PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT.

Multi-Context Processing and Data Balancing Mechanism of the Parallel Inference Machine PIE64

Hanpei Koike and Hidehiko Tanaka
The Tanaka Lab., Dept. of Electrical Engineering,
The University of Tokyo, Hongo 7-3-1, Bunkyo-ku, Tokyo, 113 JAPAN

Abstract

PIE64 is a parallel inference machine. Its main purpose is fast execution of parallel knowledge information processing programs written in an enhanced committed-choice language FLENG++. PIE64 consists of 64 Inference Units, two high speed interconnection networks with an automatic load balancing facility and a host workstation.

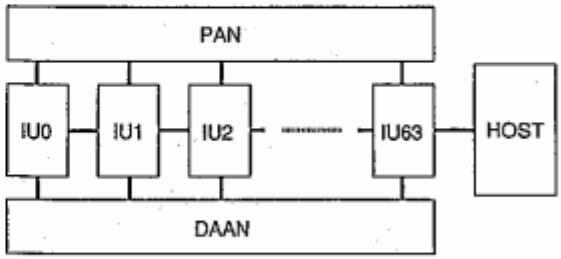
Each Inference Unit consists of an inference co-processor, network interface co-processors, a SPARC processor, and a local memory module. UNIRED, the inference co-processor, is a central part of the Inference Unit. It is a silicon interpreter of parallel logic programs which executes several goal rewriting primitives using pipelines. Two interconnection networks are Process Allocation Network and Data Allocation Network, respectively. Each interconnection network is a 3-stage network. Both networks have an *automatic load balancing facility*.

There are two key point to attain high performance on PIE64. One is how to maintain throughput of UNIRED against relatively slow remote data access and frequent process suspension. For this purpose, PIE64 adopts *multi-context processing*. UNIRED processes number of goals concurrently, and the context changes quickly on every remote data access. The other is how to avoid access contention on memory module. For this purpose, PIE64 uses *data balancing mechanism*. Automatic load balancing facility attached to Data Allocation Network is used to allocate data accessed frequently and to balance access ratio of each memory module. In this paper, these two support mechanisms for parallel processing are proposed and discussed.

1 Introduction to the Architecture of PIE64

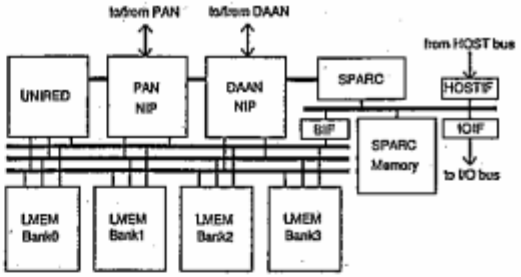
1.1 The Global Architecture

PIE64 is a parallel inference machine. It is designed mainly for fast execution of the parallel symbol manipulating programs written in the *reduced primitive set* committed-choice language FLENG [1] and its enhanced version FLENG++. 64 Inference Units (IUs) are connected using two high speed interconnection networks; Process Allocation Network (PAN) and Data Allocation Network (DAN). Both networks have an *automatic load balancing facility*. The global architecture of PIE64 is shown in Fig. 1.



IU - Inference Unit
PAN - Process Allocation Network
DAAN - Data Allocation/Accessing Network

Figure 1: The Global Architecture of PIE64



UNIRED - Unifier/Reducer
NIP - Network Interface Processor
LMEM - Local Memory

Figure 2: The Internal Architecture of the Inference Unit

1.2 The Internal Architecture of the Inference Unit

Each Inference Unit consists of:

1. Unifier/Reducer (UNIRED)
2. Network Interface Processor (NIP)
3. SPARC Processor

4. Local Memory (LMEM)
5. SPARC memory
6. Host interface
7. I/O interface

The internal architecture of the IU is shown in Fig. 2.

UNIRED and NIP are co-processors of the SPARC processor. They enhance the inference function and the parallel processing function of the IU respectively. UNIRED, NIP and SPARC share LMEM through three fast pipeline-arbitrated synchronous buses. UNIRED, NIP and SPARC exchange commands through a high speed command bus.

