ICOT Technical Report: TR-193

TR-193

"KABU-WAKE" PARALLEL INFERENCE MECHANISM AND ITS EVALUATION

by
H. Masuzawa, K. Kumon, A. Itashiki
K. Satoh and Y. Sohma (Fujitsu Ltd.)

November, 1986

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

"KABU-WAKE" PARALLEL INFERENCE MECHANISM AND ITS EVALUATION

Hideo Masuzawa, Kouichi Kumon, Akihiro Itashiki,
Ken Satoh, and Yukio Sohma

FUJITSU LIMITED
1015 Kamikodanaka, Nakahara-ku, Kawasaki 211, JAPAN

Abstract

We have proposed a new parallel inference mechanism called the KABU-WAKE method, and have built an experimental OR parallel inference system based on this method. This paper evaluates use of the KABU-WAKE mechanism in practical applications. Tests on our experimental system using a restricted bottom-up parser show that the KABU-WAKE method is very effective for large problems that contain a lot of parallelism. This paper also describes two difficulties of the KABU-WAKE method made evident through analysis of the status of each processor element during the processing of a problem with a low performance improvement ratio. We also estimated the performance improvement ratio expected with 100 processor elements. This research was sponsored by MITI as a part of the FGCS project.

1. Introduction

AI research is now flourishing. In this field, a problem with one of the important features can be solved by traversing a search tree using trial and error methods. A logic program can also be executed using parallel processing, such as OR parallel, AND parallel, or AND-STREAM parallel processing. OR parallel processing is suitable for non-deterministic algorithms and we therefore think it is well suited to a search-type problem described above.

We proposed a new parallel inference mechanism, called the KABU-WAKE method. We built an experimental OR parallel inference system implementing this method [1], [2]. We ran a simple game problem, the N-Queen problem, on the experimental system to evaluate our method [3], [4]. At present, we are evaluating our method for two practical applications, both of which are parser programs with restricted grammars. We have already presented an evaluation of one of the applications-- a top-down parser for the Japanese language. The algorithm produced a top-down parsing tree from a sentence [6]. This paper evaluates the KABU-WAKE method using a bottom-up parser for the Japanese language. The algorithm makes a parsing tree from terminal symbols at the bottom [5]. By analyzing the status of each Processor Element (PE) during processing, we will explain 1) why the performance improvement ratio (how much performance improves over performance with 1 PE) with fewer than 12 PEs was high using a large problem, 2) why the performance improvement ratio was low using a problem that cannot achieve a high performance improvement ratio with 12 PEs, and 3) estimate the performance improvement ratio expected with 100 PEs using a problem that achieved a high performance improvement ratio with fewer than 12 PEs. Chapter 2 discusses problems in parallel inference processing, and explains the features and principles of operation of the KABU-WAKE method. Chapter 3 outlines our experimental system. Chapter 4 discusses the results of executing the bottom-up parser program on our experimental system.

2. KABU-WAKE method

2.1 Problems in Parallel Inference Processing

Our original question is how to make an OR parallel processing inference system so that performance is proportional to the number of PEs. The following problems need resolution:

(1) Problems within each PE
  i. Reducing overhead for switching and scheduling tasks
  ii. Reducing the time for creating and dividing tasks

(2) Problems between PEs
  i. Achieving dynamic load balancing, because we do not know how much

processing a program requires before it is executed.
ii. Reducing the number of subtask transfers
iii. Reducing the amount of time for each subtask transfer

The KABU-WAKE method solves most of these problems for certain applications, as we discuss in the next section.

2.2 Principles of Operation

This section discuss the operation principles of the KABU-WAKE method. Figure 1-a is an example of a knowledge base written in Prolog. We have omitted the arguments of the predicates to simplify explanation. The arrows indicate the normal flow in a conventional sequential inference. Figure 1-b is a snapshot of processing using the KABU-WAKE method. The bold lines indicate the processing in PE 0, corresponding to the processing indicated by the arrows in Figure 1-a. If a request comes from PE 1, PE 0 divides its task at the node closest to the root of the search tree and transfers a portion to PE 1. If a request then comes from PE 2, PE 0 again divides a part of the task in the same way. This method enables several PEs to solve one problem.

