Workshop on Parallel Logic Programming, FGCS '94, pp.64-76

System Integration of the Parallel Inference Engine PIE64

Takuya ARAKI, Yasuo HIDAKA,
Hidemoto NAKADA,Hanpei KOIKE, Hidehiko TANAKA

Department of Electrical Engineering,
Faculty of Engineering, University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan.
Email: {araki,hidaka,nakada,koike,tanaka}@mtl.t.u-tokyo.ac.jp

Abstract

Uniform problems, such as scientific computation, can be executed in parallel using parallelizing compilers or data parallel languages these days. However, general applications, such as knowledge processing, are non-uniform, and highly-parallel execution of these problems is difficult with known technique. We designed Committed-Choice language Fleng for highly-parallel execution of non-uniform problems, and we have been developing the Parallel Inference Engine PIE64 to execute Fleng programs efficiently. The development of PIE64 system has almost finished, and it is now working. In the present paper we will describe integrated system of PIE64, including hardware and software, and its preliminary evaluation.

1 Introduction

Parallel computers are available these days. They succeed in the area of uniform problems such as scientific computation, using parallelizing compilers or data parallel languages; they can make use of a lot of data concurrency which uniform problems have.

However, general applications, such as knowledge processing and symbol processing, are non-uniform, from which we cannot expect much data concurrency. Thus highly-parallel execution of these problems is difficult with known technique.

We designed Committed-Choice language Fleng for highly-parallel execution of such problems. Fleng can extract control concurrency from non-uniform problems. We have developed Parallel Inference Engine PIE64 which executes Fleng programs efficiently. All of the hardware design and implementation of the system was done only by the staff in the university without any technical support from the industry.

The development of PIE64 system has almost finished, and PIE64 is now working. The present paper describes integrated PIE64 system and its preliminary evaluation.

Fleng is described in the next section. Hardware and software of PIE64 is described in section 3 and 4 respectively. Section 5 gives preliminary evaluation of the system and we describe conclusion in the last section.

2 Committed-Choice Language Fleng

We designed Committed-Choice language Fleng[1] to describe fine grained parallel symbol processing. GHC(Guarded Horn Clauses)[2], Concurrent Prolog[3], and PARLOG are famous as the same kind of languages. Fleng has no guard goals, and only a head realizes guard mechanism; Fleng is a simpler language than other Committed-Choice languages.

A Fleng program is a set of Horn clauses like:

$$\text{Head :- Body}_1\text{,Body}_2\text{,...,Body}_n\text{.}$$

The left side of :- is called a *head part*, and the right side is called a *body part* whose item Body_i is called a *body goal*.

Execution of a Fleng program is repetition of rewriting goals in parallel. For each execution of a goal, one clause whose head can match with the goal is selected, then the goal is rewritten into body goals.

This matching operation is called *head-unification*, and the rewriting operation is called *reduction*. This process is repeated by the goals newly created by reduction.

A variable in a goal is a single assignment variable; it cannot be assigned more than once. It has one of two states:*bound* and *unbound*. If variables needed for head-unification are unbound, the goal is *suspended*. Suspended goals are *activated* when the variables which caused suspension are bound.

Any goals can be executed in parallel, synchronizing by suspend-activate mechanism in a data-flow manner. This enables highly-parallel execution of non-uniform problems.

3 Hardware of PIE64

3.1 Overview of PIE64

Figure 1 shows the global architecture of PIE64 and its appearance working with 64 Inference Units.

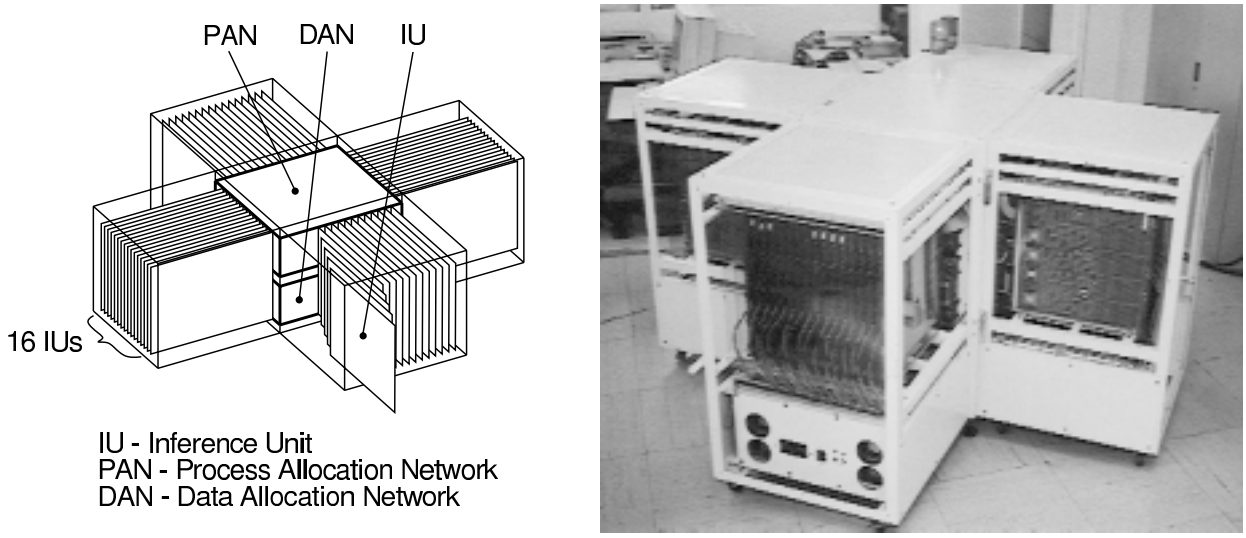

Figure 1: Global architecture and an appearance of PIE64: PIE64 consists of 64 processor elements called IU and two independent interconnection networks called PAN and DAN.

