TR-0871

Evaluation of the Cluster Structure on the PIM/C Parallel Inference Machine

by

T. Tarui, M. Asaie, N. Ido, T. Nakagawa
& M. Sugie

April, 1994

EVALUATION OF THE CLUSTER STRUCTURE ON THE PIM/C PARALLEL INFERENCE MACHINE

Toshiaki Tarui*, Machiko Asaie*, Noriyasu Ido*, Takayuki Nakagawa**, and Mamoru Sugie*
* Central Research Laboratory, Hitachi, Ltd.
** General Purpose Computer Division, Hitachi, Ltd.
1-280 Higashi Koigakubo, Kokubunji-shi, Tokyo 185, Japan
Email: tarui@crl.hitachi.co.jp

Abstract -- *The characteristics of a cluster-structure parallel computer are analyzed and evaluated on the PIM/c parallel inference machine, which consists of eight-processor shared-memory clusters communicating through a processor connected to a network. To avoid communication bottlenecks, the maximum number of processors in a cluster is limited by the ratio of communication operations to program-execution operations. Since this ratio can be as high as 30% on the PIM/c, the network receiving operations should be distributed to processors in the same cluster.*

1. INTRODUCTION

One of the most important design issues in parallel computers is the need for a structure efficiently supporting communication between processors, and there are basically two ways to connect parallel processors: one with a network architecture which has high scalability and the other with a shared-memory architecture which enables efficient communication.

A cluster structure parallel computer, in which clusters of shared-memory multi-processors are connected through a network, has been proposed to take advantage of the efficient communication provided by shared memory [1]. To construct a large-scale parallel system, a number of these clusters, each consisting of several processors and a shared memory connected by a common bus, are connected by a network. Each cluster includes a network interface, such as a communication processor, connected to the network, and the hardware costs in this cluster architecture can be reduced by limiting each cluster's connection to the network to one interface. Because shared-memory multi-processor technologies make it possible to develop a parallel computer with a cluster structure efficiently, the cluster structure is one of the most important technologies needed to develop a large-scale parallel system. Large-scale shared memory systems [2] also use the cluster structure.

The parallel inference machine (PIM) [3] was developed in Japan's Fifth Generation Computer Project implemented by the Institute for New Generation Computer Technology (ICOT). The PIM was designed to efficiently execute programs written in the parallel logic-programming language KL1 [4], which has AND-parallel feature and has dataflow process synchronization mechanism. Hitachi's PIM model c (PIM/c) [5] consists of 256 processing elements and is organized into 32 clusters of eight-processor, shared-memory multi-processors. Each cluster has one extra processor dedicated to network communication.

The two levels of communication in the cluster structure, network and common bus prevent conventional programs optimized for a one-level architecture from being executed efficiently because the two levels have communication overheads and latencies that are very different. The structure of a conventional program must therefore be changed for it to work efficiently on a cluster structure, but the characteristics of a program to be run on a cluster structure have not yet been clarified in detail. Having completed the development of the PIM/c cluster system, we can now evaluate its performance in detail, including the actual overhead of the shared memory and network communication.

This paper analyzes programming models on a cluster-structure parallel computer, focusing on network communication between clusters. The performance of application programs running on the PIM/c is measured, and the execution models are verified by the results. The characteristics and design issues of the cluster structure are also discussed. Because this paper focuses on the characteristics of the cluster structure, it investigates the total system balance in detail and does not discuss the individual performance of network/common-bus communication.

2. CLUSTER STRUCTURE

2.1. PIM/c Architecture

The processing elements (PEs) of the PIM/c (Fig. 2.1) are connected hierarchically: nine-processor shared-memory multi-processors compose a cluster, and 32 clusters are connected through a crossbar network. Inside each cluster, eight PEs, one cluster controller (CC) connected to the network, and the main memory are connected to a two-way-interleaved common-bus. The 256 PEs are contained in four cabinets. Each PE executes the KL1 programs, and the CC handles network communication. A snooping cache mechanism is used in the PEs and the CC to support efficient shared-memory communications.