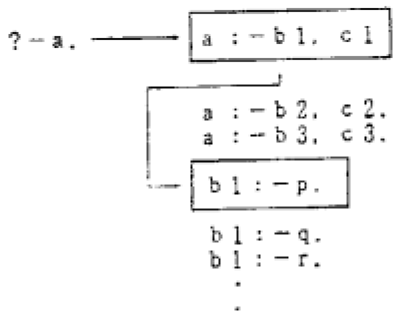


Figure 1-a

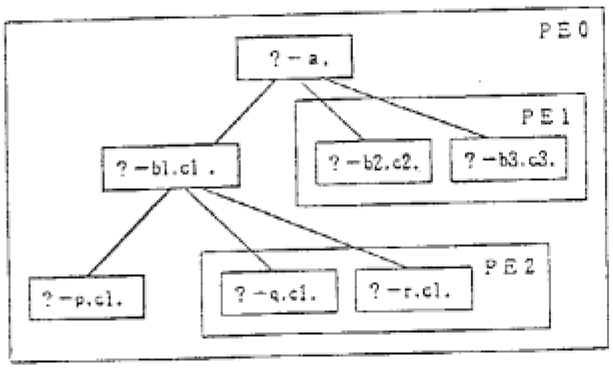


Figure 1-b

Figure 1 Basic KABU-WAKE mechanism

2.3 Features

The KABU-WAKE method has two principal features:
-- Each PE uses the same search strategy as is used by a conventional sequential processor to traverse a search tree, namely, a depth-first search.
-- A PE divides its task and transfers a portion of it only when requested by another PE.

These features have the following advantages:

(1) Low overhead in each PE
i. Each PE processes only one task.
--> There is no overhead for switching and scheduling tasks as in a multiple task environment.
ii. A task is divided at the node closest to the root of the search tree.
--> We think that each PE should divide a task into the largest possible subtasks.
iii. Each PE uses conventional sequential processing.
--> There is almost no overhead for parallel processing during execution in each PE [4]. Also, this method can use the high-speed techniques developed for conventional sequential processing, such as in compilers.

(2) Low overhead between PEs
i. A busy PE divides its task and transfers a portion only when requested by another PE.
--> When all the PEs are busy, there is no communication.
ii. A distributed processing method is used in which only idle PEs receive tasks.
--> Effective dynamic load balancing is possible.

(3) Easy implementation
i. The number of tasks does not exceed the number of PEs.
--> No mechanism is required to temporarily hold extra tasks for PEs.

3. Experimental System

3.1 System configuration

Figure 2 shows the system hardware configuration. The experimental system consists of 16 PEs (one for the man-machine interface) and two networks. These networks are used for different purposes, and were designed specifically for our method.

(1) PE

A PE performs inference processing. It gives part of it's task to another PE if a request comes during processing. A KABU-WAKE interpreter is installed in each PE. This interpreter is a conventional sequential interpreter with added control mechanisms [2],[4] to implement the KABU-WAKE method. Each PE has a copy of the entire database.

(2) CONTROL NETWORK

The control network is a communication route for requesting tasks. This network connects all PEs in a ring via control network adapters (CNAs). The clock frequency is 2 MHz. A packet, which indicates whether each PE is busy or idle, goes around the network [6] in 8-microsecond cycles.

(3) DATA NETWORK

The data network is a communication route for transferring tasks. It is a two-stage network made up of 4 x 4 switches. The communication method is DMA transfer of 8 bits in parallel. The transmission rate of the network hardware without overhead is about 400 KB/s, but it drops to 50 KB/s with the addition of the communication software overhead [6].

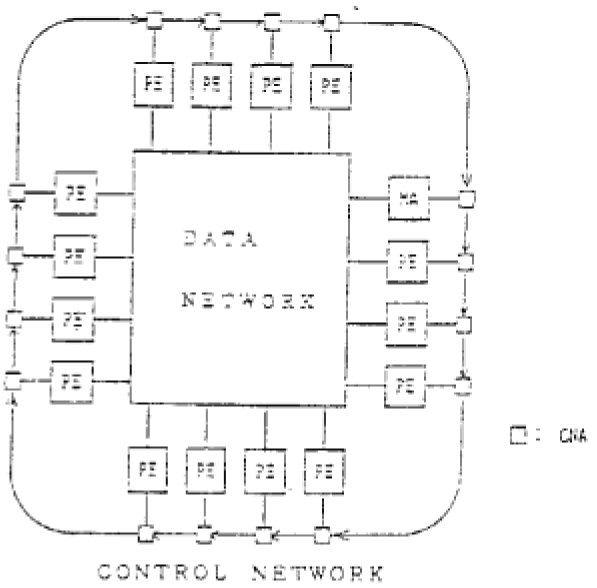


Figure 2 System configuration of experimental machine

3.2 Implementation

The following items need be implemented in our experimental system.