PIE64 is a parallel computer designed to execute Fleng program efficiently. It has 64 processor elements called Inference Unit(IU). All IUs are connected by two independent interconnection networks. These are circuit-switching multi-stage networks which have dynamic load balancing facility.

An IU contains three kinds of processors: UNIRED(Unifier/Reducer), NIP(Network Interface Processor) and MP(Management Processor). UNIRED is a processor for computation, NIP is one for communication, and MP is one for management. UNIRED and NIP are dedicated processors developed in our laboratory, and we use general purpose RISC processor SPARC as MP. These three processors cooperate to execute Fleng programs.

An IU has 4MB of local memory. The address space of PIE64 is global throughout all the IUs, i.e. PIE64 has NUMA(Non-Uniform Memory Access Time) type shared memory.

We will describe an interconnection network, an inference unit, UNIRED, NIP, and maintenance tools individually in the following subsections.

3.2 Interconnection network

The features of the interconnection network[4] of PIE64 are as follows:

- Circuit-switching
- Multi-stage network
- Dynamic load balancing facility

- Two independent networks

As a Fleng program runs, the network is mainly used to read remote variable and to transfer newly created goals. The length of variable is a few words, and that of a goal is 10 words or so. This indicates that frequently transferred data through the network are short. Thus it is more important to be of low latency than to achieve high throughput.

Reading remote data needs bidirectional communication, that is, an IU receives data from another IU after sending the address of the data to it. Because remote-memory-read operations are the dominant operation through the network, efficient bidirectional data transfer is important.

For these reasons, we adopted circuit-switching network. It enables low-latency communication and efficient bidirectional data transfer. Figure 2 shows the topology of it.

A notable feature of this interconnection network is dynamic load balancing facility. With this facility, the path to the least-loaded connectable IU is automatically established by the network itself. To realize this feature, load values are always sent through unused network paths in the reverse direction. Switches of the network compare load values to select the lowest one, and they tell the lowest value to the prior stage of the network. Selecting the path of the lowest value at each stage enables us to connect with the least-loaded IU. This facility utilizes resources unused for communication, thus it causes no overhead.

PIE64 has two independent interconnection networks of this kind: Process Allocation Network (PAN) and Data Allocation Network (DAN). They are distinguished only by its usage. PAN is used to transfer goals, and DAN is used to read remote memory and to allocate remote heap memory.

To realize this network, we developed 4×4 crossbar switches with 8 bit data width using gate array. This chip is called Switching Unit (SU). The SU chip has two modes: master mode and slave mode, intended to expand data width in a bit-slice manner. In PIE64, three slave chips are connected to a master chip to be used as a switch with 32 bit data width. Each network contains 192 SU chips.

An SU chip is implemented using an 8700-gate gate array in a 179-pin PGA package. Figure 5 shows an SU chip, together with other chips.

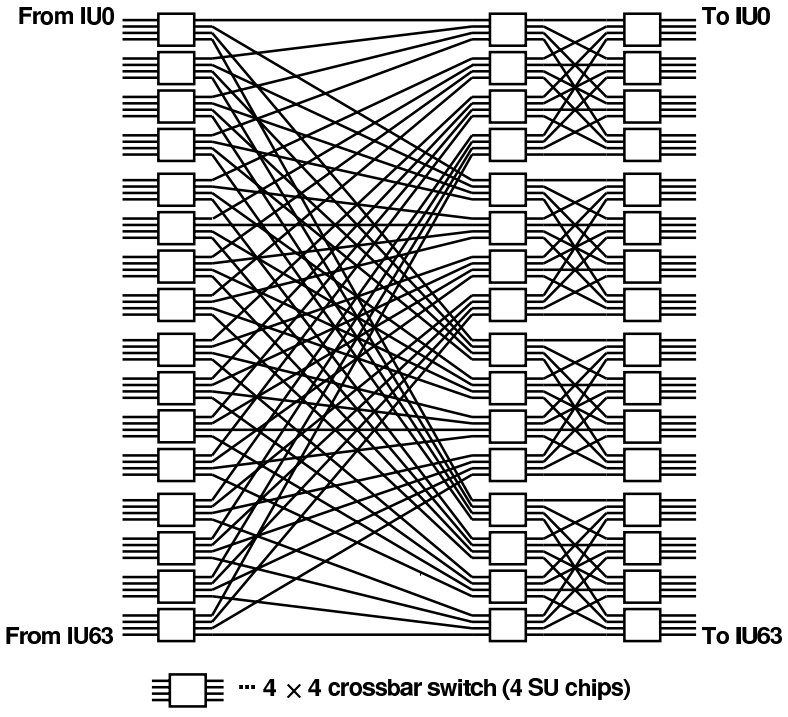

Figure 2: Topology of the network: This is a circuit-switching multi-stage network, which has dynamic load balancing facility.

