ICOT Technical Report: TR-514

TR-514

PIM/p:A Hierarchical Parallel Inference Machine

by
A. Hattori, T. Shinogi, K. Kumon (Fujitsu)
& A. Goto

November, 1989

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

PIM/p: A HIERARCHICAL PARALLEL INFERENCE MACHINE

Akira Hattori+, Tsuyoshi Shinogi+, Kouichi Kumon+ and Atuhiro Goto*

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

*Institute for New Generation Computer Technology
4-28, Mita-1, Minato-ku, Tokyo 108, JAPAN

This paper presents and evaluates the main architectural features of a parallel inference machine, PIM/p, which is now being developed for the high-speed execution of AI programs written in the concurrent logic programming language KL1. Because of the frequent interprocess communication required by KL1, PIM/p has a hierarchical structure where eight processing elements are combined with shared memory to form a cluster, and more than sixteen clusters are connected by a hypercube network. To decrease the number of code fetches on the common bus of a cluster, we have developed a macro instruction call mechanism that allows high-level instructions to be executed in the otherwise RISC-like instruction stream. Some software controlled cache memory assist instructions have also been introduced to delete unneeded operand data transfers on the common bus. Simulation has shown their effectiveness. We also evaluated the application of the two main deadlock-free routing methods to the intercluster network, and selected the E-cube method because of its relatively good performance and simple hardware.

1. INTRODUCTION

The parallel inference machine PIM/p [1] is now being developed as part of the Japan Fifth Generation Computer Project. This machine is used for the high-speed execution of large scale artificial intelligence software written in the concurrent logic programming language KL1 [1][2].

PIM consists of several hundred processing elements (PEs). Our performance goal is 10 to 20 MLIPS per 100 PEs. Because KL1 programs are composed of many processes which frequently communicate with each other using AND-stream parallelism, decreasing the cost of interprocessor communications is extremely important in achieving high-speed execution.

We have introduced a hierarchical structure where both low cost local interprocess communication and flexible global communication can easily be taken advantage of by software. Several PEs are combined with shared memory to form a cluster and multiple clusters are connected by a hypercube network.

This paper presents and evaluates the architectural features designed to decrease the cluster bus traffic and to enable efficient data transfer using the intercluster network.

2. TWO LEVEL HIERARCHICAL SYSTEM STRUCTURE

The language KL1 requires frequent and high speed interprocess communication using AND-

stream parallelism. Because quick and exclusive access to shared data such as variables as well as to processes is a key issue in KL1 parallel execution, eight processing elements are combined with shared memory to form a cluster as shown in Figure 1. Each PE has a coherent cache for high-speed memory access and reduced common bus traffic.

Because cluster bus throughput limitations only allow about eight tightly coupled PEs to share a single memory space, we used a hierarchy of shared memory clusters in a hypercube network. Every four PEs are connected to a router node of the hypercube in order to decrease the dimension and diameter of the cube as shown in Figure 1. The intercluster network is used to transfer message packets among clusters both for distributed unification and for process migration.

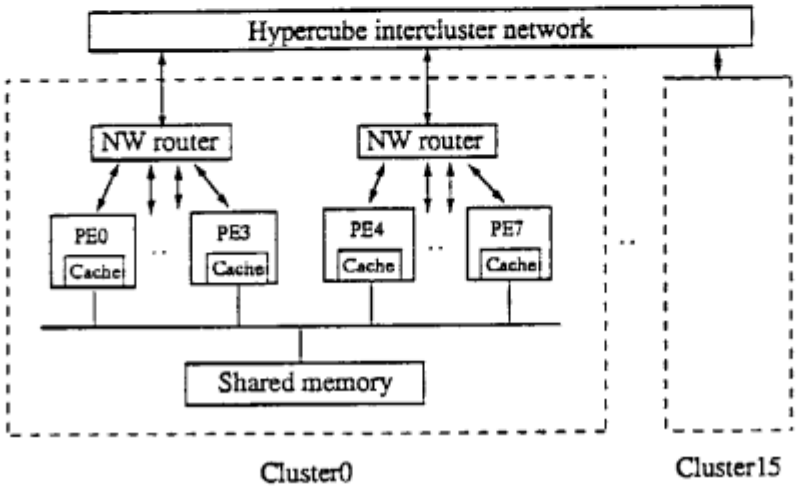

FIGURE 1 System Configuration

3. HIGH SPEED CLUSTER ARCHITECTURE

3.1. PE Architecture

The PEs ares designed for efficient KL1 execution. They use a tag architecture because KL1 does extensive runtime data type checking. The PEs are also designed with a RISC-like instruction set [3] in order to realize a short pipeline clock cycle and to facilitate good code optimization by a compiler. The resulting four stage pipeline has a 50 nS clock cycle and can be built on an 80K gate CMOS LSI. The pipeline structure is shown in Table 1. Code compiled for such an instruction set can become very large however, resulting in excessive intracluster common bus traffic. Yet, decreasing common bus traffic is absolutely necessary in order to maintain cluster performance that is highly proportional to the number of PEs. This means that the architecture must be set up to help decrease this traffic.

