TR-370

# Measurements and Evaluation for the Multi-PSI/V1 System —A Study of Inter-PE Communication versus System Performance—

by
K. Taki

April, 1988



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

(03) 456-3191 ~ 5
Telex ICOT J32964

# Measurements and Evaluation for the Multi-PSI/V1 System
## — A Study of Inter-PE Communication versus System Performance —

Kazuo Taki
Institute for New Generation Computer Technology
1-4-28, Mita, Minato-ku, Tokyo 108, Japan

**Abstract**

The Multi-PSI is an experimental parallel machine mainly used for the parallel software research.

A network-connected multiprocessor like the Multi-PSI has such characteristics that a bottleneck of the network communication resides in the communication processing of the processing element (PE) not in the network transfer. Measurements on the Multi-PSI/V1 shows that the inter-PE processing costs around ten times more expensive than the intra-PE processing. That is, the inter-PE processing consumes much CPU time reducing system performance.

The relation among system performance, communication efficiency and communication frequency is discussed and described into numerical formulas. Measurement scale of these factors are also presented. The formula can be applied to network-connected multiprocessors like the Multi-PSI.

The real value of those factors affecting system performance are measured on the Multi-PSI/V1 using sample programs which contains load distribution control. Measurements complete the numerical formula which specifies the relation among system performance, communication frequency and system work rate of the Multi-PSI/V1. Usage of the numerical formula is also discussed in order to tune the communication frequency of programs for low communication overhead and good load balance.

# 1 Introduction

One of the main research themes of the FGCS project has been the Parallel Inference Machine (PIM). The aim is to build a parallel computer system which is an engine for high performance knowledge information processing[3,13]. Not only machine architecture research but also research on parallel software is indispensable for the PIM development and feedback from the parallel software research to the architectural research is also necessary.

The Multi-PSI is a dedicated R&D tool to advance the parallel software research and development[12]. The R&D results will be incorporated in the PIM system. The

Multi-PSI is a small scale parallel computer system constructed from several PSI machines [11,7] and dedicated high speed network.

We have been developing several Multi-PSI systems:

1. Multi-PSI/V1: a prototype system including six PSI-I machines. It began operation in 1986.
2. Multi-PSI/V2: a full-scale system including 64 PSI-II machines. The PSI-II is a high speed and compact version of the PSI-I. The system will start operation in 1988.
3. Pseudo-multi-PSI: simulators for the Multi-PSI/V1 and Multi-PSI/V2 developed on the PSI-I and PSI-II. They are mainly used for debugging parallel programs. A simulator for the Multi-PSI/V1 began operation in 1986.

The Multi-PSI/V1 has been working as a prototype system. This paper reports a study on the Multi-PSI/V1 system.

Much research in many areas remains to be done in the parallel software field. We have decided on the following research themes for the Multi-PSI system.

- A parallel logic programming language, KL1, and its implementation
- The Parallel Inference Machine Operating System, PIMOS
- Debugging method , programming system , and measurement and evaluation method
- Basic research on parallel algorithms and load distribution methods
- Parallel application programs

From this wide range of possible research topics, we have selected several for the Multi-PSI/V1. They have high priority or can be researched on the prototype system. They are :

1. Detailed study of the FGHC parallel execution mechanism (FGHC is a logic programming language giving kernel language specifications of the KL1)
2. Experimental implementation of the FGHC parallel processing system
3. Writing sample programs which can be used as parallel benchmark programs

These three are reported on the other articles[4,10].

4. Basic study of the measurement and evaluation method, especially focusing on inter-PE (processing element) communication and the system performance
5. Parallel execution of sample programs and experimental load balancing

6. Identifying the focus of research through experience in the parallel programming and parallel processing

This paper mainly deals with items 4 and 5.

Section 2 describes the overall structure of the Multi-PSI/V1 and the motivation of the research work. Section 3 deals with special features of FGHC and its parallel execution mechanisms giving the basic concepts of parallel processing and the inter-PE (processing element) communication in the Multi-PSI.

Section 4 measures and analyzes component ratios of inter-PE communication time in the Multi-PSI/V1, such as software, firmware and hardware time. It shows that network transmission time is negligible and identifies time consuming items as pre- and post processing of the inter-PE network communication. Estimation for the Multi-PSI/V2 is also made.

Section 5 discusses the relation between system performance and efficiency of the time consuming communication processing and describes it into numerical formulas. Scales for communication frequency and communication efficiency measurements are also proposed. Section 6 describes load balancing, another factor affecting the system performance.

Section 7 shows measurement results of the several factors dealt with in the numerical formulas in sections 5 and 6. Relative communication time of the system, communication rate of sample programs, and average work rate of the system are measured on the Multi-PSI/V1 using sample programs which includes the load distribution control. These measurements complete the numerical formula which gives the system performance of the Multi-PSI/V1.

Section 8 discusses that these measurement can be used for tuning the communication frequency of programs in order to achieve good load balance and low communication processing.

# 2 Multi-PSI/V1 System

The Multi-PSI/V1 contains six PSI-I machines [6,11,15] as processing elements (PEs). Each PSI-I is a full workstation including I/O devices. PEs are connected by a high speed lattice network dedicated for the system. There is no shared memory. A parallel execution system for a parallel logic programming language FGHC (Flat GHC) (cf. section 3) has been implemented. The FGHC execution system is written in ESP (an extended Prolog) which is the system description language of the PSI machine. Network control functions are added to the ESP as built-in predicates.

The lattice network, shown in Figure 1, is connected to each PE at each intersection or network node (black circle).

Each network branch contains independent two-way signal lines. Network communication is performed by message passing. A network node has a message routing function whereby a message is automatically routed to the specified destination PE. The transmission data width is eight data bits and a control bit. The network interface circuit is installed in the PSI internal bus and controlled by the PSI firmware which is invoked as a built-in predicate.

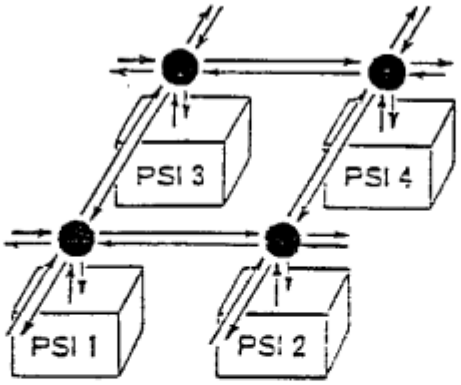


Figure 1: Network structure of Multi-PSI/V1

The Multi-PSI is a network-connected multi-processor system in which each PE has its own main memory and independent memory address space. The shared memory architecture was not chosen because it may have a serious performance problem caused by the memory access contention when applied to large scale multi-processor systems. The structure of independent memory address spaces among PEs suits to divide the garbage collection into intra-PE garbage collection and inter-PE garbage collection, which may be a good solution for the efficient memory management in the large scale multi-processor system.

In the network-connected multi-processor like the Multi-PSI, an address translation mechanism is required when a reference pointer is passed across the processor boundary. An address in the referenced processor is converted into a global identifier for the inter-PE transmission, and the global identifier is converted into a local address in the referencing PE again. Address translation tables between local addresses and global identifiers are kept at the entrance and exit of PEs. Once a garbage collection occurs in a PE, local addresses of logical variables pointed from different PEs may change, whereas global identifiers won't change. Thus, the intra-PE (local) garbage collection can be separated from inter-PE garbage collection.

The network-connected multi-processor structure with independent address spaces of PEs may have good applicability for the large scale parallel computer systems. However, inter-PE communication cost in such a structure may increase at least with the amount of address translation overhead. That is, the inter-PE communication may cost more expensive than the intra-PE processing. The major research interest in this article is how much the inter-PE communication cost will be, and how much the inter-PE communication will degrade the system performance. They are discussed in the following sections.

# 3 FGHC and Inter-PE Processing

This section describes the features of FGHC parallel execution method, giving the basic concepts of parallel processing and inter-PE communication in the Multi-PSI.