3.3 Inference Unit

In parallel processing, not only actual computation, but also communication and control are important to achieve high performance. Thus parallel processing can be formulated as follows:

```
Parallel Processing = Computation + Communication + Control
```

We adopted a cooperative execution model of three dedicated processors according to this formulation.

Computation ... UNIRED(Unifier/Reducer)
A dedicated processor for executing Fleng program.

Communication ... NIP(Network Interface Processor)
A dedicated processor for communication and synchronization.

Control ... MP(Management Processor)
A processor for parallel management. We use SPARC as MP.

An IU consists of these three kinds of processors. These processors are connected by a command bus, and cooperate by exchanging commands, as shown in Fig. 3.

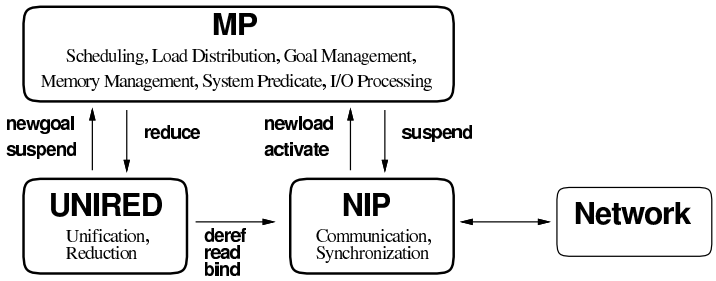

Figure 3: Cooperating Execution Model: Three kinds of processors exchange commands to cooperate in each IU.

The following is the outline of this cooperative process:
On receiving `reduce` with a goal from MP, UNIRED begins execution of a thread to perform unification and reduction of the goal. On a remote memory reference during execution, UNIRED sends `deref` or `read` to NIP. In order to assign a value to a remote variable, UNIRED sends `bind` to NIP. UNIRED delivers a new goal created by reduction to MP by `newgoal`. On a reference to an undefined variable, UNIRED sends `suspend` to MP, and suspends execution. When MP receives `suspend` from UNIRED, it creates some management information, and sends `suspend` to NIP to request to add a goal identifier to the suspension record of the variable. When NIP receives `bind`, NIP binds the value to the variable, and sends `activate` to MP.

An IU has 4MB of SRAM as local memory. These three processors share four banks of local memories through three memory buses, where bus arbitration and data transfer are pipelined to obtain high memory bandwidth.

Figure 4 shows the organization and an appearance of Inference Unit.

3.4 UNIRED

UNIRED[5] is a dedicated processor designed in our laboratory. It was designed to execute Fleng program efficiently. The features of UNIRED are as follows:

- A RISC architecture
- A tag architecture
- A dedicated instruction set for executing Fleng programs efficiently
- Multi-context processing