3.2. Macro Instruction Call Mechanism

To decrease the amount of program code on the common bus, we have developed a macro instruction call mechanism that allows high level instructions to be executed in the RISC-like instruction stream. PEs have two types of instructions, internal and external. Both can be used to execute most operations. External instructions represent compiled KL1 program code. They are fetched from the cache or main memory through an instruction buffer and executed each cycle by a four stage pipeline, as shown in Figure 2.

Internal instructions make up the subroutines of complex KL1 operations and are stored in the internal instruction memory of each PE. These instructions are invoked by external macro call instructions and executed by the same pipeline as external instructions.

3.3. Cache Assist Instructions

The other big contributor to bus traffic is operand data, so we added cache assist instructions to help delete unneeded operand data transfers.

KL1 programs allocate new memory area and write data more frequently than other languages. An ordinary write-back cache usually fetches-on-write if a cache miss occurs. When new data is created in a new memory area, a block fetch from shared memory is not needed, so a direct write instruction is introduced that allocates a new area to write to in a cache memory and avoid unneeded block fetches.

TABLE 1 Pipeline Structure

Stage	ALU Instruction	Memory Access Instruction
D	Decode	Decode & reg. read
A	----	Address calcuration
T	Register Read	Cache access(Tag)
B	ALU & reg. write	Cache access(Data) & reg.write

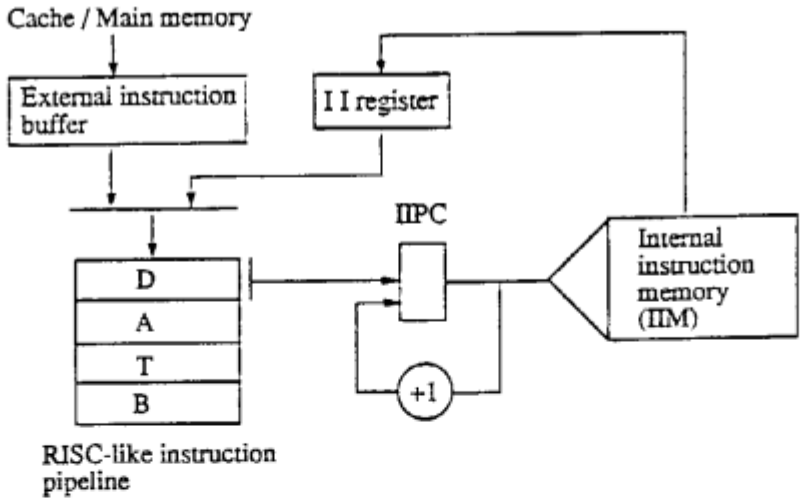

FIGURE 2 Macro Instruction Call

3.4. Simulating Cluster Performance

To evaluate the macro instruction call mechanism and cache assist instructions, we measured cluster bus traffic using a software simulator. The measurements were made for 8 PEs, each with 64K bytes of cache memory, and using four variations of each test program.

The simulator first executed KL1-b code [5] with direct write and macro call instructions. KL1-b is a WAM-like [6] abstract instruction set. The size of the KL1-b code was then doubled and quadrupled in order to simulate the absence of the macro call instruction. Tests with small

hand compiled KL1 programs showed that the absence of the macro call instruction increases the code by about 3.5 times. The KL1-b code was then executed without direct write instructions. Table 2 shows the simulation results measured in cluster bus cycles, using two benchmark programs. One is a bottom-up parser (BUP), and the other is the 8 Queens problem.

As the size of the code doubles, the number of bus cycles increases an average of 14%. As the code quadruples, the number increases 72%. The absence of the direct write instruction increases the number of bus cycles by 32%.

3.5. Cluster Performance

The cluster performance is calculated based on the preceding simulation, as shown in Table 3. Cluster performance decreases by 5% when the code size is doubled, and about 30% when it is quadrupled. The absence of the direct write instruction decreases cluster performance by 6%.

TABLE 2 Common Bus Cycles

unit: 1000 cycles

	Standard	Doubled code size	Quadrupled code size	No direct write
BUP	512	643	1217	654
Ratio to standard	1	1.26	2.38	1.28
8 Queens	337	344	356	461
Ratio to standard	1	1.02	1.06	1.37
Average	1	1.14	1.72	1.32

TABLE 3 Cluster Performance (BUP)

Tcb and Tw units: 1000 cycles

	Standard	Doubled code size	Quadrupled code size	No direct write
Com. bus cycle per PE (Tcb)	64	80	152	82
Wait time (Tw)	5.5	10.2	57.8	10.5
Performance ratio	1	0.95	0.67	0.94

4. INTERCLUSTER NETWORK

4.1. Design Issues

We connected multiple PEs directly to each router node on the hypercube network both in order to prevent increased cluster bus traffic due to intercluster communications, and to decrease the size of the network.