## 3.1 Flat GHC

Flat GHC (FGHC) is a subset of Guarded Horn Clauses (GHC) [14]. It is a parallel logic programming language similar to Concurrent Prolog [?] and PARLOG [2].

A FGHC procedure consists of a set of clauses of the form:

$$\underbrace{H :- G_1, \ldots, G_m}_{\text{guard}} \mid \underbrace{B_1, \ldots, B_n}_{\text{body}}. \ (m > 0,\ n > 0)$$

where $H$, $G_i$, and $B_i$ are atomic formulas. $H$ is called the head, $G_i$ the guard goals; together they form the guard part. $B_i$ are called the body goals. The vertical bar ( | ) is called a commitment operator. The guard part of the clauses contains unifications and calls to system predicates only.

The execution of a FGHC procedure can be intuitively explained as follows. Executable goals are queued in a ready queue. One of them is picked up and a procedure whose clause head matches the goal is invoked.

After invocation, all clauses defining the procedure can run in parallel[1]. If some of the clauses succeed in the execution of the guard part, one and only one of them is (nondeterministically) selected and execution of its body part begins (the execution of the other clauses is discarded). This is called a reduction of a goal into body goals. Then the body goals are queued again.

The unification in the guard part cannot instantiate variables – instead the unification is suspended until the variables become instantiated. When the suspensions occur in all guard parts of the clauses, the goal that has invoked this procedure is regarded as suspended. This is the basic mechanism of synchronization in FGHC.

We extend the original FGHC to include the metacall[2] mechanism [2] and the pragmas [8,1]. The former is included for ease of writing system programs.

## 3.2 Pragma

The pragma is designed to allow the programmer to specify explicitly how the goals should be assigned to the processors. Body goals may have pragmas specifying on which processing element the goal should be executed when the parent goal is reduced to the body. Syntactically, a goal, `G`, with a pragma, `P`, is denoted by

```
P@@G.
```

Currently in our system, a pragma directly specifies the processor number[3]. In the following example, the invocation of *translate(1)* will result in the first body goal to be executed on the processor **PE#1**, the second on **PE#2**, and the third on **PE#3**.

[1] In current implementation, all guard parts are executed sequentially in literal order.

[2] In the newest KL1 specification, metacall is called sho-en.

[3] In the future system, the pragma should specify the load distribution at a more logical level.

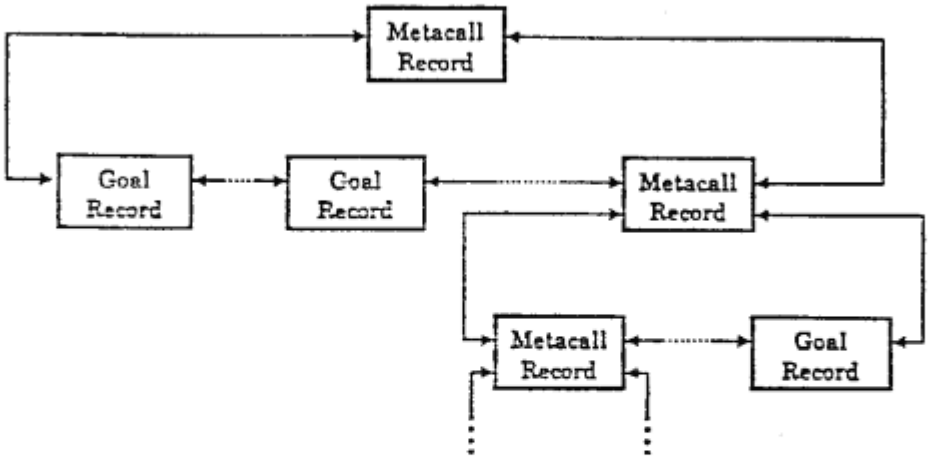


Figure 2: AND-tree

```
translate(PE1) :-
    PE2 := PE1+1,
    PE3 := PE1+2 |
        alloc(PE1)@@instream(I),
        alloc(PE2)@@translate(I,O),
        alloc(PE3)@@outstream(O).
```

## 3.3 AND-tree

The AND-tree (Figure 2) maintains all goals under execution. The roots of the subtrees are metacalls, and the leaf nodes are FGHC goals other than metacalls. (Probably the root metacall will invoke the operating system.)

A metacall is of the form

```
call(Goal,Result,Control),
```

where `Goal` is the goal to be executed under it, `Result` is the result of the call (one of *success*, *failure*, and *stop*), and `Control` is an input stream through which control messages (sequence of *suspend*, *resume*, and *stop*) pass.

The leaf goals are direct or indirect descendant goals of the metacall just above them in the AND-tree. Note that if `Goal` fails, the metacall does not fail but instantiates `Result` to *failure*.

The metacall is used for user task control and resource management in the operating system.

## 3.4 Inter-PE Processing

In the FGHC programs, a user does not handle special message primitives for the inter-PE communication, but simply writes goals with pragmas and unifications. Inter-PE communication messages are automatically generated by the language execution system.

There are two major message types; one concerns FGHC goal management, such as goal sending and termination reporting; and the other concerns unification across the PEs.

When a goal with a pragma is executed, a message *throw_goal* is generated and sent to a specified PE. In the message, not only constants but reference pointers may be sent as arguments of the goal.

Guard unification always suspends for instantiating a value to a reference pointer across the PE boundary. At this time, a message *read_value* is sent to obtain a value pointed by the inter-PE reference pointer. When a value is returned through a message, *return_value*, a goal suspended by a unification suspension is resumed. When a unification of a value with an inter-PE reference pointer occurs in a body part, a message, *unify* is generated and sent with a value.

There are three types of inter-PE communication processing for sending and receiving these messages. They are essential for implementing a parallel FGHC execution system on a network-connected multiprocessor like the Multi-PSI.

1. AND-tree maintenance: any goal must belong to some AND-tree even if it has been sent from another PE. AND-tree maintenance must be done both when sending and receiving a goal. To reduce the maintenance cost and to simplify the maintenance algorithm, the *proxy-and-foster-parent scheme* is used [4].

2. Address translation: each PE has a different address space. When a reference pointer is passed from one PE to another, the internal address of the source PE is converted to a global identifier (ID). The address translation table between the internal address and global ID is maintained in each PE. This address translation mechanism is essential for separating local and global garbage collection.

3. Message composition and decomposition: the FGHC execution system must compose and decompose the inter-PE communication message. The main task is data type conversion between 40-bit intra-PE data and 8-bit data sequence for network transmission.

Processing of type 3 takes place in all message sending and receiving. 1 and 2 are performed according to message types and argument data types. We think that these processing cannot be removed from the execution system of an network-connected multiprocessor system. And they make the inter-PE communication expensive. Processing for a message often takes more time than a goal reduction indeed. This will be discussed in later sections.

# 4 Inter-PE Communication Time

The inter-PE communication may cost more expensive than the intra-PE processing in the network-connected multi-processor system like the Multi-PSI. This section shows the real value of the communication cost measured on the Multi-PSI/V1, an experimental prototype model. The measured cost is analyzed and component items of the cost are discussed. The analysis is also used to estimate the inter-PE communication cost of the Multi-PSI/V2, much improved model being developed.