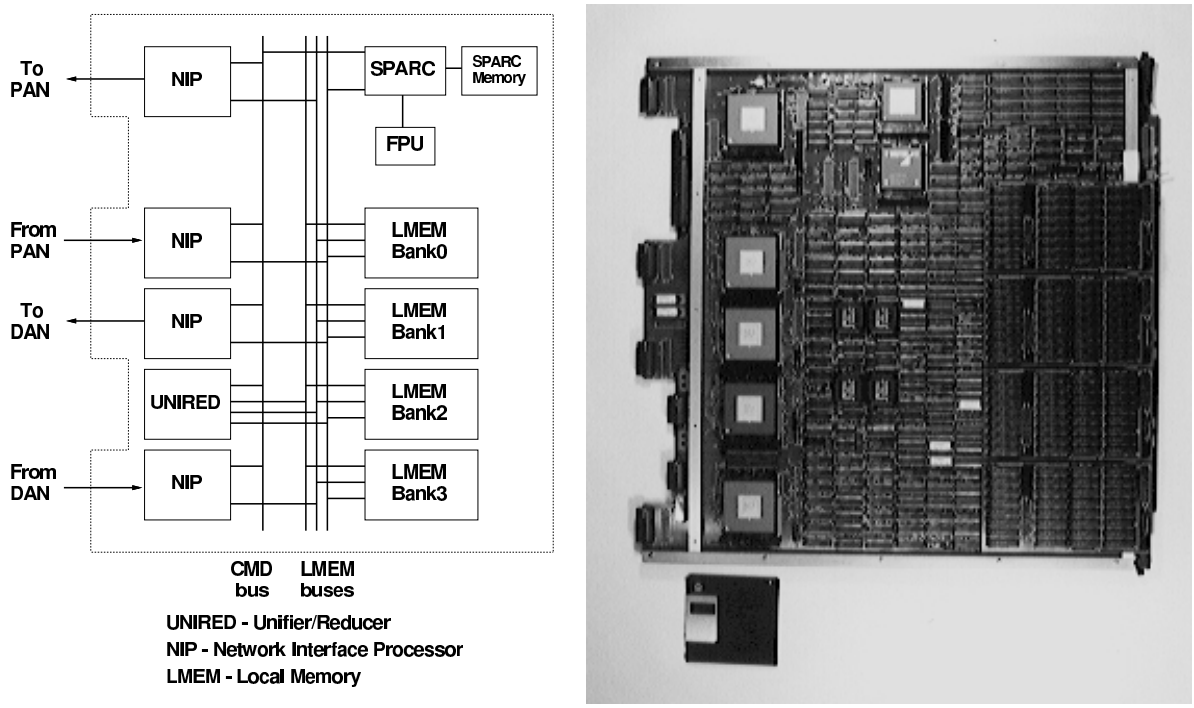

Figure 4: Inference Unit: MP, UNIRED, and NIP are connected by a command bus. These three processors share four banks of local memories through three memory buses.

Adopting a RISC architecture enables UNIRED to execute every instruction within a single cycle.

UNIRED has not only general purpose, but also special instructions for executing Fleng programs efficiently

We adopted multi-context processing, taking advantage of the fact that Fleng programs create many threads which can be executed in parallel. The internal pipeline of UNIRED is shared by multiple instruction streams. When one of the contexts waits for a result of some remote memory access, UNIRED fills its pipeline with other contexts dynamically. This enables us to hide the latency of remote memory access. With this feature, UNIRED acts as a pipeline-shared MIMD processor.

Moreover, the pipeline of UNIRED is not a cyclic pipeline; i.e. having pipeline interlocking capability, UNIRED can execute instructions in every cycle, even if there is only one active context. Thus, there is no significant performance loss to execute a sequential program. The number of contexts is four.

UNIRED accesses heap memory in a single address space throughout all the IUs. When UNIRED reads memory, it detects automatically if the address is local or remote. If the address is remote, it sends commands to NIP. Thus, remote and local memory reference need not be distinguished on the UNIRED machine language level. PIE64 has distributed shared memory or a NUMA architecture using this facility. Figure 5 shows an appearance of UNIRED.

3.5 NIP

NIP[6] is a dedicated processor designed in our laboratory. It was designed for communication and synchronization.

The functions of NIP are as follows:

- Read, write, bind, and dereference remote variables
- Suspend and activate goals
- Support garbage collection

NIP interfaces the interconnection network and an IU with such commands as `read`, `write`, `bind` and `dereference`. Sending these commands to NIP enables UNIRED and MP to execute actual computation in parallel with these operation.

`Suspend` and `activate` are executed with wired-logic sequencer. This enables low-cost synchronization. Figure 5 shows an appearance of NIP.

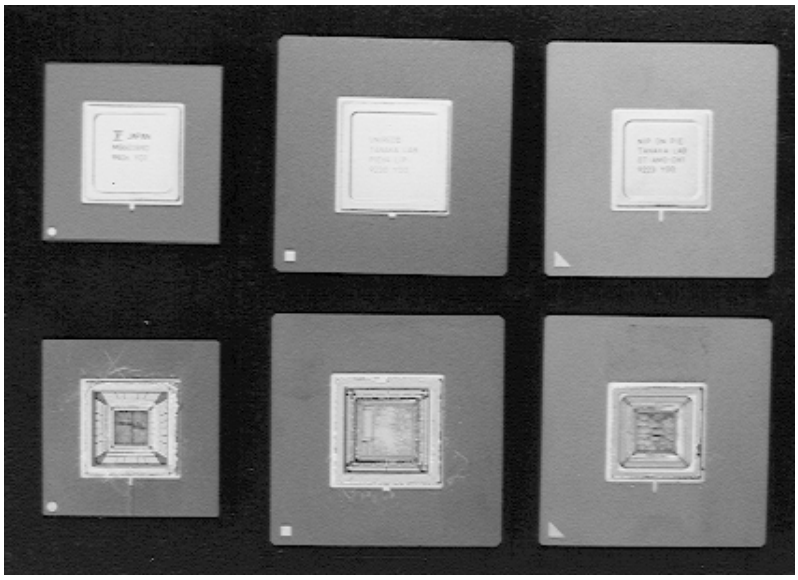

Figure 5: LSIs used in PIE64. From left to right, SU, UNIRED and NIP.

3.6 Maintenance tools

Easy maintenance is also important on large scale parallel computer development. PIE64 is equipped with maintenance hardware TAKO[7], which works as a logic analyzer, network maintenance hardware, a host interface and a clock generator. TAKO played an important role in completing PIE64.

TAKO contains a network scan interface and a logic analyzer with eight probes. The scan interface can read and set state of crossbar switches, with the network stopped. The logic analyzer is used both for a network and for an IU; it can watch both data on networks and data on buses of IUs.

We developed TAKO supporting software which give us graphical user interface. It shows acquisition pattern of the logic analyze in time chart format, and enables us to see and set network state graphically.

4 Software of PIE64

4.1 Compiler

A Fleng compiler[8] is written in Fleng itself and can be executed in parallel. We are now developing a new compiler which has following features:

- Static load partitioning
- Granularity control

Each feature is explained as follows.