UNIRED is a support hardware for inference. UNIRED is a silicon interpreter of logic programs and executes goal rewriting primitive operations shown below using the goal rewriting pipeline.

- Passive Unification
- Active Unification
- Goal Reduction
- Overwrite Goal Reduction

When a goal and clause pair, or *context*, is supplied to UNIRED from SPARC processor, UNIRED applies Passive Unification to the context first. If the unification succeeds and the context is committed, UNIRED executes Active Unification and Goal Reduction using this context.

These operations are executed on *multiple contexts* concurrently supported by the fast context switching mechanism in order to maintain high throughput of pipeline processing against relatively slow remote data access and frequent suspension. In this way UNIRED processes alternative clauses for independent goals concurrently. UNIRED also supports primitive operations for garbage collection and data compaction.

NIP is a support hardware for parallel processing. NIP provides to the Inference Unit an abstraction of the network hardware and other remote Inference Units.

Basic operation of the NIP is to transfer contents of the Local Memory to/from other remote IUs through PAN and DAN on a request from the UNIRED or the SPARC processor. The destination of transfer is either addressed explicitly by the requester, or selected automatically to one whose load is the lowest. The latter is supported by the load balancing facility of the interconnection network described below. Commands shown below are data transfer commands:

- read1 read2 readn readx deref
- write1 write2 writen writex
- writel1 writel2 writeln writelx

Read command reads or copies data from the remote Inferece Unit. *Write* command writes data to the addressed remote Inferece Unit, and *writel* command writes data to the remote Inference Unit whose load value is the lowest. The suffix 1, 2 and n mean the size of the data; each corresponds to data types of a logical variable, a list cell and a vector.

The other function of the NIP is to exchange process synchronization messages and to manage goal suspension information. Commands shown below are provided for this purpose:

- suspend
- bind
- activate

Suspend command registers a suspended context id to the suspension list of the remote unbound variable. *Bind* command binds a value to a remote variable and generates *activate* commands towards each context suspended on the variable. More complex condition for goal waiting can be also realized by combining these basic primitives.

SPARC processor schedules the goal execution in UNIRED, performs load distribution and manages memory areas. It also executes system predicates and program modules written in conventional languages such as C.

1.3 The Interconnection Network

Both PAN and DAN are 3-stage networks and have an automatic load balancing facility [3, 4]. Load information of each destination is sent through the unused path of the network in reverse direction. On a load distribution request, the network automatically selects the connectable path to the processor whose load value is the lowest. PAN uses this facility to balance processing load among IUs. DAN uses this facility to balance allocation of data among IUs and to reduce access contention.

One network system hardware is implemented on five circuit boards. The size of the network is as compact as $25cm \times 50cm \times 50cm$ shown in Fig. 3. Two network systems and 64 IU boards will be assembled as shown in Fig. 4.

1.4 Host Processor

A Sun-4 workstation is used as a host processor of PIE64. Its main functions are program loading, user interface and system maintenance.

1.5 Current State of Hardware Design

The switching unit (SU) chip of the interconnection network has been implemented in a gate array. It is now under testing and performance mesuring. The network interface processor (NIP) chip and the network system

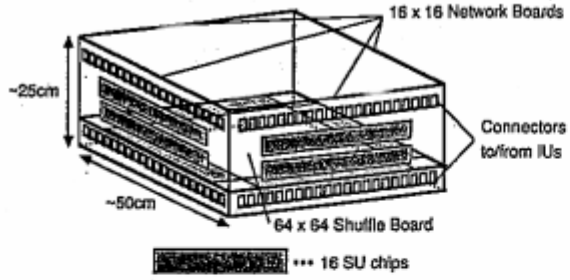


Figure 3: The Hardware Implementation of the Network

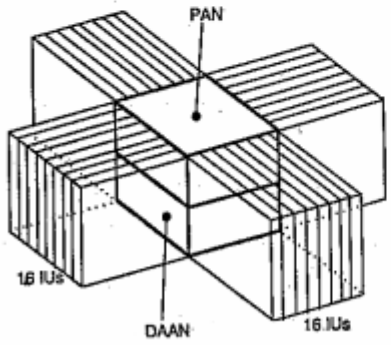


Figure 4: The Hardware Implementation of PIE64

hardware are under designing. We also start designing UNIRED chip and the Inference Unit board.