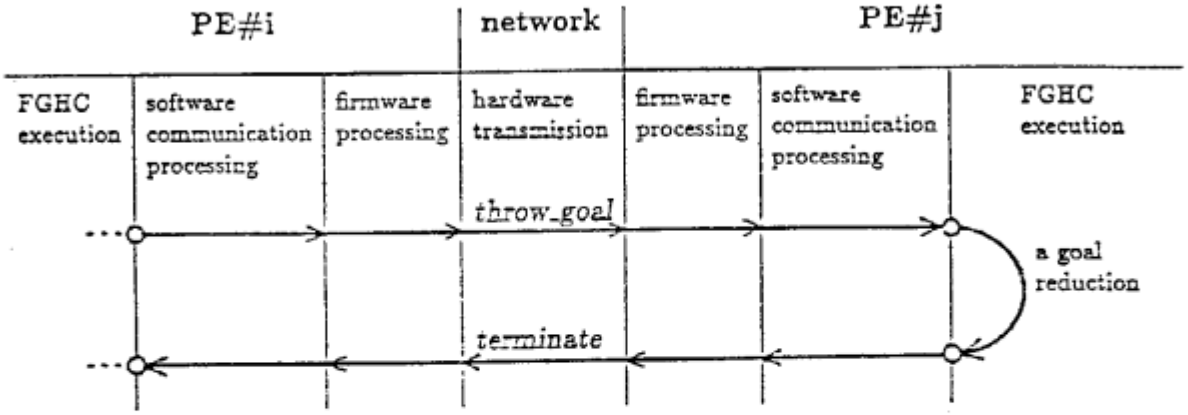


Figure 3: A network communication used in the measurement

## 4.1 Measurements on the Multi-PSI/V1

Inter-PE communication time was measured on the Multi-PSI/V1 for very simple cases [5]. Inter-PE communication time on the Multi-PSI/V1 consists of three items.

1. Hardware time: real network transmission time of a message from **PE#i** to **PE#j**
2. Firmware time: message sending and receiving control time in firmware of **PE#i** and **PE#j** . The main task is putting and getting a byte sequence of a message to/from the network interface hardware.
3. Software time: communication processing time of the FGHC execution system just described in the previous section in different three types of processing. The communication processing takes place in **PE#i** and **PE#j** for message sending and receiving.

The purpose of the measurement is to determine the component ratio of these roughly.

Measurement was made as follows. Adjacent PEs in the lattice network were used. A simple goal was sent from **PE#i** to **PE#j** . The goal terminated at once after one reduction on **PE#j** . The termination was reported from **PE#j** to **PE#i** . Figure 3 shows a network communication used in the measurement. Total time from goal sending to termination receiving was measured by the CPU timer on **PE#i** . The time of a goal reduction was also measured. The hardware time and the firmware time were calculated based on message sizes. Then the software time was calculated using these measurements.

When sending a goal with one integer argument, inter-PE communication time was measured as shown in Table 1. Ratios of each component time are also shown in the table using a reduction time as a unit time. "Sum of software time" means the total software time measured in **PE#i** and **PE#j** .

Table 1 shows that the ratios of hardware time and firmware time are much lower than software time. In other words, the hardware and firmware time are negligible and

Table 1: Inter-PE communication time

| item | time(ms) | ratio |
|---|---|---|
| one reduction time | 2.25 | 1 |
| sum of hardware time | 0.07 | 0.03 |
| sum of firmware time | 0.22 | 0.10 |
| sum of software time | 36.07 | 16.0 |
| total time | 38.32 | 17.0 |

almost all the inter-PE communication time is taken by software communication processing such as address translation, AND-tree maintenance and message composition in the Multi-PSI/V1. It should be emphasized that the communication bottleneck is not the hardware message transmission but the software message processing in this system. This characteristics is assumed in considerations through the paper.

The total time is seventeen times larger than the reduction time. That is, sending a goal and executing one goal reduction in a different PE costs much more than a goal reduction within a PE. Most of the cost arises from the software communication processing.

The table shows that the software time is sixteen times of a reduction time, both of which consume CPU time. It means that an inter-PE communication consumes much more CPU time than a goal reduction within a PE, and the communication degrades the system performance. The following section looks at the amount of degradation.

The software communication processing is not fast in current implementation because it is written in ESP. However the goal reduction processing is also written in ESP. It means that the ratio of the software time cannot be decreased much even if the firmware support becomes available for both the processing in Multi-PSI/V2. Because both the processing can be improved in speed. The ratio between a reduction time and the software time does not depend on the PE power, but depends on algorithms of both processing. Ratios of the hardware time and the firmware time can vary according to the system characteristics. The ratios for the Multi-PSI/V2 will be estimated in the following section.

## 4.2 Estimation for the Multi-PSI/V2

Both the goal reduction processing and the communication processing will be implemented in firmware on the Multi-PSI/V2. The goal reduction speed is expected to be around one hundred times of current speed. In the following discussion, it is assumed that both the goal reduction and communication processing become one hundred times faster. The hardware transmission time will be improved in five times, and firmware speed will be the same. Each the component time of the inter-PE communication and the time ratio is estimated using these assumptions. The Table 1 (measurements for the Multi-PSI/V1) is rewritten into Table 2 (estimation for the Multi-PSI/V2).

Ratio of firmware time has increased much and ratio of total time increased nearly

Table 2: Estimation of communication time for Multi-PSI/V2

| item | time(ms) | ratio |
|---|---|---|
| a reduction time | 0.02 | 1 |
| sum of hardware time | 0.01 | 0.5 |
| sum of firmware time | 0.22 | 11 |
| sum of software time | 0.36 | 18 |
| total time | 0.61 | 31 |

two times. However ratio of hardware time is the least as before. Two observations can be made from these results.

Ratio of firmware time has increased, however, summation of software time and firmware time still accounts for most of the total time. Both the software and firmware time consume CPU time. That is, the inter-PE communication speed is still CPU performance bound. The hardware speed is not a bottleneck as before. This characteristics means that the consideration of the communication overhead made for the Multi-PSI/V1 can be applied to the Multi-PSI/V2.

Ratio of total time in Table 2 becomes nearly two times of the ratio in Table 1, and most of it is taken by processing in PEs (not by the hardware transmission). It causes increase of the CPU time, that is, increase of the communication overhead. The communication overhead of the Multi-PSI/V2 will be nearly two times of the Multi-PSI/V1, in other words, the inter-PE communication of the Multi-PSI/V2 will cost nearly two times more expensive. This estimation will be used in section 8.

# 5 Communication and System Performance

Inter-PE communication may degrade the system performance as shown in the previous section. This section considers the amount of degradation and tries to describe into numeric formulas. Measurement scale for the system performance, communication efficiency and communication rate are presented in order to make numeric formulas.

## 5.1 System Performance

The performance of a single PE can be measured using benchmark programs. The unit of measurement is RPS (reductions per second). The ideal system performance can be calculated as

$$(PE\ performance) \times (number\ of\ PEs) = (ideal\ performance).$$

The ideal performance on a real system is obtained when the load of all PEs completely balances and there is no communication overhead. The system performance may deteriorate for these reasons:

1. Communication overhead: communication processing consumes CPU time. It makes execution time longer and decreases the RPS as a result.

2. Load imbalance: if PE loads are not uniformly distributed, the work rate of some PEs goes down. It decreases average work rate of the system.

3. Program sequentiality: if a program has much sequentiality, PEs sometimes wait for data available with no executable tasks but many waiting tasks. It makes the PE work rate lower and decreases the average work rate of the system.

In this section, it is assumed that PEs have uniform loads and programs have much parallelism (low sequentiality). Only the communication overhead degrades the system performance.

## 5.2 Communication Efficiency

Here, the communication efficiency is defined as *average throughput* of the communication message processing. It is a weighted average of each message processing throughput based on a certain *communication mix* in which each message appears at a certain rate. The measurement scale of the average throughput is *per second*.

For ease of treatment, average communication time is used instead of average throughput:

$$(average\ \ communication\ \ time) = \frac{1}{(average\ \ throughput)}$$

The meaning of the average communication time is average processing time for a message. the measurement scale is second.

Relative communication time based on average reduction time is more useful than absolute time. It is the relative communication performance based on the average reduction performance. Since the communication processing and the goal reduction are done on a same PE, the relative value shows the communication efficiency without depending on the PE power. The relative communication time is defined as follows.

$$(relative\ \ communication\ \ time) = \frac{(average\ \ communication\ \ time)}{(average\ \ reduction\ \ time)}$$

This is mainly used in the rest of this paper.