Static load partitioning On Fleng execution model, all goals are executed in parallel synchronizing in data-flow manner. But assigning all goals to different IUs causes unnecessary communication, because data-dependent goals cannot be executed in parallel. The compiler analyzes data flow of a program, and assign goals and data which depend on each other to one IU. This reduces communication maintaining concurrency.

Granularity control Fine grain execution can extract sufficient concurrency, but it causes large overhead such as goal management. When sufficient parallelism is extracted and all IUs have sufficient load, there is no need to extract more parallelism.

Thus, the compiler compiles a program into two granularity codes: a fine grain code and a coarse grain code. Parallel management kernel selects which code to use according to the parallelism at run time. When the parallelism is too low to fill all IUs with load, the kernel uses a fine grain code, and when all IUs are sufficiently loaded, the kernel uses a coarse grain code.

This feature is now under development.

4.2 Parallel management kernel

Parallel management kernel[9] is executed by MP, and it plays a part of `Control` out of three elements of parallel processing: `Computation`, `Communication`, and `Control`. It is a counterpart of operating system kernel, and performs low-level system management which cannot be written on Fleng level.

It has roughly two kinds of facility: one is ordinary management needed even by sequential processing, and the other is management which is characteristic of parallel processing. The former includes management of clauses, I/O, and memory, and the latter includes goal scheduling, load partitioning, and switching compiled code.

Significant functions of the kernel are load partitioning and selecting compiled code. By dynamic load balancing facility of the interconnection network, an IU can know the load value of the least-loaded IU. The value indicates if all IUs are sufficiently loaded. When all IUs have sufficient load, the kernel does not send newly created load to other IUs, and executes them on the local IU; and it uses a coarse grain code if the program was compiled into two granularity codes. It eliminates excessive concurrency, and reduces communication and goal management overhead.

4.3 Debugger

4.3.1 Multi-window Debugger HyperDEBU

Debugging parallel programs is more difficult than sequential programs, because in a parallel program many processes run simultaneously and interact with each other. We developed a multi-window debugger HyperDEBU[10][11] to debug highly parallel Fleng programs.

HyperDEBU visualizes execution of a program as processes communicating each other. Multi-window system provides various views which enable a programmer to observe and operate complicated control/data flow of programs.

HyperDEBU is written in Fleng itself and can be executed in parallel. Figure 6 shows an overview of HyperDEBU.

4.3.2 Performance debugging tool Paf

A cause of inefficiency of program is called a *performance bug*. Paf[12] is a debugging tool to debug performance bugs of Fleng programs.

Paf executes a Fleng program on infinite virtual processors and traces it, so that the trace indicates program's behavior which is independent of execution environment. Paf can display concurrency and critical path of a program, and programmers can debug performance bugs using these information.

Figure 7 shows an overview of Paf.

4.4 Application software

To evaluate system properly, we have to use practical applications other than toy programs. Developing application software is important also for this reason.

We developed non-monotonic reasoning system as application software. This is composed of production system based on RETE algorithm and ATMS(Assumption-based Truth Maintenance System).

We are also developing formula manipulation system like REDUCE.

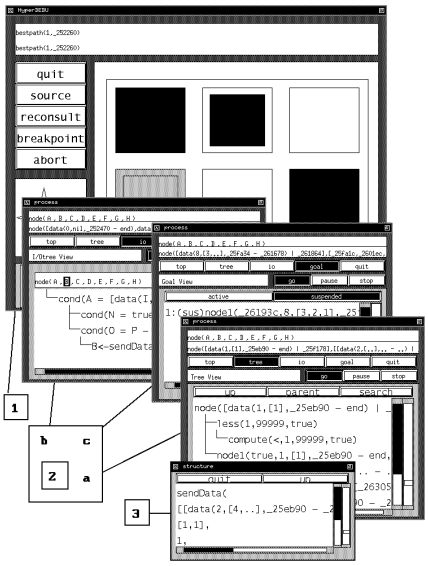

Figure 6: Multi-window Debugger HyperDEBU

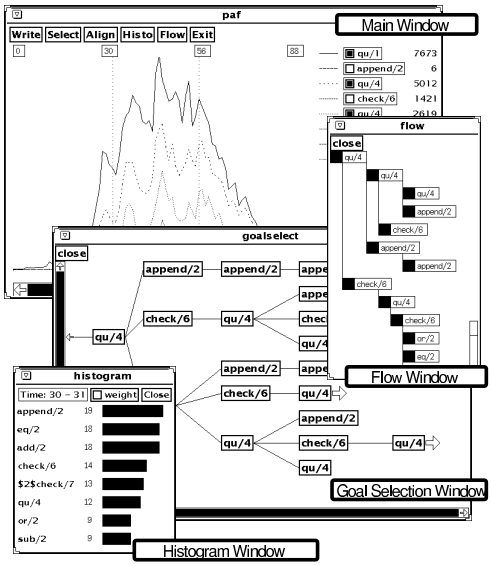

Figure 7: Performance debugging tool Paf

5 Preliminary evaluation

5.1 Evaluation condition

The compiler used here is rather naive without static load partitioning or granularity control mentioned above; Programs are executed according to Fleng execution model. The evaluation of static load partitioning is mentioned in [13].