2 A Committed-Choice Language FLENG and its Execution Model

2.1 A Committed-Choice Language FLENG

FLENG [1] is a stream-AND type parallel logic programming language, or a *committed-choice language*. AND goals are processed in parallel. In order to avoid inconsistency of shared variables, two rules are adopted. The first one is that only one of clauses which succeeds in unification is committed and choiced to execute its body goals. Other clauses are discarded, and backtrack never occurs. The other rule is that head unification cannot instantiate variables of a goal. These variables are instantiated using an active unification predicate in the body goals after commitment of one clause. If some clause tries to instantiate an unbound variable in a goal during head unification, the unification suspends until the variable is instantiated by an active unification of the other goal. This mechanism is used as a way of process synchronization.

FLENG differs from other committed choice languages such as GHC [2] et al, in the sense that FLENG is *not* real logic programming language. Body goals have *no logical relationship*. A failure of a goal does not affect the execution of other goals. The body goals of the committed clause are just forked and processed in parallel. In this sense, a goal is nothing other than a process. Other difference is that FLENG has no guard goals. This simplifies the processing of the language. Semantical equivalence can be obtained by translation from programs using guard goals.

2.2 FLENG++/FLENG--: The Higher / Lower Level Languages for PIE64 System

We adopt FLENG as a base language of PIE64. To enhance both description power and processing efficiency of FLENG, we are designing high level and low level languages based on FLENG. These are called FLENG++ and FLENG-- respectively.

FLENG++ is a set of user languages for knowledge information processing. FLENG++ includes object-oriented language, knowledge base representation language and so on. All of them are designed based on a single base language FLENG to insure consistency of the system. FLENG++ is compiled into FLENG or FLENG--.

FLENG-- is a kernel language of PIE64. It is designed mainly for high performance parallel symbol manipulation. FLENG-- adopts several annotations for efficient parallel processing and memory management. FLENG-- is a super set language of FLENG. FLENG-- programs without any annotations is compatible with original FLENG.

2.3 The Execution Model of FLENG

The execution model of FLENG on PIE64 is shown in Fig. 6.

Execution processing of FLENG can be devided into:

1. Goal Rewriting
2. Inter Goal Synchronization
3. Goal Scheduling
4. System Predicate Execution

The basic execution cycle of FLENG on PIE64 is rewriting of a *Goal Frame* (GF). Goal Frame is a internal representation of a goal in PIE64. Goal Frame is an array of goal arguments on memory.

The internal representation of a clause is a Definition Template (DT). Some unifiable clauses are selected from alternative clauses for the goal by *clause indexing*. Each unifiable clause is paired with the goal. This goal and clause pair is called a *context*. Variables contained in a goal is called *global variables*, in a sense that they are shared among some goals. Variables occurred in a clause

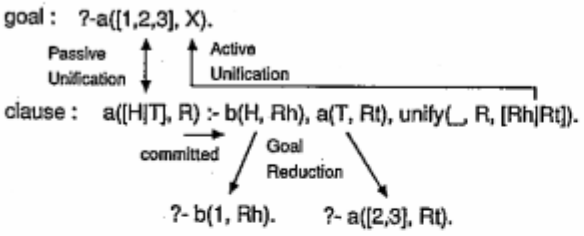


Figure 5: FLENG

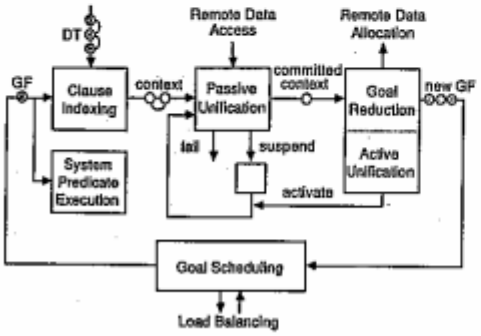


Figure 6: The Execution Model of FLENG on PIE64

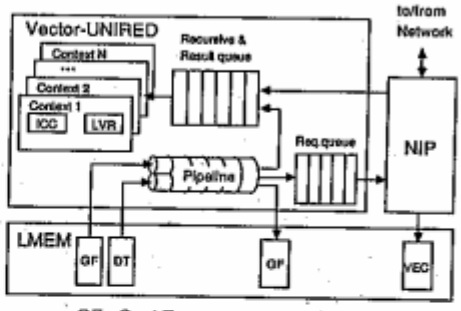
is called *local*. Local variables are *globalized* during Goal Reduction.