KL1 execution results in a large firmware overhead for intercluster communication such as address translation, real-time garbage collection, and packet construction/deconstruction. To reduce the communication overhead on the PEs in the PIM/c, normal system firmware is designed so that all operations involved in communication are executed on the CC. Execution on the CC, however, may become a bottleneck even when the performance of the network hardware itself is sufficient. To prevent this, the PIM/c has an additional execution

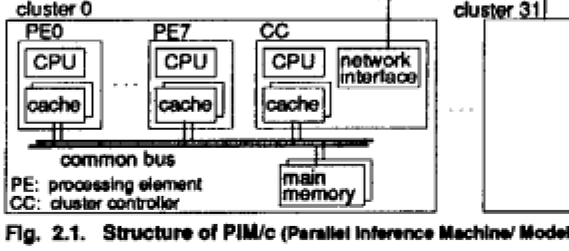

Fig. 2.1. Structure of PIM/c (Parallel Inference Machine/ Model C)

mode in which the network receiving operations are distributed to the PEs. As will be shown in Section 2.4, the performance saturation point is thus increased but the performance of each PE decreases because the PEs must execute additional communication operations.

2.2. Characteristics

The PEs are connected at two levels of architecture: between the shared memory in each cluster and between the clusters in the network. PEs in the same cluster can communicate by using the high-speed shared-memory system, and PEs in different clusters communicate through the network interface. This is why the communication overheads and latencies differ between the two levels.

Compared with a one-level, network-connected, parallel computer, a cluster structure has two significant advantages:

(1) If each cluster includes n PEs, network hardware can be reduced to about 1/n. And system hardware costs can be further reduced by using low-cost shared-memory multiprocessors.

(2) Network communication can be reduced by utilizing the communication locality; that is, by ensuring that the processes which communicate frequently with each other are mapped to the same cluster. System performance is improved because intracluster communications are much faster than intercluster communications.

The cluster structure also has some disadvantages:

(3) Network throughput per PE is reduced to 1/n because the n processors share one network path. Thus, the network interface may become a bottleneck if a program has frequent communications.

(4) Intercluster communication latency is high because the PEs do not have a direct network path and have to communicate through the network interface. Furthermore, message latency increases because communication requests are serialized when several PEs in the same cluster attempt to communicate at the same time.

2.3. Communication within a Parallel Program

For a parallel program running in a cluster structure, network communication can be classified into three types.

(1) execution dominant

When a program requires little communication, communication cannot be a significant limit to system performance. This type of program is not evaluated in this paper.

(2) latency dominant

When a program must wait during communication, it becomes idle and system performance deteriorates. In this case, system performance strongly depends on the speed of intercluster communication.

(3) throughput dominant

When a program can hide network latency by communicating and executing programs in parallel, or when the program has a sufficient number of processes and a lightweight context-switch mechanism for communication (like the suspension/resumption mechanism in KL1) is provided, system performance is independent of network latency. If network throughput becomes overloaded, however, system performance deteriorates because of communication-waiting time. For executing throughput-dominant programs, network throughput must therefore be high enough to support the communication produced by all of the PEs in all of the clusters.

2.4. Execution Models

Our models use the following assumptions:

- Throughput of the network and the common bus is sufficient.
- Shared-memory multi-processor overhead is low enough.
- Hardware latency in the network is so much smaller than the overhead in the CC (network interface) that it can be ignored.

Thus, the only communication overhead occurs in the CC. Because of the large firmware overhead required in KL1 communication, these assumptions are appropriate for the PIM/c. Each PE executes processes that have similar characteristics.

We use the following notations in our analysis:

n:	number of PEs in each cluster
i:	system cycles used by the program to execute one process (does not include communication cycles)
c:	system cycles used by the communication processing during one process $(c < i)$
e:	system cycles used by the communication receiving processing in c $(e < c)$
C:	communication overhead (c/i)
E:	receiving overhead (e/i)

Figure 2.2 shows the execution model for the throughput-dominant program. In the CC, nc cycles are used for communication during the execution of one process. Since communication and program execution can be performed in parallel, the execution time for one process is

$$T1(n) = \max(i, nc), \tag{1}$$

where each cluster executes n processes. Therefore, the speedup compared with the execution time with one PE is

$$S1(n) = n \quad (\text{when } n < 1/C), \text{ or}$$
$$= 1/C \quad (\text{when } n > 1/C). \tag{2}$$

Linear speedup can be obtained when execution time in the CC is less than the program execution time in the PE $(n < 1/C)$. When the number of PEs is greater than 1/C, the CC becomes a bottleneck and system performance saturates.