Used programs for evaluation are 'n-queens' and 'primes'. 'N-queens' is a program to solve 'n-queen problem' and 'primes' is a program to find prime numbers. 'N-queens' has a lot of concurrency and 'primes' has less concurrency than 'n-queens'.

Clock frequency of PIE64 is 5MHz, which is half of designed frequency.

5.2 Speedup

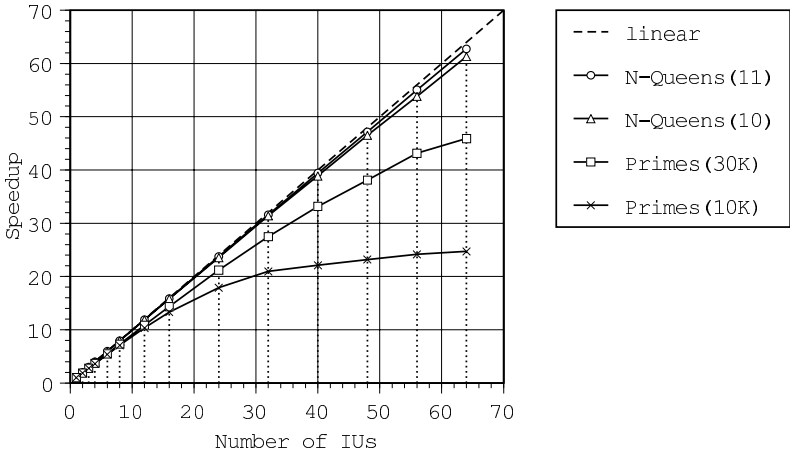

Figure 8: speedup

Figure 8 shows the speedup of programs. Execution time used here does not include GC(Garbage Collection) time.

The speedup of each program depends on its concurrency. 'N-queens' has an advantage over 'primes'

in this point. Because of high concurrency, speedup of 'n-queens' is almost linear. The 'primes 10k' has less concurrency than 'n-queens', and the speedup is limited.

5.3 Comparison with other computers

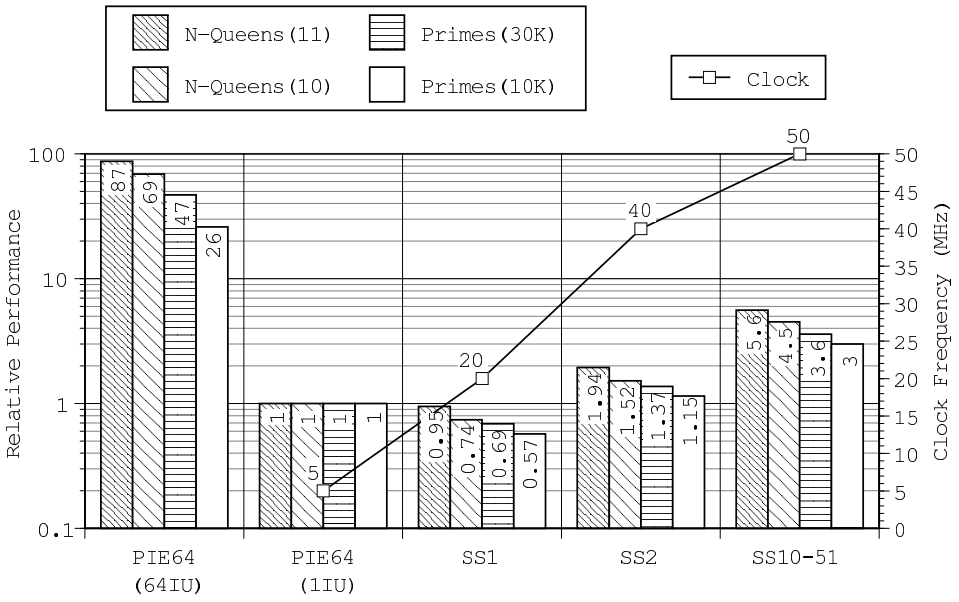

Figure 9: Relative performance

Figure 9 shows performance of PIE64 compared with other computers. SS1, SS2, SS10-51 means SPARC station 1, SPARC station 2, SPARC Station 10 model 51, respectively.

To measure performance of workstations, we used Fleng programs here. The Fleng compiler on workstations compiles a Fleng program into a C program, then a C compiler compiles it to an object code. The Fleng system on workstation loads it to execute. The Fleng compiler of PIE64 and that of workstations are same, except for a code generation part.

Execution time used here includes GC time. Performance is normalized by the speed of PIE64 working with one IU. Note that the y-axis of the graph is log scale.

This graph indicates that UNIRED executes Fleng programs efficiently for its clock frequency; UNIRED runs faster than SS1 whose clock frequency is four times as high as that of UNIRED. This high performance of UNIRED is due to its architecture specialized to Fleng execution, such as a dedicated instruction set, a tag architecture, multi-context processing.

PIE64 with 64 IUs executes Fleng program much faster than these workstations.

Speedup of 'n-queens' is super linear. This is because the rate of GC time is large in these programs ($10 \sim 30$ %), and not only GC time but also number of GC occurrence decreases with increasing number of IUs (and amount of memory).