## 5.3 Communication Rate

The communication rate should be defined when discussing the communication overhead. The sense of the communication rate is a ratio between the amount of communication processing and the amount of goal reduction processing. For ease of measurement, the following definition is used.

$$(communication\ \ rate) = \frac{(total\ \ number\ \ of\ \ messages)}{(total\ \ number\ \ of\ \ reductions)}$$

This definition can be used either within a PE or all over the system. For the latter case, *total* means a sum of measurements in each PE. The latter case is mainly used.

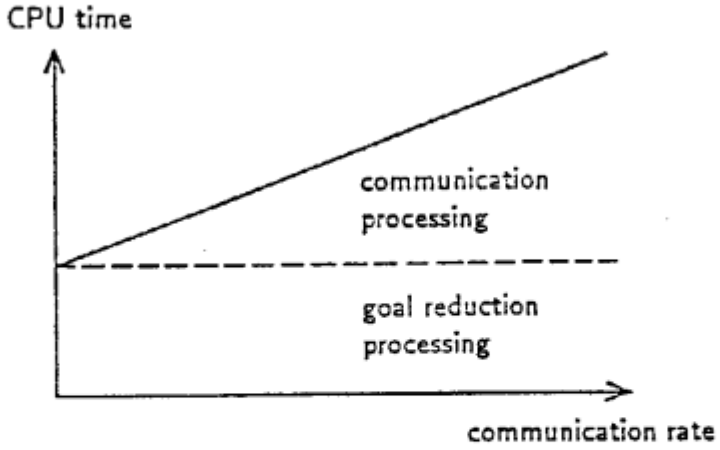


Figure 4: CPU time against communication rate

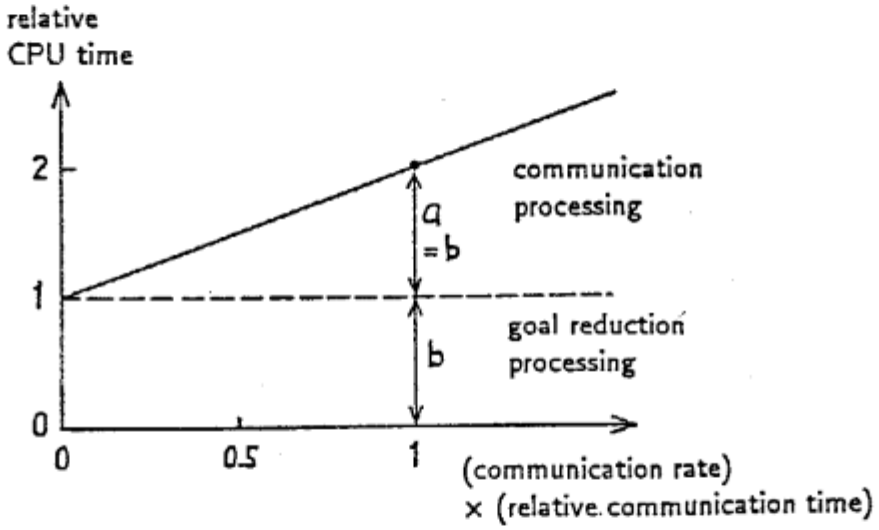


Figure 5: CPU time against communication overhead

## 5.4 Communication Rate versus System Performance

When the inter-PE communication consumes CPU time, the system performance decreases as the communication rate increases.

Figure 4 shows the relation between the CPU time of a PE (vertical axis) and the communication rate (horizontal axis). Assume various subproblems which contain the same reduction amount but a different communication rate. Figure 4 shows the CPU time for executing each subproblem. When a fixed amount of reduction processing is done on a PE, the CPU time for the reduction processing is constant (below the broken line in the figure). However, the CPU time for the communication processing increases in proportion to the communication rate (above the broken line) determined by the program characteristics.

The horizontal axis is changed in Figure 5. The horizontal axis becomes

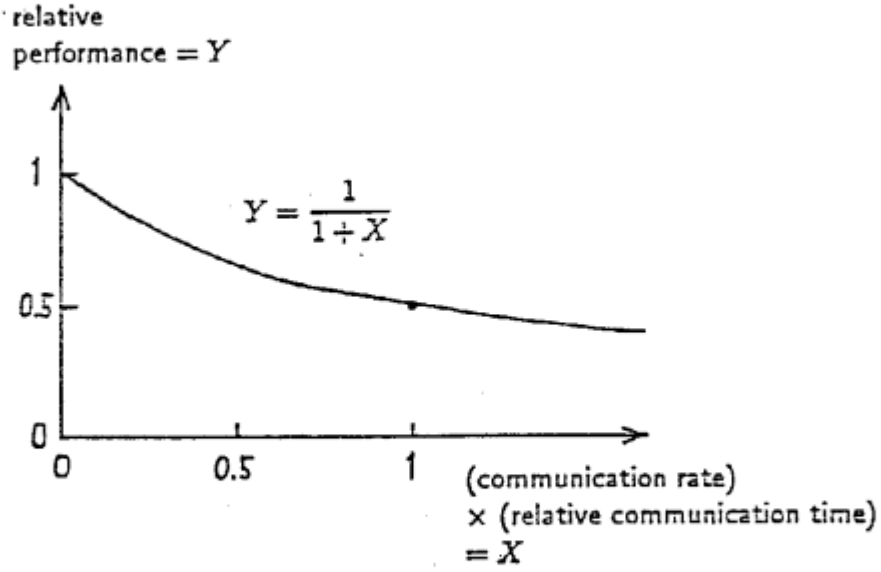


Figure 6: Relative performance against communication overhead

$$
\begin{aligned}
&(communication\ rate) \times (relative\ communication\ time) \\
&\quad = \frac{(total\ number\ of\ messages)}{(total\ number\ of\ reductions)} \cdot \frac{(average\ communication\ time)}{(average\ reduction\ time)} \\
&\quad = (total\ communication\ time)/(total\ reduction\ time) \\
&\quad = (relative\ communication\ overhead)
\end{aligned}
$$

instead of the communication rate. It is a kind of the communication overhead (time ratio). Depending on the change, the horizontal axis may have a scale. At the point where the horizontal axis is equal to one, the CPU time for the reduction processing and for the communication processing are the same. The CPU time is twice the best case (case of no communication), and the system performance is half of the best performance.

The point, *h-axis = 1*, corresponds to the case of an example in which a system and a program meet with the characteristics shown in the following example.

- System characteristics: relative communication time = 20. That is, the average message processing time is twenty times the average reduction time.
- Program characteristics: communication rate = $\frac{1}{20}$. That is, one message is sent every twenty goal reductions.

Figure 6 shows the relative performance of the system. The horizontal axis is same as Figure 5. It is calculated as $b/(a+b)$ from Figure 5. The system performance is also shown as a formula:

$$
\begin{aligned}
&(system\ performance) = (ideal\ performance) \cdot \frac{1}{1+X} \\
&\quad \texttt{where}\ X = (communication\ rate) \cdot (relative\ communication\ time)
\end{aligned}
$$

Figure 5 is a special case of $(ideal\ \ performance) = 1$ in this formula. The decline in system performance can be read easily from the figure. The relative performance of the system depends on the product of the *relative communication time* and the *communication rate*.

# 6 Work Rate and System Performance

The system performance may decline depending on the non-uniformity of the load distribution to the PEs and also depending on the program sequentiality. Both are completely program dependent in the Multi-PSI/V1[4]. Both the former and the latter decrease the average work rate of the system. The system performance is specified as shown below when there is no communication overhead.

$$(system\ \ performance) = (ideal\ \ performance) \cdot (average\ \ work\ \ rate)$$

--- without communication overhead

When there is the communication overhead, it is rewritten as follows combining the formula derived in section 5.4. This is the general equation specifying the system performance.

$$(system\ \ performance) = (ideal\ \ performance) \cdot (average\ \ work\ \ rate) \cdot \frac{1}{1+X}$$

where $X = (communication\ \ rate) \cdot (relative\ \ communication\ \ time)$

--- with certain communication overhead

The performance decrease rate is closely related to the network communication response. When the communication response goes down, the waiting time of PEs increases; for example, it takes more time to receive goals sent by other PEs at load distribution, and the suspension time of a goal increases when it waits for data to arrive from a different PE. In these cases, the waiting time of PEs increases and the average work rate of the system goes down.