Passive Unification is applied to each context. Matching of the goal and the clause head occurs. The first context which succeeds in unification is committed, and other contexts are discarded. The unification may suspend, if it tries to unify an uninstantiated global variable with a value. The suspended context may be activated by Active Unification executed by other goal.

After commitment of the clause, Active Unification and Goal Reduction are applied to the *committed context*. Active Unification tries to instantiate global variables. Goal Reduction generates new goals from body goals of the committed clause. Body goals are copied onto the memory, while the instantiated local variables are replaced its value. Uninstantiated local variables are globalized. Structure data contained in the body goals are also copied onto the memory. The newly generated goals are added to the *goal pool*.

3 UNIRED: inference co-processor of PIE64

In this section, the primitive operations and architectural support for FLENG execution of the UNIRED, the inference co-processor of PIE64, are discussed.



GF - Goal Frame
DT - Definition Template
LVR - Local Variable Registers
ICC - Indexing Condition Cache

Figure 7: UNIRED

3.1 Primitive Operation of UNIRED

UNIRED is a support hardware for inference of the inference unit. It is a silicon interpreter of the parallel logic programs. It executes goal rewriting primitive operations shown below using pipelines. (Fig. 7)

- Passive Unification

  Passive Unification is a primitive operation for the head unification of a clause head with a goal. Passive Unification unifies all the arguments of the goal and the clause head. If both arguments are pointer to structure data, these pointers are inserted to *recursive queue*, and the nested structure data are unified recursively. If the structure data is on the remote memory, the remote data is copied onto the local memory using Network Interface Processor before unification.

  Passive Unification cannot instantiate globalized variables in the goal. The unification suspends, if it tries to instantiate an unbound global variable. The suspended context is added to the suspension list of the unbound global variable. The suspended context is activated, when the variable is instantiated by active unification executed by the other goal.

- Active Unification

  Active Unification is a primitive operation for the instantiation of global variables. Active Unification is executed after the commitment of the clause. Active Unification unifies two terms just like Passive Unification, except instantiation of global variable is permitted. Active unification may accompany activations of other suspended goals.

- Goal Reduction

  Goal Reduction is a primitive operation for process creation. Body goals of the committed clause are copied onto the local memory as new Goal Frames.

Variables in the clause remaining unbound are globalized. Structure data contained in the body goals are also copied.

- Overwrite Goal Reduction

  Overwrite Goal Reduction is a special version of goal reduction for memory saving. Overwrite Goal Reduction is applied to a special body goal on which several conditions are met. Overwrite Goal Reduction reclaims the literal area of the old Goal Frame on the local memory and reuses the area to generate a new Goal Frame. In this way, Overwrite Goal Reduction saves memory space substantially. Not only it saves memory space, but it also lowers the bus traffic between UNIRED and the local memory. The value of several arguments of the new Goal Frame tends to be the same as the arguments of the old Goal Frame and is not necessarily written again. In this way, reduction processing becomes faster than simple goal reduction.

4 Consideration on Performance Improvement

PIE64 executes logic programs using 64 UNIRED with goal rewriting pipelines on each Inference Unit. Viewing PIE64 as a set of pipelines, the factor of performance improvement of PIE64 can be divided into two factors shown below.