5.4 Comparison with other programming languages

Figure 10 shows performance comparison with other programming languages. We used HCL (Ver. 1.6 Rel. 2.3) as the language processor of Lisp, SICStus Prolog (Ver. 2.1) as that of Prolog, cc of SunOS (Rel. 4.1.1) as that of C. Used program here is 'n-queens', which we wrote in these languages using same algorithm. Compiled programs are used to measure execution time. Used workstation is SPARC station 1, whose generation is same as UNIRED.

Performance is normalized by the speed of PIE64 working with one IU. Note that the y-axis of the graph is log scale.

Though the programs compiled by current compiler run slower than Lisp and Prolog, hand-optimized codes run almost as fast as Lisp and Prolog. This indicates that there is still room for improvement of Fleng compiler. C shows the best performance; but PIE64 with 64 IUs can run much faster than it.

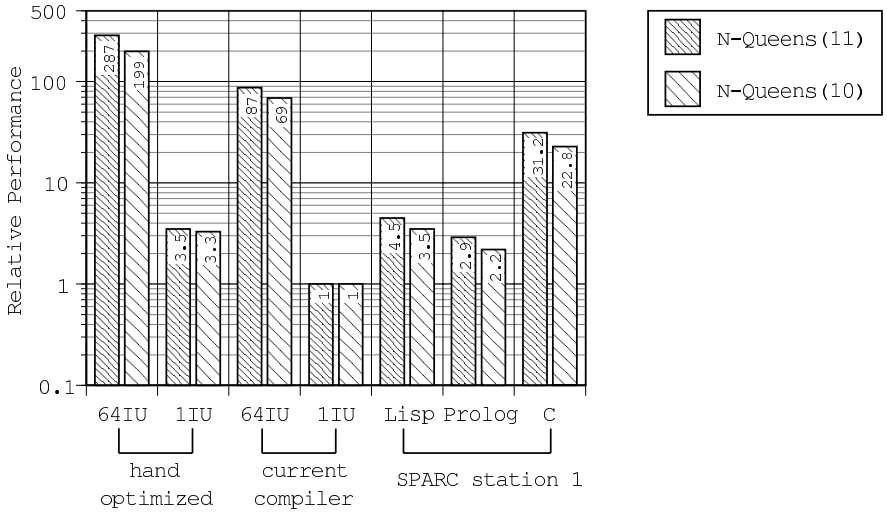

Figure 10: comparison with other programming languages

5.5 Execution time of each processor

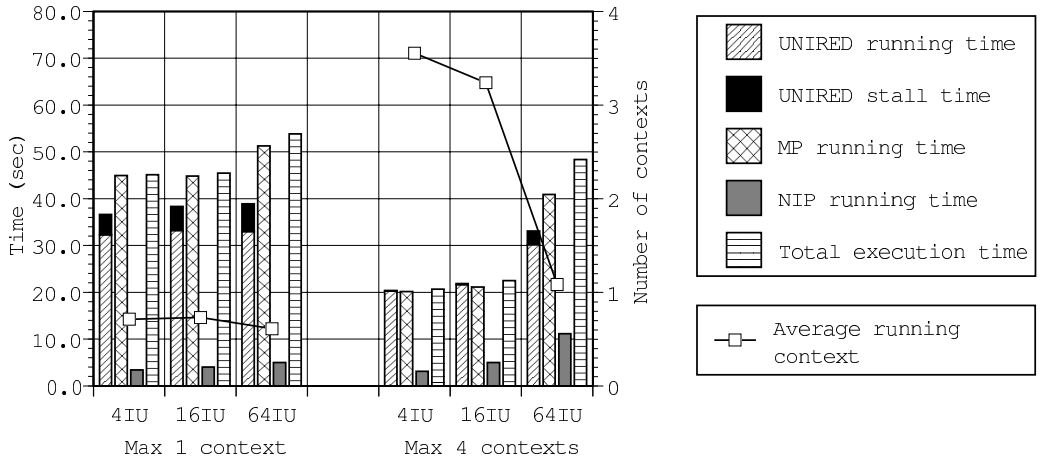

Figure 11: Execution time of each processor

Figure 11 shows execution time of each processor of IUs. Executed program here is 'primes 10k'.

Execution time of this graph is sum of all IUs' execution time. This would not vary with number of IUs, if there were no overhead of parallel execution. Actually, the overhead increases total execution time with increasing number of IUs. Left side of the graph is execution time with one context of UNIRED, and right side is that with four contexts. 'UNIRED stall time' is the time when UNIRED waits for a reply from NIP and stops.

When the program is executed with small number of IUs, such as 4 or 16, four contexts execution makes total execution time short and get rid of almost all the stall time of UNIRED. This is because multi-context processing of UNIRED can reduce probability of pipeline interlocking and hide latency of remote memory access.

When the program is executed with 64 IUs, total execution time and stall time increases. This is because the concurrency of the 'primes 10k' is too low to fill all UNIRED contexts of all IUs with loads; average running context also decreases with increasing number of IUs.

6 Conclusion

This paper presented the integrated PIE64 system and its preliminary evaluation.

Main features of PIE64 hardware are:

- Low latency interconnection network,
- Cooperative model of three processors,
- Multi-context processing of main processor.