(1) Unbinding of variables

When a search tree is split, the status of the variables in the split tree must be as if all the processes located below the split portion of the search tree had failed and backtracked, i.e., unbinding. To speed up this unbinding operation, we reserved an area for each variable in which was stored the time at which it was bounded. Each time a subgoal is reached, a new level number, corresponding to the depth of subgoal, is assigned. For variables bound during the processing of that subgoal, the same level number as that of the subgoal is flagged. When a tree is split, we can determine whether binding should be released by comparing the level of the subgoal to be split with the variable's level number. In this method, the time required to release variables for splitting is proportional to the number of variables in the subgoal to be split. Using a trailing stack is another alternative, but we think the processing would take more time.

(2) Use of rule numbers

When a tree is split and transferred, it is transferred in the form of a subgoal. However, since some of the definitions for the subgoal are already being processed, the other PEs must start from subsequent definitions. We, therefore, we adopted a method in which a special predicate, called a rule number, is prepared and attached to the subgoal before transfer to indicate where to start execution.

(3) Control of request

Some tasks cannot be split when a request is received from another PE. If this is often the case, PE performance deteriorates. To handle this situation, we used a request reception flag to indicate whether a task can be split. This enables a PE to concentrate on a task without being disturbed by other PEs.

4. Evaluation

Let us now consider the results obtained by running the bottom-up parser on our experimental system. We will first present the results, then discuss them.

4.1 Results

4.1.1 Test programs

We entered three sentences into the bottom-up parser (44rules, 103facts). The characteristics of the three bottom-up parser programs corresponding to the three sentences entered are shown in Table 1. Here, the bottom-up parser is a Japanese-language parser; the algorithm makes a parsing tree from terminal symbols at the bottom. We define a execution time with 1 PE as the amount of processing time. The ratio of the amount of processing time in Sentence 1 to that in Sentence 2 is 1:6 and of Sentence 1 to Sentence 3 is 1:60. The amount of processing time required in Sentence 1 is about 1.3 times as much as in 7-Queen.

Table 1 Logical characteristics of bottom-up parser programs

ITEM / PROGRAM	Average number of parallel degrees *1	Number of inference levels
Sentence 1	45	1294
Sentence 2	236	2081
Sentence 3	1427	2388

Notes

*1 : Average number of parallel degrees = ((total number of inferences) / (number of inference levels))

*2 : The three sentences entered are as blow :

Sentence 1
"wakai otoko wa kouen de akai boushi no onna no ko ga hon wo yomu nowo mita"
(meaning) :
A young man saw a girl with a red hat reading a book at a park.

Sentence 2
"aoi fuku no wakai otoko wa kouen no ki no benchi de akai boushi no onna no ko ga atarashii ryouri no hon wo yomu nowo mita"
(meaning) :
A man wearing blue clothes saw a girl with a red hat reading a new cookbook on a wood bench at a park.

Sentence 3
"akai fuku no wakai otoko wa watashi no machi no kouen no ki no benchi de akai boushi no onna no ko ga atarashii ryouri no hon wo yomu nowo mita"
(meaning) :
A young man wearing red clothes saw a girl with a red hat reading a new cookbook on a wood bench at a park in my town.

4.1.2 Performance improvement ratio

Figure 3 shows the relationship between the number of PEs and the performance improvement ratio due to parallel inference. It shows that in a large problem, such as Sentence 3, the performance improvement ratio is almost proportional to the number of PEs. As the amount of processing decreases (Sentence 3 -> Sentence 2 -> Sentence 1), the performance improvement ratio decreases. The ratio of Sentence 1 increases only until the number of PEs reaches 8, then it levels off.