To prevent this bottleneck the load on the CC must be reduced, and one way to do this is to distribute portions of the communication operations to PEs in the same cluster. When this is done, the execution time for one process is

$$T2(n) = \max(i+e, n(c-e)), \tag{3}$$

where e of c cycles have distributed. The speedup compared with the execution time with one PE (the receiving operation is not distributed) is

$$S2(n) = n / (1+E) \quad (\text{when } n < (1+E) / (C-E)), \text{ or}$$
$$= 1 / (C-E) \quad (\text{when } n > (1+E) / (C-E)). \tag{4}$$

Distributing the receiving operation to PEs can increase the maximum performance and the speedup-saturation point, but when the number of PEs is low, performance is limited because the workload on each PE increases because of the receiving overhead (Fig. 2.4).

Figure 2.3 shows the execution model for the latency-dominant program. Because communication and program execution cannot be overlapped, the execution time for one process is

$$T3(n) = i + nc, \tag{5}$$

where each cluster executes n processes. Therefore, the speedup compared with the time for single-PE execution of a throughput-dominant program is

$$S3(n) = n / (1 + nC). \tag{6}$$

As shown in Fig. 2.4, the latency-dominant program is always slower than the throughput-dominant program, since PEs become idle during communication.

So far, we have assumed that the ratio of communication cycles to program-execution cycles in each cluster is indepen-

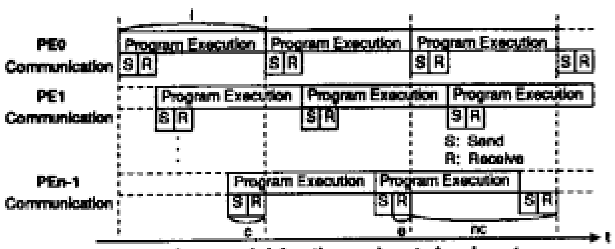

Fig. 2.2. Execution model for throughput-dominant program.

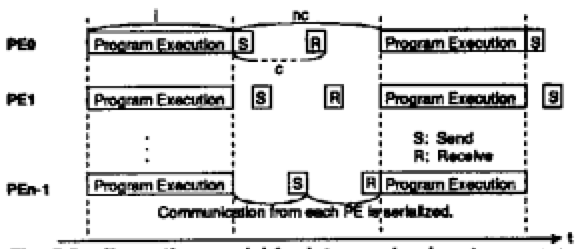

Fig. 2.3. Execution model for latency-dominant program.

dent of the number of PEs in the cluster. This implies that the total number of program-execution cycles in each cluster increases as the number of PEs increases. In some latency-dominant programs, on the other hand, the total number of program-execution cycles in each cluster does not increase as the number of PEs increases. For such programs the execution time for one process is the same as that given by Eq. (5) but the total number of program execution cycles in each cluster is constant. Therefore, the speedup for these programs is

$$S4(n) = T3(1) / T3(n) = (1 + C) / (1 + nC). \qquad (7)$$

Because the total number of program execution cycles in each cluster is independent of n but the number of communication cycles (nc) increase with n, S4(n) decreases as the number of PEs increases. Thus, for each cluster the ratio of communication cycles to program-execution cycles increases as n increases.

3. EVALUATION

3.1. Evaluation Method

We measured the performance of a cluster structure by using the PIM/c with large-scale application programs written in KL1. Although KL1 is designed to solve knowledge-information programs, it is a general purpose language that includes the synchronization and load-balancing mechanisms essential to parallel processing.

We used two ICOT-developed KL1 programs [6] as benchmark programs:

- LSI routing program

  This throughput-dominant program uses the lookahead line-search method. For each grid line on the LSI, many routing processes are created and are used to execute routing in parallel. The system therefore has a large number of routing processes, and the suspension/resumption mechanism of KL1 makes it possible to hide communication latency.