It is a future study subject to analyze the relation among the system performance and communication response which must be considered with the load distribution method.

# 7 Measurements

This section shows measurement results of the several factors dealt with in the numerical formulas in sections 5 and 6. Relative communication time of the system, communication rate of sample programs, and average work rate of the system were measured on the Multi-PSI/V1 using sample programs which included the load distribution control. These measurements give parameter values to the numerical formula which specifies the system performance.

[4]The load distribution will be controlled by both the system and user programs in the Multi-PSI/V2. The system will manage the dynamic road balancing.

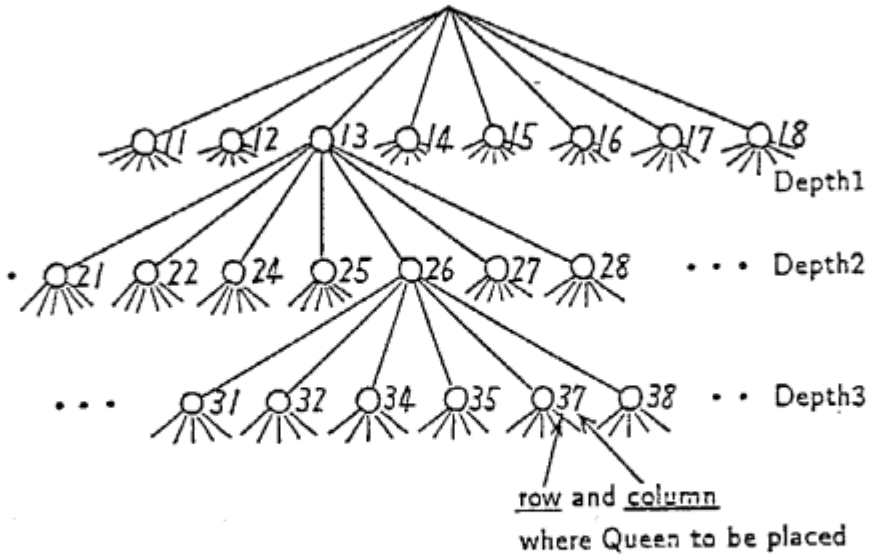


Figure 7: Execution tree of eight-queen problem

## 7.1 Sample Programs

The load distribution is entirely controlled by a programmer using the pragma in the Multi-PSI/V1. For the measurement, two eight-queen programs are used, each of which can change the distribution grain size of the load. Both programs are based on same original program but use different load distribution strategies. Both programs search all solutions of the eight-queen problem, expanding an execution tree shown in Figure 7.

1. Program 1 : the grain size for the load distribution can be specified using the depth of the execution tree from the root. All the nodes greater than or equal to the specified depth are distributed to PEs. The grain size for Depth1 is approximately 1/8 of the total computation load. Depth2, Depth3, $\cdots$ are $1/(8\times 7)$, $1/(8\times 7\times 6)$, $\cdots$ each. A kind of circular distribution [5] is used. The communication rate increases as the depth increases. The number of PEs can be specified as a program parameter.

2. Program 2 : it manages to maintain the distribution grain size uniform. The strategy is to estimate the number of tree leaves, $N$, and to distribute a lump of $N/M$ leaves to each PE, where $M$ is the number of processors. $N/M$ can be divided by $K$ specified by a program parameter in order to change the grain size.

   Source programs of both Program 1 and Program 2 are shown in the appendix.

## 7.2 Measurement Method

The following items were measured.

[5] See the source program in the Appendix.

Table 3: PE performance

| program | time(sec) | reductions | RPS |
|---|---|---|---|
| Program1 | 89 | 38882 | 0.44 k |
| Program2 | 105 | 38909 | 0.37 k |

- Total execution time
- Those listed below were measured for each PE
  1. Number of reductions
  2. Number of suspensions
  3. Real work time
  4. Number of communication messages (for sending)
  5. Number of *throw_goal* messages

An OS timer supported by the PSI was used to measure the total execution time and real work time of PEs. The number of reductions and suspensions were counted by adding counters in the FGHC execution system. The number of communication messages was determined to be the number of built-in predicate calls which were counted by a hardware supported evaluation counter in the PSI.

Special attention was paid to minimize the measurement overhead, because, when different overheads are added to the intra-PE processing and to the inter-PE processing, the communication response changes and the system behaviour may also change (especially for executing nondeterministic programs).

## 7.3 PE Performance

Table 3 shows the result of the sample programs executed on a PE. The number of reductions does not contain the built-in predicate calls.

Since Program 2 has more calls of arithmetic built-in predicates than Program 1, average processing time of one reduction increases and the RPS value decreases.

## 7.4 System Performance and Number of PEs

Figure 8 shows the RPS value for the execution of Program 1. The number of PEs varied from one to six. Five lines are drawn for variations of load distribution depth, from Depth1 to Depth5. The broken line shows the ideal performance which was calculated from $(RPS\ of\ a\ PE) \times (number\ of\ PEs)$.

There are two factors which degrade the system performance from the ideal value.

- Low average work rate caused by load imbalance
- Communication overhead (increase of CPU time for communication processing)

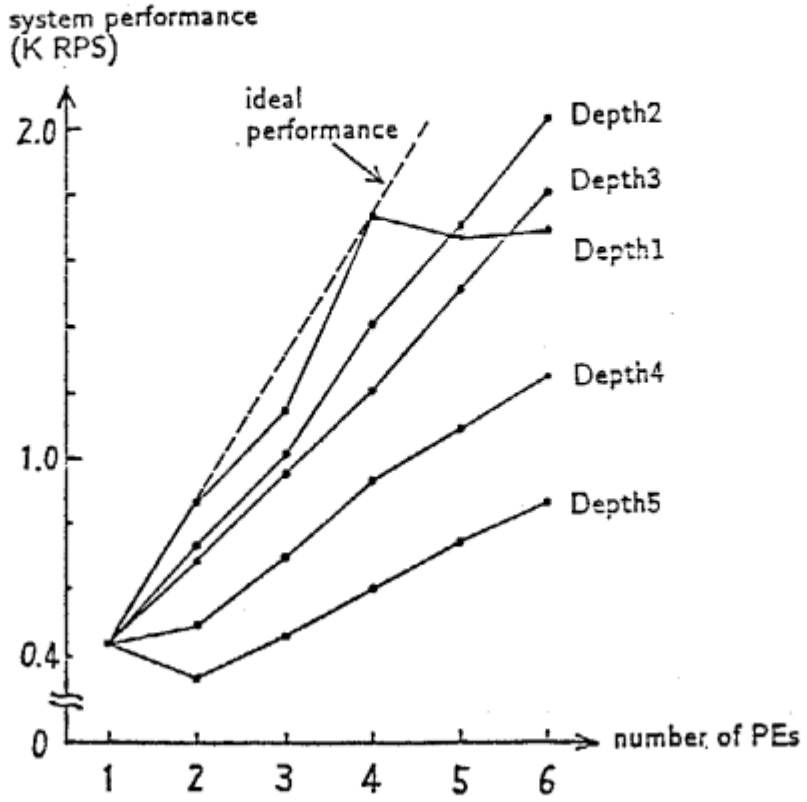


Figure 8: System performance against number of PEs (Program 1)

The sample program has low program sequentiality which almost never reduces the work rate.