4.2 Discussion

This section discusses the following three items.

(1) Performance improvement ratio of Sentence 3
Figure 3 shows that the performance improvement ratio of Sentence 3 is higher than that of Sentence 1 and 2. We will discuss the cause.

(2) Performance improvement ratio of Sentence 1
Figure 3 shows that the performance improvement ratio of Sentence 1 is low. We will discuss the cause, and describe several difficulties related to the KABU-WAKE method made evident through our analysis.

(3) Estimated performance improvement ratio of Sentence 3
One of the AI applications is a search problem; we think this type of problem requires a great deal of processing. We regard Sentence 3 as such a search-type problem and estimate the performance improvement ratio with 100 PEs.

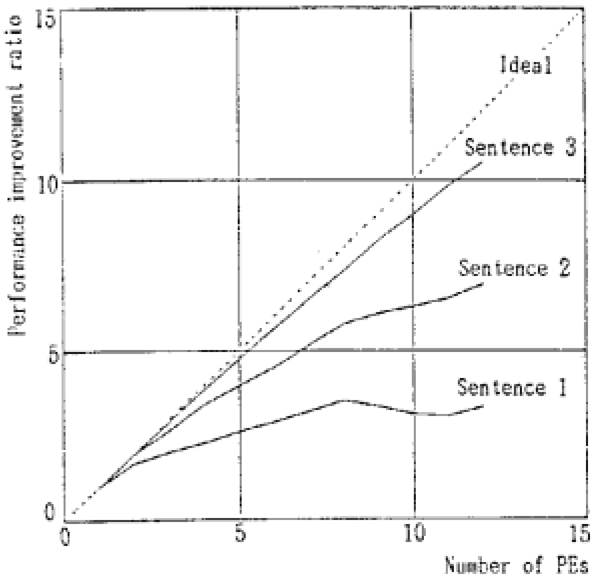


Figure 3 Performance improvement ratio due to parallel inference

4.2.1 Performance improvement ratio of Sentence 3

Figure 4 graphs the relationship

between the processing time (with 1 PEs) and the number of subtask transfers (with 12 PEs). In Figure 4, the xcoordinate is a logarithm of the processing time. Figure 4 indicates that the number of subtask transfers is almost proportional to the logarithm of the processing time, but not to the processing time itself. This means that, as the processing time increases exponentially, the number of subtask transfers increases linearly. That is, for similar programs, the longer the processing time, the less communication overhead. This is why the performance improvement ratio of Sentence 3 is higher than that of Sentence 1 and 2.

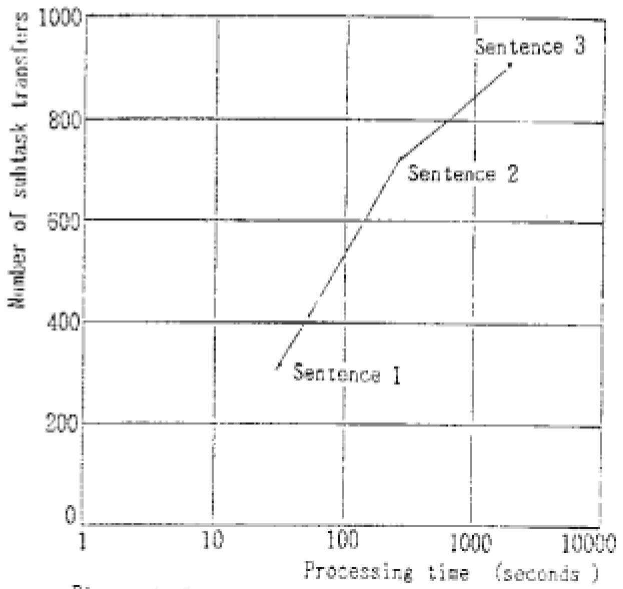


Figure 4 Relationship between processing time and number of subtask transfers (PEs =12)

4.2.2 Performance improvement ratio of Sentence 1

Table 1 shows that the average number of logical parallel degrees in processing by the KABU-WAKE method is 45. However, the performance improvement ratio of Sentence 1 does not reach 4, even when Sentence 1 is processed with 12 PEs.