- Peak performance of the each pipeline
- Average work ratio of the each pipeline

While the former determines the performance of each Inference Unit, the latter determines the total performance of the parallel machine. The ideal parallel machine has the value of 1 for this factor. On a MIMD machine like PIE64, it is much more difficult to maintain this factor high, because:

- processing load, or the number of Goal Frames, may be unbalanced among Inference Units,
- access time of remote data is relatively slower than the pipeline cycle,

and

- access time of remote data may be prolonged by contentions occurred on both interconnection network and memory modules. This may cause problems such as *hot-spot* and *tree saturation* [5].

In order to attain high performance on a parallel machine, it is necessary to attach mechanisms to solve these problems. We adopt an automatic load balancing facility on the Process Allocation Network to solve the first problem.

The second and third problems can be restated as:

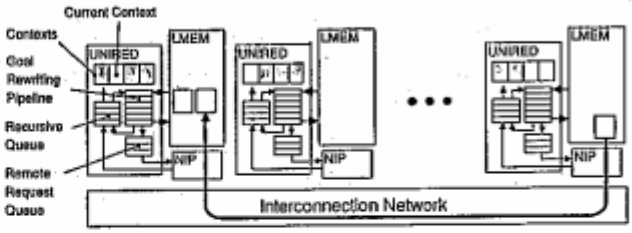


Figure 8: Multi-Context Processing

- the latency of remote data access must not affect the pipeline, and
- the throughput of remote data access should be kept high.

In this paper, we propose schemes which will maintain the work ratio of each pipeline of Inference Units. In the following sections, the execution mechanism of FLENG on PIE64 are presented first. Then the support mechanisms to solve the second and third problems shown above are proposed and discussed.

5 Multi-Context Processing

5.1 Multi Context Processing

UNIRED itself has no mechanism for direct remote data access such as remote variables or remote structure data. In such a case, UNIRED requests Network Interface Processor (NIP) to copy the remote data onto the local memory through the interconnection network. The interval between the request and the arrival of the data is rather longer than the pipeline cycle of UNIRED. Moreover, it is expected that context switching such as goal suspension occurs more frequently than we expected during the execution of a large program. Therefore, the throughput of the UNIRED may decrease substantially. In order to maintain the throughput of the UNIRED high, several support mechanism is necessary.

UNIRED can receive processing request of several contexts from SPARC processor. These contexts are processed concurrently inside the UNIRED. Context switching occurs every remote data access or suspension. Processing of another context is inserted between remote data fetch and the arrival of data. This *multi contexts processing* avoids decrease of the throughput of the UNIRED (Fig. 8).

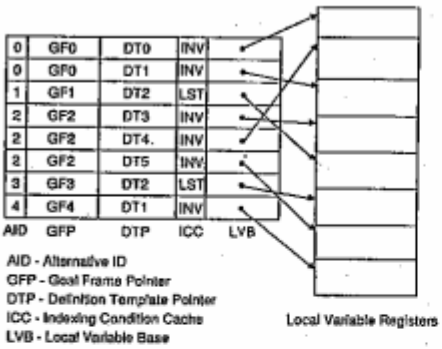


Figure 9: The Context inside the UNIRED

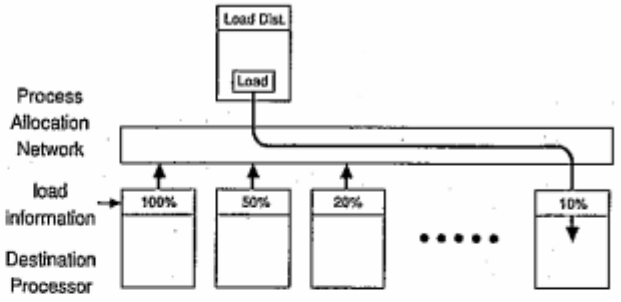


Figure 10: Automatic Load Balancing

5.2 Context inside UNIRED

The representation of context inside UNIRED is shown in Fig. 9.

1. AID is alternative ID. This is used to discard alternative contexts when some context succeeds in Passive Unification.
2. GFP and DTP is a goal frame pointer and definition template pointer, respectively.
3. ICC is a indexing condition cache. This is described later.
4. LVB is a local variable register base. LVB points local variable area of the context in the local variable register file.