At the point of $(number\ of\ PEs) = 6$, Depth2 achieves the highest performance. The distribution grain size of the load is the largest for Depth1. It may cause the load imbalance and may reduce the average work rate. The reason for the low performance of Depth1 is considered to be the load imbalance. On the other hand, the grain size is the smallest for Depth5. It may give a good load balance but may cause frequent communication. The reason for the low performance of Depth5 is thought to be the communication overhead. That is, according to the increase of the depth, the load balance improves, thereby improving the system performance, and the communication overhead becomes larger, thereby reducing the performance. The overlap of these factors is thought to give Depth2 the highest performance. These items will be made clear in the following subsections.

## 7.5 Work Rate

The work rate of a PE is obtained by

$$\frac{(real\ work\ time\ of\ a\ PE)}{(total\ execution\ time)}\ .$$

The average work rate of the system is an average of all the work rates of PEs. Measurements for Program1 are presented. Figure 9 shows the sum of PE work rates

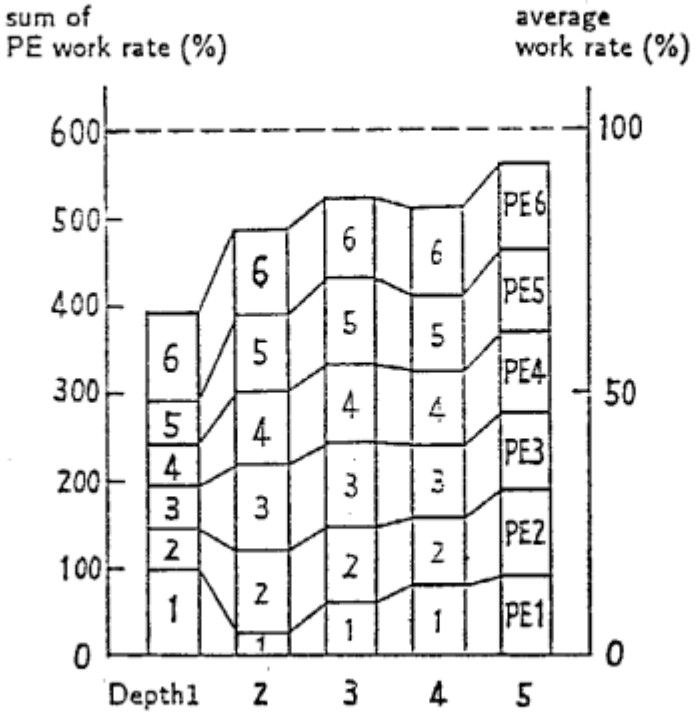


Figure 9: PE work rate and average work rate

(% value) in the left scale. That is, the % values of each PE work rate are piled on top of one another. When all PEs achieve the maximum work rate, the sum becomes 600%. The right scale shows the average work rate.

The work rate variation of PEs can be seen in the figure. Depending on the increase in depth, it can be seen that the work rate variation among the PEs becomes small and the average work rate increases.

## 7.6 Communication Rate

The communication rate of Program 1 is shown in Figure 10. The definition of the communication rate defined in section 5.3 is used. The vertical axis is the communication rate. The horizontal axis is the number of PEs. Five lines are drawn for each depth. The highest communication rate is more than 100 times the lowest rate.

## 7.7 Communication Cost

The relative communication time for a message is calculated in this section, as discussed in section 5.2.

Section 5.4 stated that the *relative communication overhead* can be obtained as a product of the *communication rate* and the *relative communication time*. However, the opposite procedure is taken in this section. The relative communication overhead is calculated from the measurements first, then the relative communication time is obtained dividing the overhead by the communication rate measured in section 7.6.

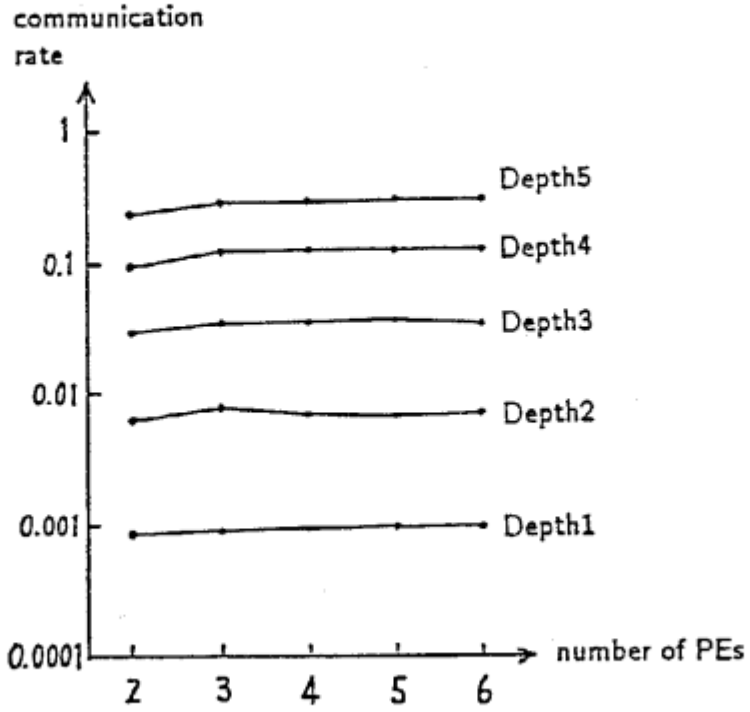


Figure 10: Communication rate against number of PEs

### 7.7.1 Relative communication overhead

Two factors cause the performance decrease of the system.

1. Low average work rate
2. Communication overhead

A graph of these is shown in Figure 11. It can be seen that the decline in performance depends on the factors. The vertical axis of the graph is the relative performance based on the ideal performance. The graph shows the case of Program1 using six PEs.

The upper value of the graph shows the performance in which *the communication overhead = 0* and the performance decrease only depends on the *work rate*. The value is the average work rate measured in section 7.5. The lower value of the graph shows the system performance (relative value) obtained from the measurements in section 7.4.

$$\begin{aligned} upper\ value &= (average\ work\ rate) \\ lower\ value &= \frac{(measured\ performance)}{(ideal\ performance)} \end{aligned}$$

The difference between the upper and lower value is thought to be the performance decrease caused by the communication overhead. The performance decrease rate can be specified as

$$\frac{(measured\ performance)}{(average\ work\ rate)}\ .$$

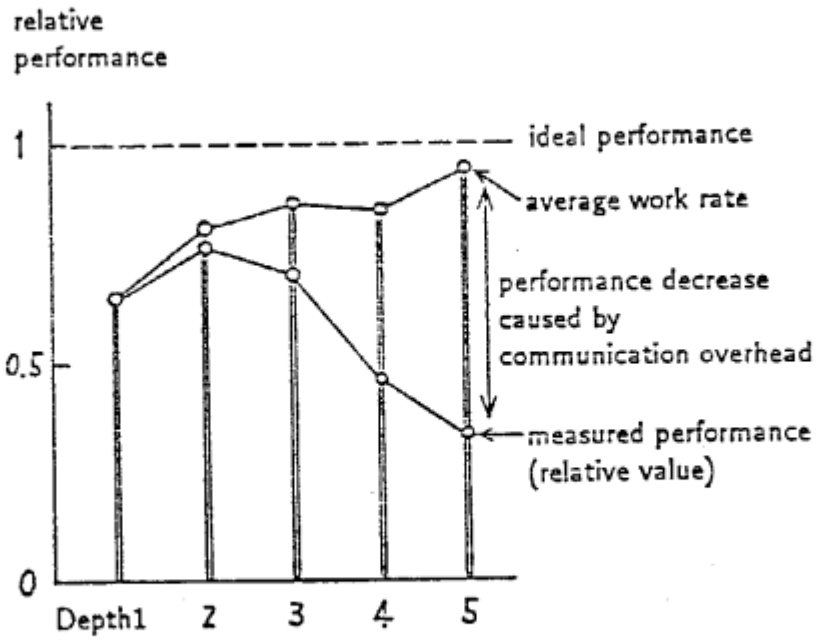


Figure 11: Relative performance based on ideal performance

The execution time increases at the rate of

$$\frac{(average\ work\ rate)}{(measured\ performance)},$$

which is the inverse of the performance decrease rate. The communication overhead is the increase of the execution time caused by the inter-PE communication. The relative communication overhead is the relative value of the communication overhead based on the execution time with no communication overhead. It can be specified as

$$(relative\ communication\ overhead) = \frac{(average\ work\ rate)}{(measured\ performance)} - 1\,.$$

The relative communication overhead corresponding to each depth of Figure 11 can be calculated by this equation and will be used to derive the relative communication time in the following section.