- Logic simulation program

  This latency-dominant program is a conventional time-wheel version of the parallel logic simulation program. Several (1-8) simulation engines are placed on each cluster and execute the simulation in parallel. Each simulation engine must synchronize once per simulation cycle by sending a message to the time manager. This is done simultaneously by each PE in the cluster, so the simulation engines become idle.

We measured the relationship between the number of working PEs in a cluster and the speedup. Execution time was measured by a software-controlled timer, and the number of PEs in each cluster was adjusted by changing the active PEs in the routing program or changing the number of simulation engines in the logic simulation program. The number of clusters was eight (maximum 64 PEs).

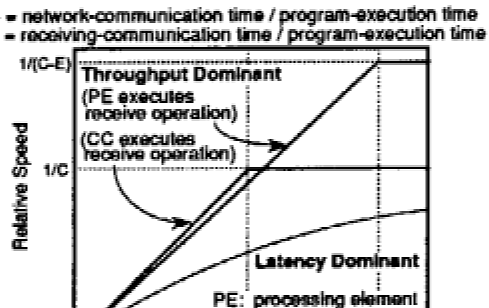

Fig. 2.4. Performance of throughput-dominant and latency-dominant programs.

3.2. Results

3.2.1. Effect of the Number of PEs in a Cluster

Table 3.1 summarizes the PIM/c execution time for each benchmark program, and Fig. 3.1 shows the relationship between the number of PEs in each cluster and the speedup of the routing program. Because these results are consistent with the analytical results shown in Fig. 2.4, the execution model of the throughput-dominant program discussed in Section 2.4 applies to the PIM/c.

With normal execution strategy, in which all receive operations are executed on the CC, the speedup saturates at about three PEs. With more PEs, the utilization of the CCs (observed using a real-time performance meter) is 100% while the utilization of the PEs is only 60-80%; that is, the PEs become idle because of the CC bottleneck. On the other hand, when the receive operations are distributed to the PEs, an almost linear speedup can be obtained. For four or more PEs, the execution time is faster than that with the normal strategy, but when there are only one or two PEs the distributed strategy is 10-20% slower than the normal strategy. With the distributed strategy the utilization of the PEs is almost 100% and the utilization of the CCs is about 50%. This confirms that communication bottlenecks on the CC occurring when executing a throughput-dominant program can be prevented by distributing the receive operations to the PEs.

The communication overhead and other parameters discussed in Section 2.4 can be estimated from Fig. 3.1. The communication overhead C is 0.3 and the receiving overhead E is 0.15; that is, network communication takes as much as 30% of the program execution time and about half of this is for receiving. Thus, the number of PEs for which performance saturates is three when all communications are executed on the CC and eight when the receiving operations are distributed to the

Table 3.1. Execution time of benchmark programs.
8 clusters (Max. 64 PEs) (seconds)

Benchmark Program		Routing Program		Simulation Program
		CC executes receive operations	PE executes receive operations	
Number of PEs in a Cluster	1 PE	129.9	152.8	94.0
	2 PE	71.0	86.2	104.0
	4 PE	49.6	39.3	131.0
	8 PE	53.4	22.5	190.0

Table 3.2. Number of multi-processor events per PE.

Number of PEs in a Cluster	Number of Cache Accesses	Number of Bus Accesses	Bus-Busy Cycle Time	Lock-Waiting Cycle Time
	(thousand times)		(thousand cycles)	
4	47,559	549	4,184	406
8	31,594	554	3,458	3,114

Routing program, CC executes receive operations.

PEs. (These values of course vary with the application program.)

Figure 3.2 shows that the speedup of the simulation program decreases as the number of PEs in each cluster increases. These experimental results are equal to the speedup S4(n) derived analytically in Section 2.4. In this program the total number of execution cycles in each cluster is independent of the number of PEs because the number of the simulation gates is constant. The communication from each cluster to the time manager, on the other hand, increases as the number of PEs (simulation engines) increases. System performance thus decreases because the ratio of communication cycles to program-execution cycles increases as the number of the PEs increases.

3.2.2. Multi-processor Overhead in the Cluster

The speedup of the routing program saturates at around three PEs when all network operations are performed on the CC. Although the execution model in Section 2.4 predicts that constant performance should then be obtained, the experimental results show that speedup decreases slightly when there are more than six PEs. This is because the shared-memory multi-processor overhead inside the cluster degrades performance.