<1> Results

The experimental results show that when the number of PEs is 12, the rate of processing items (busy, KABU-WAKE, idle) are as follows:

busy time: 30.6%,
KABU-WAKE time: 43%,
idle time: 26.4%

Busy time includes all of the inference processings. KABU-WAKE time includes all of the KABU-WAKE processings (explained later).

The ratio of processing time shows that KABU-WAKE time and idle time accounted for more than 70% of all the time used to process Sentence 1. These two factors (we call them overhead for parallel execution) prevent performance improvement.

<2> Analysis

To discover why the overhead for parallel execution such as both KABU-WAKE time and idle time are large, we discuss some factors which may cause the overhead.

There are two factors which may cause the overhead:

Factor 1: KABU-WAKE processing time per transaction
The time depends on what feature of KABU-WAKE processing is being used.
Factor 2: number of KABU-WAKE processings
The number depends on the nature of parallelism in the search tree of Sentence 1.

(1) Features of KABU-WAKE processing

Figure 5 details the KABU-WAKE processing time in the execution of Sentence 1. The KABU-WAKE processing time is 106 ms: 49 ms for Sender to transfer a subtask and 57 ms for Receiver to receive the subtask. While Sender does Split processing in the KABU-WAKE processing, the Receiver is idle (Wait in Figure 5), and the Wait time is 33 ms. This minimum time is needed for each KABU-WAKE processing. It takes 2 ms for the KABU-WAKE interpreter to process a inference, so the interpreter can process about 53 inferences during KABU-WAKE processing time and about 16 inferences during Wait time. The KABU-WAKE processing time per transaction was long.

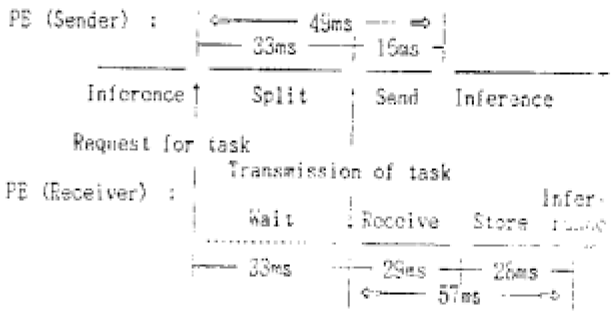
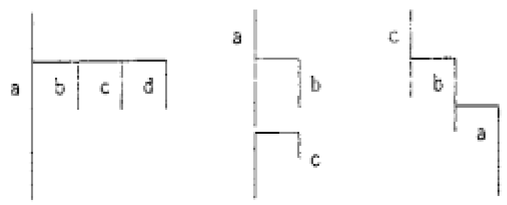


Figure 5 Details of KABU-WAKE processing time

(2) Nature of parallelism in the search tree of Sentence 1.

Almost all the parallelisms are of the type shown in Figure 6, and almost all branches (b, c, and d in Figure 6) in the portions with such parallelisms have a number of inferences smaller than 60.



Almost all branches b, c, or d has a number of inferences smaller than 60.

Figure 6 Characteristics of a subtask that to can be processed in parallel by a bottom-up parser

In the execution of Sentence 1 with 12 PEs, the transferred subtasks, consisting of fewer inferences than 24, is about 36% of all the transferred subtasks, and consisting of fewer than 60 is about 81%. It means that the number of KABU-WAKE processings was large because the transferred subtasks consisted of a small number of inferences. Further the beginning and the ending of execution was generally not good. Also because of the long KABU-WAKE processing time and the large number of transferred subtasks, there were many idle PEs that sometimes became busy. This cause a lot of idle time. Figure 7 shows the dynamic