### 7.7.2 Relative communication time

The *relative communication time*

$$(average\ communication\ time)/(average\ reduction\ time)$$

is calculated using the *relative communication overhead* defined in the previous section and the *communication rate* in section 7.6. This value shows how many reductions can be done during average message processing time for a message, in other words, how much the message processing is heavier than the reduction processing.

Table 4: Relative communication time

| item | Program1 | Program2 |
|---|---|---|
| average | 6.3 | 7.7 |
| maximum | 7.4 | 9.8 |
| minimum | 5.6 | 7.1 |
| samples | *20* | *40* |

This formula was presented in section 5.4.

$$(communication\ rate) \times (relative\ communication\ time) = (relative\ communication\ overhead)$$

A formula of the relative communication time is obtained from this and a formula of the relative communication overhead defined in the previous section.

$$(relative\ communication\ time) = \frac{(relative\ communication\ overhead)}{(communication\ rate)} = \frac{\frac{(average\ work\ rate)}{(measured\ performance)} - 1}{(communication\ rate)}$$

The value of the relative communication time is calculated for all the depths of the sample program and for all the numbers of PEs, from two to six. The result for Program 1 and Program 2 is shown in Table 4.

The table shows that the message processing costs 6.3 times reduction processing for Program 1, and 7.7 times for Program2 on average. The difference is caused by two factors. One is the difference of goal arguments in the *throw_goal* message, eight arguments for Program 1 and twelve for Program 2. The other is the difference of the appearance ratio of the messages. The component ratio of the *throw_goal* message is 23% for Program 1 and 30% for Program 2. The *throw_goal* message requires the heaviest processing.

## 7.8 System Performance

A formula specifying the overall system performance was derived in section 6. It is shown again.

$$(system\ performance) = (ideal\ performance) \cdot (average\ work\ rate) \cdot \frac{1}{1+X}$$
$$\text{where } X = (communication\ rate) \cdot (relative\ communication\ time) \ .$$

The *ideal performance* depends on the computation speed (reduction processing performance) of the system. And the *relative communication time* depends on the communication processing efficiency of the system. These parameters were measured for the Multi-PSI/V1 in sections 7.3 and 7.7.2. On the other hand, the *communication rate* and the *average work rate* are determined by the program characteristics.

Then, the system performance of the Multi-PSI/V1 is described as follows rewriting the previous formula.

$$(system\ performance\ of\ Multi\_PSI/V1\ (K\ RPS))$$
$$= (0.44 \times 6) \cdot (average\ work\ rate) \cdot \frac{1}{1 + 6.3 \times (communication\ rate)}$$

--- for program 1

$$(system\ performance\ of\ Multi\_PSI/V1\ (K\ RPS))$$
$$= (0.37 \times 6) \cdot (average\ work\ rate) \cdot \frac{1}{1 + 7.7 \times (communication\ rate)}$$

--- for program 2

Note that the *ideal performance (K RPS)* and the *relative communication time* may vary depending on program characteristics. However, these equations are still very useful to have a rough estimation of the system performance when the average work rate and the communication rate can be assumed or estimated.

# 8 Discussions

Small grain size of the load distribution may increase the uniformity of the PE load among the PEs and may increase the average work rate of the system. On the other hand, it may cause high communication rate which increases the communication overhead and reduces the system performance. Programmers have to control the grain size of the load distribution in their programs in order to achieve well uniformity of the PE load (good load balance) and less communication overhead than a certain practical level.

The formula of the system performance in section 7.8 can be used to give a guideline for a practical lower bound of the grain size or an upper bound of the communication rate. When a programmer can decide the lower bound of allowable performance decline caused by the communication overhead, he can obtain the upper bound of the communication rate from the last formula in section 7.8. For an example, when the lower bound of the system performance is 50%, the upper bound of the communication rate is 1/7.7. It is the maximum communication rate which the programmer can handle in his program in order to achieve good load balance. According to this value, the programmer tunes up his load distribution algorithm.

The allowable maximum communication rate is approximately 1/7 for the Multi-PSI/V1, at most one inter-PE communication for seven reductions, when the lower bound of the system performance is assumed as 50%. The allowable maximum communication rate for the Multi-PSI/V2 will decrease into approximately 1/2 of the Multi-PSI/V1 which is roughly estimated comparing Table 1 and Table 2 in section 4.

Programmers have to maintain the communication rate of their programs to achieve good load balance and low communication overhead. A system implementor may be

required to prepare a programming paradigm and a programming system which can support a programmer writing a parallel program with low communication rate and well load balance. And a programming guideline should be also presented how much communication rate arises when a certain programming style chosen. This is a study theme to be challenged.

# 9 Conclusion

A network-connected multiprocessor like the Multi-PSI has such characteristics that a bottleneck of the network communication resides in the communication processing of the processing element (PE) not in the network transfer. The inter-PE processing costs around ten times more expensive than the intra-PE processing. These facts were shown through the measurements.

When the communication frequency increases in the Multi-PSI, the inter-PE processing consumes much CPU time reducing the system performance.

Measurement scale of communication efficiency and communication frequency were proposed to handle the performance decrease in a numerical formula. The relation between the system performance and the inter-PE communication was derived into a formula, which could be applied to a network-connected multiprocessor like the Multi-PSI.

The real value of these measurement scales were obtained by the measurement on the Multi-PSI/V1. Measurements completed the numerical formula which specified the relation among system performance, communication frequency and system work rate of the Multi-PSI/V1. Usage of the numerical formula was also discussed in order to tune the communication frequency of programs for low communication overhead and good load balance. It was also shown that the communication frequency on the Multi-PSI/V2 (high performance model under development) had to be maintained lower than Multi-PSI/V1.

The load balancing method, which affects the system work rate very much, was not dealt with in this paper. It is another important research theme for the future.

## Acknowledgments

The measurements was carried out by members of ICOT fourth laboratory and collaborating companies. The author gives grateful thanks to them. The author also thanks the chief of fourth laboratory, Dr. S.Uchida, and the Director of ICOT, Dr. K.Fuchi, for their advice and giving us the opportunity to do this research.

# References

[1] T. Chikayama. Load balancing in a very large scale multi-processor system. In *Proceedings of Fourth Japanese-Swedish Workshop on Fifth Generation Computer Systems*, SICS, 1986. Also in ICOT Technical Memo TM-276.

[2] K. L. Clark and S. Gregory. PARLOG: parallel programming in logic. *ACM Transactions on Programming Languages and Systems*, 8(1):1–49, 1986.

[3] A. Gotou and S. Uchida. Toward a High Performance Parallel Inference Machine –The Intermediate Stage Plan of PIM–. Technical Report TR-201, ICOT, 1986.

[4] N. Ichiyoshi, T. Miyazaki and K. Taki. A distributed implementation of flat GHC on the Multi-PSI. In *Proceedings of the Fourth International Conference on Logic Programming*. pages 257–275, 1987. Also in ICOT Techinical Report TR-230.

[5] H. Iwayama, K. Masuda et al. Connection Network Architecture of the Multi-PSI and Its Evaluation (in Japanese). Technical Memo TM-306, ICOT, 1987.

[6] H. Nakashima, K. Nakajima et al. Evaluation of PSI micro-interpreter. In *Proceedings of Compcon Spring 86*, pages 173–177, IEEE, 1986. Also in ICOT Technical Report TR-142.

[7] H. Nakashima and K. Nakajima. Hardware architechture of the sequential inference machine: PSI-II. In *Proceedings of 1987 Symposium on Logic Programming*, pages 104–113, IEEE Comuter Society, August 1987. Also in ICOT Technical Report TR-265.