Software on PIE64 includes Fleng language processor, programming environment, and application. We preliminarily evaluated performance of PIE64, and showed the high performance of PIE64.

Used programs to evaluate the system were toy programs such as 'n-queen' and 'primes'. This is because the current compiler is not perfect yet. We are now completing the compiler. We will evaluate the system properly using more realistic application.

The compiler used here is rather naive, and there is still room for improvement. We are now developing a more efficient compiler which does granularity control. We will improve performance with it in the future.

Acknowledgments

This project was supported by Grant-in-Aid for Specially Promoted Research of the Ministry of Education, Science and Culture(No.62065002), and was also partly supported by Fujitu Laboratories Limited.

Specification

Total system	
No. of Inference Units	64
No. of Networks	2
Basic clock frequency	10MHz (Now 5MHz)
SPARC clock frequency	20MHz (Now 10MHz)
No. of Used ICs	31821
Power dissipation	about 14000 VA
Inference Unit	
SPARC	Fujitu S-20
Memory	SRAM, 512K Bytes, no wait
FPU	WEITEK 3170 (2.54MFLOPS)
UNIRED	1.2μ CMOS gate array 42696 gates
NIP	1.2μ CMOS gate array 18906 gates
Local memory	SRAM, 4M Bytes
No. of Used ICs	482
Interconnection network	
Switching Unit	1.5μ CMOS gate array 6185 gates
No. of Used ICs	416

Development history

'87	Jul.	Abstract design of whole system started.
	Sep.	Detailed design of network LSI started.
'88	Jul.	Network LSIs finished.
'89	Jan.	Detailed design of network started.
	Apr.	Detailed design of NIP LSI started.
	Oct.	Detailed design of UNIRED LSI started.
	Dec.	Detailed design of IU PCB started.
'90	Apr.	Network construction finished.
'91	Feb.	Prototype version of IU PCB finished.
'92	Apr.	NIP LSIs finished.
	Jul.	UNIRED LSIs finished.
	Sep.	Mass production of IU PCB started.
'93	Feb.	Mass production of IU PCB finished.
	Aug.	Fleng interpreter on PIE64 finished.
	Oct.	PIE64 began to execute Fleng compiled code.
'94	Mar.	IU hardware problems were fixed.
	Sep.	Network hardware problems were fixed.
	Nov.	PIE64 began to work with 64 IUs
After	Nov.	Release to users, Evaluation, Tuning, Maintenance, etc.

References

[1] M. NILSSON and H. TANAKA: "Fleng prolog - the language which turns supercomputers into prolog machines", LNCS264, Springer-Verlag (1989).

[2] K. Ueda: "Guarded Horn Clauses", Technical report, ICOT (1985). ICOT Technical Report TR-103.

[3] E. SHAPIRO Ed.: "Concurrent Prolog", The MIT Press (1987).

[4] E. TAKAHASHI, H. KOIKE and H. TANAKA: "A study of a high bandwidth and low latency interconnection network in PIE64", Proc. of IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing, pp. 5–8 (1991).

[5] K. SHIMADA, H. KOIKE and H. TANAKA: "UNIRED II: The high performance inference processor for the parallel inference machine PIE64", Proc. of Int. Conf. of Fifth Generation Computer Systems, pp. 715–722 (1992).

[6] T. SHIMIZU, H. KOIKE and H. TANAKA: "Details of the network interface processor for PIE64 (In Japanese)", IPSJ SIG Notes, **91**, ARC-87 (1991).

[7] Y. HIDAKA, H. KOIKE and H. TANAKA: "Maintenance environment of Parallel Inference Engine PIE64 (In Japanese)", IPSJ SIG Notes, **91**, ARC-89, pp. 111–118 (1991).

[8] H. NAKADA, T. ARAKI, H. KOIKE and H. TANAKA: "A Fleng Compiler for PIE64", Proceeding of the IFIP WG10.3 Working Conference on Parallel Architectures and Compilation Techniques '94, pp. 257–266 (1994).

[9] Y. HIDAKA, H. KOIKE and H. TANAKA: "Architecture of parallel management kernel for PIE64", Future Generation Computer Systems, **10**, 1, pp. 29–43 (1994).

[10] J. TATEMURA, H. KOIKE and H. TANAKA: "Hyperdebu: a multiwindow debugger for parallel logic programs", PROGRAMMING ENVIRONMENT FOR PARALLEL COMPUTING, Elsevire Science Publishers B.V.(North-Holland) (1992).

[11] J. TATEMURA, H. KOIKE and H. TANAKA: "Control and data flow visualization for parallel logic programs on a multi-window debugger HyperDEBU", PARLE'93 Parallel Architectures and Languages Europe, LNCS 694, Springer-Verlag, pp. 414–425 (1993).

[12] O. SHIRAKI, J. TATEMURA, H. KOIKE and H. TANAKA: "Paf: Performance debugging tool for highly parallel programs(In Japanese)", IPSJ SIG Notes, **92**, PRG-8, pp. 147–154 (1992).

[13] S. MURAKAMI, H. NAKADA, H. KOIKE and H. TANAKA: "Load distribution system of PIE64", Workshop on Parallel Logic Programming attached to International Symposium on Fifth Generation Computer Systems 1994 (1994). To appear.