Deadlock-free routing of message packets is important because KL1 programs need dynamic and flexible interprocess communications. Store-and-forward deadlock [7] is an especially significant issue in a packet switching network.

4.2. Deadlock Free Routing Methods

We studied two main deadlock-free routing methods, one using a structured buffer pool algorithm [8] and the other using the E-cube algorithm. The structured buffer pool (SBP) method does not depend on the routing path, but restricts the assignment of node buffers to message packets. This method can avoid crowded paths and increase throughput. Packet sequencing is not maintained however, and the buffer control circuitry is complex. The E-cube method restricts the routing path, and throughput may decrease because of crowded paths, but packet sequencing is retained and buffer control circuitry is simple.

4.3. Intercluster Network Simulation

We simulated both methods under conditions expected on the actual PIM/p machine. The network is a four dimensional hypercube with four 256-byte buffers at each router node. We simulated configurations using both one and four PEs connected to each router node. The packet size was distributed uniformly between 0 and 2K bytes.

Figure 3 shows the network performance of each routing method. We tested three routing methods: the E-cube and two types of structured buffer pools. The difference between SBP1 and SBP2 is the flexibility of buffer bank assignment to message packets.

The E-cube has almost the same throughput as both the structured buffer pool methods. Since it has simpler buffer control circuitry and maintains correct packet sequencing, we adopted this method. As shown in Figure 3, the network with four PE ports per node performed better than that with one PE port per node because message packet destinations are better randomized.

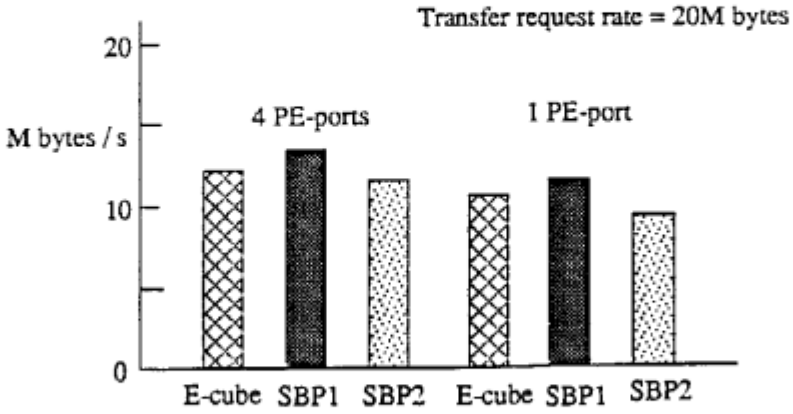

FIGURE 3 Network Performance

5. SUMMARY

We have presented and evaluated the main architectural features of the parallel inference machine PIM/p now being developed. These features are:

(1) A macro instruction call mechanism and cache memory assist instructions to decrease cluster bus traffic. Simulation showed their effectiveness.
(2) Multiple PEs connected to each hypercube router node to increase network efficiency.
(3) Use of the E-cube deadlock free routing method for the intercluster network because of its performance and simple buffer control circuitry.

ACKNOWLEDGEMENTS

We would like to thank Mr. J. Tanahashi, Mr. H. Hayashi, and Dr. S. Uchida for their valuable suggestions and guidance. We would also like to thank all the PIM researchers both in ICOT and in Fujitsu Limited. Special thanks goes to Mr. A. Matsumoto of Mitsubishi Electric Co. for the cluster simulation and measurements.

REFERENCES

[1] Goto, A., Overview of Parallel Inference Machine Architecture (PIM), Int. Conf. FGCS (1988), pp. 208.
[2] Ueda, K., Guarded Horn Clause: A Parallel Logic Programming Language with the Concept of a Guard, TR208, ICOT (1986).
[3] Patterson, D. and Sequin C., A VLSI RISC, IEEE Computer, Vol.15, No.9 (1982), pp. 8.
[4] Shinogi, T., et al., Macro-call Instruction for the Efficient KL1 Implementation on PIM, Int. Conf. on FGCS (1988), pp. 953.
[5] Kimura, Y. and Chikayama, T., An Abstract KL1 Machine and its Instruction Set, Proc. SLP (1987), pp.468-477.
[6] Warren, D.H.D., An Abstract Prolog Instruction Set, Technical Note 309 (1983), Artificial Intelligence Center, SRI.
[7] Merlin, P. and Schweitzer, P., Deadlock Avoidance in Store-and-Forward Deadlock, IEEE Trans., Comm., Vol.28, No.3 (1980), pp.345-354.
[8] Raubold, E. and Haenle, J., A Method of Deadlock-Free Resource Allocation and Flow Control in Packet Networks, Proc. ICCC 1976 (1976), pp. 483-487.
[9] Dally, W. and Seitz, C., Deadlock-Free Message Routing in Multiprocessor Interconnection Networks, IEEE Trans., Comp., Vol.C-36, No.5 (1987).