[8] E. Shapiro. Systolic programming: a paradigm of parallel programming. In *Proceedings of FGCS'84*, pages 458–470, 1984.

[9] E. Y. Shapiro. A Subset of Concurrent Prolog and Its Interpreter. ICOT Technical Report TR-003, ICOT, Tokyo, Japan, January 1983.

[10] S. Takagi et al. A collection of KL1 programs, –Part 1–. Technical Memo TM-311, ICOT, 1987.

[11] K. Taki, M. Yokota et al. Hardware design and implementation of the personal sequential inference machine (PSI). In *Proceedings of FGCS'84*, ICOT, 1984. Also in ICOT Technical Report TR-075.

[12] K. Taki. The parallel software research and development tool: Multi-PSI system. In *Proceedings of France-Japan Artificial Intelligence and Computer Science Symposium 86*, pages 365–381, 1986. Also in ICOT Technical Report TR-237.

[13] S. Uchida. Inference machines in FGCS project. In *Proceedings of International Conference on VLSI'87*, IFIP TC-10, August 1987. Also in ICOT Technical Report TR-278.

[14] K. Ueda. Guarded Horn Clauses: A Parallel Logic Programming Language with the Concept of a Guard. Technical Report TR-208, ICOT, 1986.

[15] M. Yokota, A. Yamamoto, K. Taki, H. Nishikawa, and S. Uchida. The Design and Implementation of a Personal Sequential Inference Machine: PSI. ICOT Technical Report TR-045, ICOT, 1984. Also in New Generation Computing, Vol.1 No.2, 1984.

# Appendix

## Sample Programs

### Program 1 :

```
module queenx.

:- public go/3, t/3.

go(L,N,Depth) :- true | queen(L,Ans,N,Depth), write_results(Ans).

t(L,N,Depth) :- true | queen(L,_,N,Depth).

write_results([]) :- write(end) | true.
write_results([X|Y]) :- write(X) | write_results(Y).

queen(Original_list,Answer_list,N,Depth) :- true |
        queen_n(Original_list,[],[],Answer_list,[],6,N,Depth).

append([A|X],Y,Z) :- true | Z=[A|Z1],append(X,Y,Z1).
append([],   Y,Z) :- true | Z=Y.


queen_n([P|U],C,L,I,O,PE,N,Depth) :-
        P1 := ((PE + N - 6) mod N) + 7 - N |
        append(U,C,NN), c1_n(P,1,NN,L,L,I,X,PE,N,Depth),
        alloc(P1)@@queen_n(U,[P|C],L,X,O,P1,N,Depth).
queen_n([],[_|_],_,I,O,_,_,_,_) :- true | I=O.
queen_n([],[],   L,I,O,_,_,_,_) :- write(L) | I=[L|O].

queen_1([P|U],C,L,I,O) :- true |
        append(U,C,N), c1(P,1,N,L,L,I,X),
        queen_1(U,[P|C],L,X,O).
queen_1([],[_|_],_,I,O) :- true | I=O.
queen_1([],[],   L,I,O) :- write(L) | I=[L|O].


c1_n(T,D,N,[P|R],B,I,O,PE,NN,DX) :-
                T =\= P + D, T =\= P - D, D1 := D + 1 |
                c1_n(T,D1,N,R,B,I,O,PE,NN,DX).
c1_n(T,D,_,[P|_],_,I,O,PE,NN,DX) :- T =:= P + D | I = O.
c1_n(T,D,_,[P|_],_,I,O,PE,NN,DX) :- T =:= P - D | I = O.
c1_n(T,D,N,[],   B,I,O,PE,NN,DX) :- DX > 1, DD := DX -1 |
                queen_n(N,[],[T|B],I,O,PE,NN,DD).
c1_n(T,D,N,[],   B,I,O,PE,NN,1) :- true | queen_1(N,[],[T|B],I,O).
```

```
c1(T,D,N,[P|R],B,I,O) :- T =\= P + D, T =\= P - D, D1 := D + 1 |
                c1(T,D1,N,R,B,I,O).
c1(T,D,_,[P|_],_,I,O) :- T =:= P + D | I = O.
c1(T,D,_,[P|_],_,I,O) :- T =:= P - D | I = O.
c1(T,D,N,[],   B,I,O) :- true | queen_1(N,[],[T|B],I,O).

end.
```

Program 2 :

```
module queenz.

:- public go8/2.

go8(N,Depth) :- true |
        cul(Depth,N,40320,Base),
        queen([1,2,3,4,5,6,7,8],Ans,N,Base,5040,8)
%       , write_results(Ans)


cul(K,N,A,B) :-
        A >= N,A1 := A/N,A1 >= K,AA := A1/K |
        B = AA.
cul(K,N,A,B) :- A < N | B = 1.
cul(K,N,A,B) :- A >=N,A1 := A/N,A1 < K | B = 1.

queen(Original_list,Answer_list,N,Base,B,M) :- M1 := M-1 |
        queen_n(1,Base,Original_list,[],[],
                                Answer_list,[],6,N,Base,B,M1).

send_queen(Self,Border,List,C,L,I,O,PE,N,Base,Width,Level) :-
        Self > Border,
        B := Border+Base,
        P1 := (PE + N - 6) mod N + 7 - N |
        send_queen(Self,B,List,C,L,I,O,P1,N,Base,Width,Level).

send_queen(Self,Border,List,C,L,I,O,PE,N,Base,Width,Level) :-
        Self =< Border |
        alloc(PE)@@queen_n(Self,Border,List,
                                C,L,I,O,PE,N,Base,Width,Level).


queen_n(Self,Border,[P|U],C,L,I,O,PE,N,Base,Width,Level) :-
        Self > Border |
        send_queen(Self,Border,[P|U],C,L,I,O,PE,N,Base,Width,Level).
```

```
queen_n(Self,Border,[P|U],C,L,I,O,PE,N,Base,Width,Level) :-
        Self =< Border,
        S1 := Width+Self |
        append(U,C,NN),
        c1_1(P,1,NN,L,L,I,X,PE,N,Self,Border,Base,Width,Level),
        queen_n(S1,Border,U,[P|C],L,X,O,PE,N,Base,Width,Level).
queen_n(_,_,[],[_|_],_,I,O,_,_,_,_,_,_) :- true | I=O.
queen_n(_,_,[],[],   L,I,O,_,_,_,_,_,_) :- true | I=[L|O].

c1_1(T,D,N,[P|R],B,I,O,PE,NN,Self,Border,Base,W,Lev) :-
                 A1 := P + D,T =\= A1,
                 A2 := P - D,T =\= A2,
                 D1 := D + 1 |
                 c1_1(T,D1,N,R,B,I,O,PE,NN,Self,Border,Base,W,Lev).
c1_1(T,D,_,[P|_],_,I,O,PE,NN,Self,Border,Ba,W,Lev) :-
                 A1 := P + D,T =:= A1 | I = O.
c1_1(T,D,_,[P|_],_,I,O,PE,NN,Self,Border,Ba,W,Lev) :-
                 A1 := P - D,T =:= A1 | I = O.
c1_1(T,D,N,[],   B,I,O,PE,NN,Self,Border,Base,W,Lev) :-
                 Lev > 1,
                 WW := W / Lev, LL := Lev - 1  |
                 queen_n(Self,Border,N,[],[T|B],I,O,PE,NN,Base,WW,LL).
c1_1(T,D,N,[],   B,I,O,PE,NN,Self,Border,Base,W,1) :- true |
                 queen_n(Self,Border,N,[],[T|B],I,O,PE,NN,Base,W,1).

write_results([]) :- write(end) | true.
write_results([X|Y]) :- write(X) | write_results(Y).

append([A|X],Y,Z) :- true | Z=[A|Z1],append(X,Y,Z1).
append([],   Y,Z) :- true | Z=Y.

 end.
```