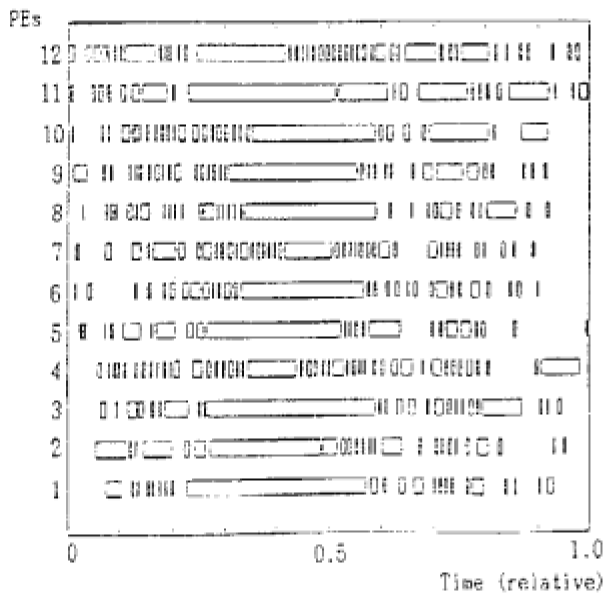


Figure 7 Change of status in each PE during processing of Sentence 1

change of status of each PE during the processing of Sentence 1. In Figure 7, busy, KABU-WAKE, and idle states have the following meanings:

-- Much idle time occurred during the first and last thirds of the execution. This accounted for more than 90% of all idle time.
-- Communication was distributed uniformly from the beginning to the end of execution. In the first third, the number of subtask transfers was 112, in the second third, 108, and in the last third, 125.
-- All of the PEs processed the majority of inferences during the middle third of execution.

The KABU-WAKE processing and the search tree of Sentence 1 have the above characteristics. When we return to the factors which cause overhead in parallel execution, the KABU-WAKE processing time per transaction was long and the number of KABU-WAKE processings was large. So, the overhead (KABU-WAKE time and idle time) took up more than 70% of all the time used to process Sentence 1.

Two difficulties in the KABU-WAKE method were revealed during analysis of the above experiment.

i. KABU-WAKE processing time per transaction is longer than it should be.
ii. The number of inferences which make up a transferred subtask have a large influence on the performance improvement ratio.

4.2.3 Estimated performance improvement ratio of Sentence 3

This section describes the estimated performance improvement ratio expected if Sentence 3 is processed with 100 PEs.

The following expression shows how the performance improvement ratio with 100 PEs is obtained.

$$\text{performance improvement ratio} = \frac{\text{(execution time with 1 PE)}}{\text{(execution time with 100 PEs)}}$$

The value of the execution time with 1 PE is based on actual measurement. During execution with 100 PEs, each PE does inference processing, does KABU-WAKE processing, or is idle. So, the execution time with 100 PEs is determined by the following expression.

```
execution time with 100 PEs =
     ((busy time) +
        (KABU-WAKE time) + (idle time))
    ---------------------------------
                  (100)
```

In the above expression, busy time is the sum of inference processing time from PE 1 to PE 100: we can regard the busy time as the execution time with 1 PE. The KABU-WAKE time is determined by the following expression.

```
KABU-WAKE time =
   ((KABU-WAKE processing time
                per transaction) *
      (number of subtask transfers))
```

Here, the value of the KABU-WAKE processing time per transaction is based on actual measurement of the execution of Sentence 3 with less than 12 PEs. To estimate the number of previous subtask transfers when Sentence 3 is processed with 100 PEs, we used the relationship between the number of PEs and the number of subtask transfers obtained by actually measuring the execution of Sentence 3 with less than 12 PEs. (In the KABU-WAKE method, the relation is linear [4],[6].) We estimated the idle time from the relationship between the number of PEs and the amount of idle time (shown in Figure 8) obtained by actually mesuring the execution of Sentence 3 with less than 12 PEs. Figure 8 indicates that the relationship is almost linear.

The above computations give an estimated performance improvement ratio of about 50 when Sentence 3 is processed with 100 PEs.

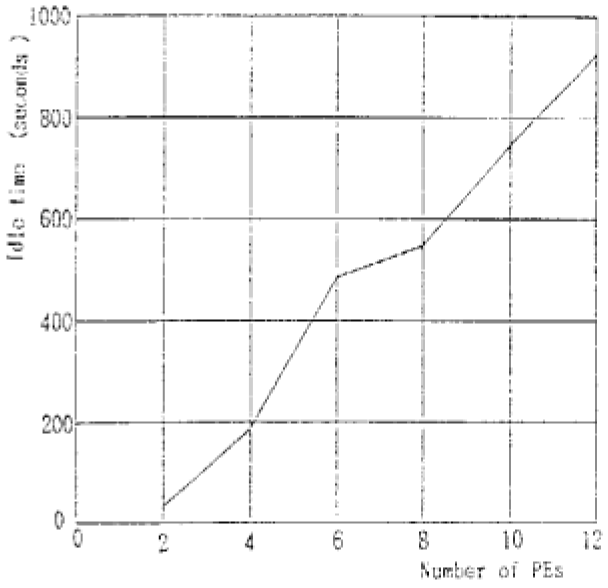


Figure 8 Relationship of number of PEs to idle time

5. Conclusion

We evaluated the KABU-WAKE method by running a bottom-up parser program with restricted a grammar on our experimental system. The results led us to the conclusions listed below.