5.3 Indexing Condition Cache

Some of FLENG clauses tend to call itself recursively for iteration. In such a case, same predicate is recursively called inside the UNIRED. If the interval after the Goal Reduction of the goal and the start of the next Passive Unification is too long, the pipeline of UNIRED cannot be used effectively. Most of the delay is the time to select unifiable clauses for the new goal by clause indexing. These operations are executed by SPARC processor. As long as the iteration repeats, the same indexing condition tends to be met.

The last indexing condition is saved in the *indexing condition cache* inside the UNIRED, and each time a goal is generated, it is inspected whether this condition is met or not. If the indexing condition of the current context is met by the new goal, the current context is reused by replacing the goal of the context with the new goal generated. This causes reduction of data transfer from SPARC to UNIRED, and the delay from goal reduction to the next passive unification can be shortened.

Normally, the processing of the context ends and the context is deleted after all the body goals are reduced. If one of the body goals has the same predicate as the original goal of the context, and the first argument of the goal matches the *indexing condition* of the context, the goal of the context is replaced with the new goal, and the processing mode of the context is rewound to Passive Unification, thus overhead of creation of new context is avoided by reusing context using indexing condition caching.

The entry of the Indexing Condition Cache of the context is one of:

- INV

  This shows that there are alternative clauses and the indexing condition cannot be met. The context is always discarded and the newly generated goal is always put outside the UNIRED.

- LST

  This shows that the clause of the context is the only clause which can be unifiable with the goal with the first argument of list. If the type of the first argument of the newly generated goal is list, the goal is overwritten to GFP entry of the context.

- VEC

  This shows that the context has the clause which can only be unifiable with the goal with the first argument of vector. If the first argument is vector, the context can be reused.

6 Automatic Load Balancing Facility of the Interconnection Network of PIE64

Besides normal destination-addressed communication, the interconnection networks of PIE64 have an *automatic load balancing facility*. The interconnection network automatically selects the route to the destination whose load is the lowest, by comparing load information sent on free paths in reverse direction (FIg. 10).

Examples of several kind of communication offered by the SU chip are shown in Fig. 11. In this figure, 1. shows

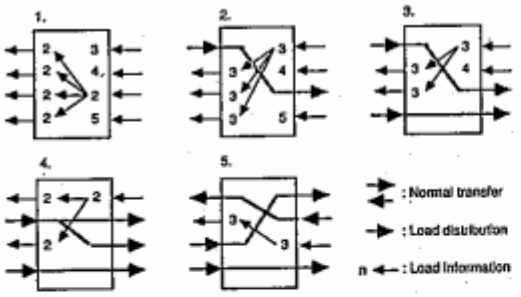


Figure 11: The Example of Communication using the SU chip

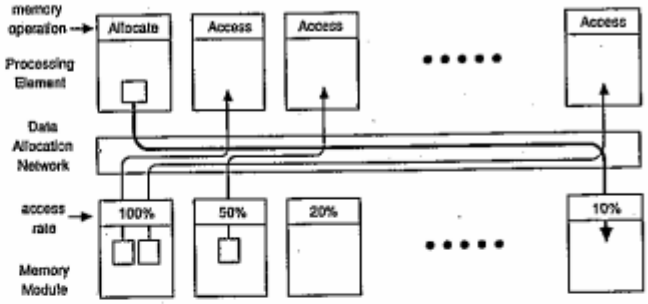


Figure 12: Automatic Data Balancing Mechanism

transfer of load information, 2. shows load distribution, 3. shows normal connection, 4. shows multicasting and 5. shows reverse communication.

7 Data Allocation Balancing Mechanism of PIE64

In this section, the data allocation balancing mechanism of PIE64 is discussed.

7.1 Dynamic Data Allocation

Besides dynamic creation of processes, execution of FLENG requires dynamic allocation of data such as variables and structure data. These data are allocated on the local memory of each Inference Unit. If data are not allocated appropriately, access contention occurs frequently and the throughput of remote data access decreases substantially. However, it is difficult in nature to allocate data optimally based on a static analysis of program. Accordingly, FLENG machine must support optimal data allocation onto local memory modules, just as optimal process allocation onto processors.

