure 4: cooperation case and reductions

to be a practical technology for natural language processing, and that it can help increase the amount of cooperation with fields like pragmatics, speech recognition and the utilization of world knowledge.

Acknowledgments

We would like to thank Yuuichi Tanaka, Hiroshi Sano and other members of 6th laboratory of ICOT.

We were supported by Hirosi Onodera, Naoto Hirota, Yoshiko Kamimura and other members of Fujitsu-FIP Ltd in our experiments.

We are also grateful to Eric Visser for his support in finalizing this paper.

References

[Martin-Löf 84] Per Martin-Löf : Intuitionistic Type Theory, Studies in Proof Theory, Lecture Notes, 1984.

[Aït-Kaci 86] Hassan Aït-Kaci and Roger Nasr, LOGIN: A Logic Programming Language with Built-in Inheritance, The Journal of LOGIC PROGRAMMING, Vol. 3, No. 3, Oct. 1986.

[Schmidt-Schauss 89] M. Schmidt-Schauß, Computational Aspects of an Order-Sorted Logic with Term Declarations, Lecture Notes in Artificial Intelligence, Springer-Verlag, 1989.

[Smolka 88] Gert Smolka, A Feature Logic with Subsorts, IBM Deutschland, Stuttgart, West Germany, LILOG Report 33, May 1988.

[Yasukawa 90] Hideki Yasukawa, Kazumasa Yokota, The Overview of a Knowledge Representation Language Quixote, ICOT (draft), Oct. 21, 1990.

[Sano 91] Sano Hiroshi, User's Guide to SFTB, ICOT (draft), Sep. 1991.

[Hasida 91] Koiti Hasida, Aspects of Integration in Natural Language Processing (In Japanese), Japan Society for Software Science and Technology, COMPUTER SOFTWARE, Vol. 8, No. 6, Nov. 1991.

[Hasida 90] Koiti Hasida, Sentence Processing as Constraint Transformation, Proceedings of 9th European Conference on Artificial Intelligence, 1990

[Matsumoto 86] Yuuji Matsumoto, A Parallel Parsing System for Natural Language Analysis, Proc. of 3rd Internatioanl Conference on Logic Programming, London, 1986.

[Sato 90] Hiroyuki Sato, An improvement of a parallel natural language analyzing system PAX (In Japanese)), Proc. of KL1 Programming Workshop '90, ICOT, Tokyo, 1990.

Appendix

Example sentences

Example1 太ってやる
(I will get fat.)

Example2 木に竹を継ぐな
(Do not connect bamboo to wood.)

Example3 彼は父の事業を継いだ
(He inherited his father's business.)

Example4 話の途中で電話を切るな
(Don't hang up the telephone in the middle of a conversation.)

Example5 今日の奈々子はこの前より太っていた
(Today's Nanako is fatter than before.)

Example6 部屋から出たら電気のスイッチを切りなさい
(Turn off the light when you leave the room.)

Example7 彼を呼んで小言を言ったから順調に進んでいる
(Because I called and scolded him, it is progressing well.)

Example8 そういう理由で髪を切ったことを彼女は話しはじめた
(She began to tell me that that was the reason why she cut her hair.)

Example9 東京に行く彼女が乗った汽車が進みはじめたら雪が降ってきた
(When the train that she, who was going to Tokyo, was on started moving, snow began to fall.)

Example10 私は彼女を信じていたのにそういう奈々子は夜までわざと帰ってこなかった
(Though I believed her, Nanako didn't return until the night on purpose.)

Example11 その会議で私だけが意見を言ったことで後で会社でこういう問題になった
(Because only I expressed my opinion at that meeting, later at the company we had this kind of problem.)

Example12 電車が来るのを待ちながらパーティに呼ぶ友達のことを奈々子が彼に話していた
(While waiting for the train, Nanako told him about the friends that she would invite to the party.)

Example13 私の父が問題の荷物を解いて彼に見せたら後でサウジアラビア王国の女性から電話が掛かってきた
(First my father opened the package in question and showed it to him, and later we got a telephone call from a woman from the Kingdom of Saudi Arabia.)

Example14 クリスマスの夜に学校から帰ってくる奈々子を待ちながら父と話していたら雪になった
(As I talked to my father while waiting for Nanako to come home from school on Christmas Eve, it started snowing.)

Example15 アメリカからこの前に話していた専務を呼んだからあの伯父が父の事業を継ぐだろう
(Since he called that managing director I was telling you about before from America, that uncle will probably inherit my father's business.)

Example16 私だけがその方程式を解いていなかったから解き方を知っている友達に教わってその問題を解いた
(Since only I hadn't solved that equation, my friends who knew how to solve it helped me out and I solved the problem.)

Example17 私だけがその方程式を解いていなかったから解き方を知っている友達に教わってその問題を解いて寝た
(Since only I hadn't solved that equation, my friends who knew how to solve it helped me out and I solved the problem and went to bed.)

Example18 社長がアメリカからこの前に話していた専務を呼んだからあの伯父が父の事業を継ぐだろう
(Since the president called that managing director I was telling you about before from America, that uncle will probably inherit my father's business.)