(1) With the KABU-WAKE method, we can obtain a performance improvement ratio proportional to the number of PEs for large problems with a lot of parallelism. The method has the advantage for parallel inference that, as the amount of processing increases exponentially, the number of subtask transfers increases linearly.

(2) Difficulties in the KABU-WAKE method made evident through analysis of the cause of the low performance improvement ratio when 12 PEs were used is as follows:
  i. The KABU-WAKE processing time per transaction is longer than it should be.
  ii. The number of inferences which make up a transferred subtask greatly influences the performance improvement ratio.

(3) The estimated performance improvement ratio is about 50, when a problem that achieved a high performance improvement ratio with fewer than 12 PEs is processed with 100 PEs.

In our analysis described in 2, above, we saw that the KABU-WAKE processing time had a great deal of influence on the performance improvement ratio. We think it is important to reduce the KABU-WAKE processing time per transaction. For 2 i, above, we have built a KABU-WAKE compiler that reduces the KABU-WAKE processing time per transaction and are awaiting an evaluation of it. For 2 ii, as the KABU-WAKE processing time gets shorter, its influence on the performance improvement ratio becomes smaller. We are investigating other possible solutions to 2 ii that relate to the basic principles of the KABU-WAKE method.

We think that one of the practical applications in AI, the search problem, requires a large amount of processing with a large number of subtasks that can be processed using OR parallel. The KABU-WAKE method is very suitable for this type of application.

FGCS is attempting to develop an AND-STREAM parallel inference machine, which executes the GHC (Guarded Horn Clause) language [7]. From the point of view of GHC, OR parallel Prolog, that is pure Prolog, is a language that is used to describe special types of algorithms such as a search problem. Therefore, when we think of the GHC-oriented machine as a general purpose parallel inference machine, we can regard an OR parallel Prolog-oriented machine as a special-purpose parallel inference machine. As search problems appear in most AI applications, we think the OR parallel Prolog-oriented machine is quite valuable.

6. Acknowledgments

The authors would like to thank Mr. Tanahashi, Section Manager, and Mr. Sato, General Manager, for their unfailing encouragement and Mr. Yamada, Managing Director, for giving us the opportunity to conduct this research.

References

[1] Itashiki, et al., "PARALLEL INFERENCE PROCESSING SYSTEM -- EXPERIMENT OF IMPROVED CLAUSE UNIT PROCESSING METHOD", 30th meeting of Information Processing Society, 7c-7, March 1985. Japanese.

[2] Kumon, et al., "PARALLEL INFERENCE PROCESSING SYSTEM -- IMPROVED CLAUSE UNIT PROCESSING METHOD", 30th meeting of Information Processing Society, 7c-8, March 1985. Japanese.

[3] Kumon, et al., "EVALUATION OF KABU-WAKE PROCESSING", 31st meeting of Information Processing Society, 2c-5, October. 1985. Japanese.

[4] Sohma, et al., "A NEW PARALLEL INFERENCE MECHANISM BASED ON SEQUENTIAL PROCESSING", IFIP TC-10 Working Conference On Fifth Generation Computer Architecture, July 15-18, 1985. Also, J.V.Woods, "A NEW PARALLEL INFERENCE MECHANISM BASED ON SEQUENTIAL PROCESSING", Fifth Generation Computer Architecture, North-Holland, 1986.

[5] Mizoguchi,et al.,"Prolog and its applications", Sohken Syuppan Press, 1985. Japanese.

[6] Kumon, et al., "KABU-WAKE : A NEW PARALLEL INFERENCE METHOD AND ITS EVALUATION", COMPCON 86 Spring, March 4-6, 1986.

[7] Ueda,K., "Guarded Horn Clauses", TR-103, 1985.