7.2 Hot-spot

A *Hot-spot* [5] is a memory location where many processes are concentrated on to access. Semaphore and loop index are examples of a hot-spot. If FLENG programs are written using such primitive, many hot-spots are created at many places in the memory modules during the execution of FLENG.

The hot-spot is caused by the fact that a memory module can be accessed by only one processor at a time basically. Since each IU of PIE64 has two network interfaces, it is possible that two processors access memory at a time at best, and the thing becomes slightly better, but the problem of hot-spot remains not solved completely. Uniformity of data access among memory modules is very important.

7.3 Tree Saturation

Access contention can be divided into network contention and memory contention. Hot-spot is a problem caused by memory contention. However, this memory contention causes another problem of network path contention; *tree saturation* [5]. Network request waiting to access hot-spot even interferes network request trying to access another cool memories. If 1000 processors access a hot-spot at a rate of one percent, the total throughput of the shared memory degrades to ten percent [5].

7.4 Automatic Thermal Balancing among Memory Modules

Since the hot-spot not only causes the memory contention, but it also causes the network contention and degrades the performance of the multi processor system substantially, it is very important to *balance the temperature* of each memory module.

An *automatic thermal balancing mechanism* is attached to PIE64 to balance the temperature of each memory module and to reduce memory contention. A *thermometer* is used to measure the temperature of each memory module. This value can be regarded as the load of the memory module. This temperature is used just like load information of a processing element, and this load value is sent using DAN (Data Allocation Network) which also has load balancing facility as the PAN (Process Allocation Network) has.

When each processor need to allocate data, the processor compares the temperature of its own local memory and the lowest temperature of the memory modules outside. If the temperature of outside is lower by certain threshold, the data are allocated on the memory module outside. Hence, data are automatically allocated on the memory module whose temperature is the lowest.

7.5 Thermometer of the Memory Module

How can we define the temperature of memory module?

We are going to use:

- memory access ratio in the past certain period

as the temperature of the memory module.

8 Conclusions

From a point of view that the parallel inference machine PIE64 is a set of pipelines, its performance improvement factor are analized. Based on this analysis, two schemes to maintain the work ratio of each pipeline high are proposed.

Multi-Context Processing processes number of goals concurrently, and the context quickly changes on each remote data access. This scheme compensate the effect of the remote access latency on pipeline.

Automatic Data Balancing Mechanism balances access ratio of memory modules by allocating data on the memory module whose access ratio is the lowest using automatic load balancing facility attached to the Data Allocation Network. This mechanism reduces access contention on memory modules, and maintains throughput of remote data access high.

Next step is to demonstrate the effectiveness of these schemes. We are going to measure the effectiveness of these schemes by simulation.

9 Acknowledgement

The authors would like to thank many people who helped us, especially the members of the Special Interest Group of the Inference Engine (SIGIE) of our laboratry. This work is supported by Grant-in-Aid for Specially Promoted Research of the Ministry of Education, Science and Culture.

References

[1] Nilsson, M. and Tanaka, H.: *Fleng Prolog - The Language which turns Supercomputers into Prolog Machines.* In Wada,E. (Ed.): Proc. Japanese Logic Programming Conference. ICOT, Tokyo, 1986. p 209-216. Proceedings also printed as Springer LNCS 264.

[2] Ueda, K.: *Guarded Horn Clauses.* D.Eng. Thesis, Information Engineering course, University of Tokyo, Japan, March 1986.

[3] Sakai, S., Koike, H., Tanaka, H. and Moto-oka, T.: *Interconnection Network with Dynamic Load Balancing Facility*, J. IPS Japan, Vol.27, No.5, 1986, p.518-524 (In Japanese).

[4] Koike H., Takahashi E., Yamauchi T. and Tanaka H.: *The High Performance Interconnection Network of Parallel Inference Machine PIE64*, Computer Architecture Symposium IPS Japan, 1988.

[5] Pfister, G.F. and Norton, V.A: *"Hot Spot" Contention and Combining in Multistage Interconnection Networks*, Proc. of the 1985 International Conference on Parallel Processing, IEEE, pp.790-795, 1985.