Table 3.2 shows, for four PEs and for eight PEs, the number of common-bus transaction per PE (measured by the hardware bus monitor). For four PEs there are more cache accesses and bus-busy cycles because each PE does twice the work. On the other hand, the lock-waiting time is much greater with eight PEs, despite the small workload on each PE. Common-bus utilization is only 5.8% and common-bus throughput is sufficient. This indicates that when the number of PEs in the cluster is close to eight the shared-memory multi-processor overhead increases rapidly because of lock concentration, which decreases system performance even though the common-bus throughput is sufficient.

When the application program has sufficient parallelism, this multi-processor overhead does not degrade performance because the speedup is then more significant. When cluster performance saturates, however, increasing the number of working PEs beyond the saturation point degrades system performance because of the shared memory overhead. The number of PEs that can perform efficiently in each cluster is therefore limited.

4. DISCUSSION

4.1. Communication Overhead

When a communication processor is placed in each cluster, the maximum number of PEs in one cluster is limited by the network overhead: the ratio of communication operations to program-execution operations must be controlled to avoid communication bottlenecks. As a result, total system performance strongly depends on the communication overhead. If it is large, the number of executing processors in the cluster should be limited to minimize the shared memory overhead. Since network hardware performance in a normal parallel computer is usually sufficient, the communication overhead is primarily caused by the software involved in network communications and depends on the communication characteristics of the application program.

But because the communication overhead for a KL1 program on the PIM/c is as much as 30%, system performance is limited by CC bottlenecks. Some effort to reduce the communication overhead should therefore be made. One approach is to distribute some of the communication operations to PEs. Another approach is to reduce intercluster communication, by using the communication locality of the program. A roughly 20% speedup of the LSI routing program can be obtained by optimizing the process distribution [7].

4.2. Network Communication Execution Strategy

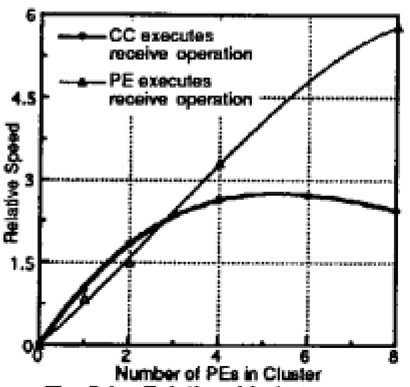

Fig. 3.1. Relationship between speedup of routing program and the number of PEs.

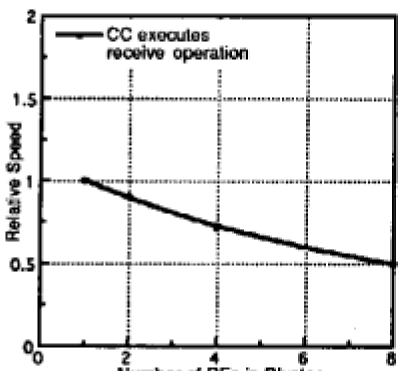

Fig. 3.2. Relationship between speedup of logic simulation program and the number of PEs.

Balancing the system load between the communication processor and the computation processors is crucial, and the following strategy is appropriate for cluster structures.

- When the number of computation processors in a cluster is small or the communication overhead is not large, all communication operations should be performed on the communication processors in order to reduce the overhead on the computation processor.
- When the number of computation processors in a cluster is large or the communication overhead is large, some of the communication operations should be distributed to the computation processors.

- Since communication overhead varies with programs, the system should be designed so that communication processors seldom become the bottleneck. When they do, utilization of the computation processors decreases dramatically.

When a communication bottleneck occurs, reducing the communication overhead in the network processor is of primary importance. Communication-related processing must therefore be performed in the computation processor, even though this ties up the computation processor during this processing.

On the PIM/c the communication overhead is so heavy that, when all of the communication operations are performed on the CC, system performance saturates at only three PEs. An improved execution strategy, in which the receiving part of the network operation is distributed to the PEs in the cluster, should therefore be used. With this strategy, CC utilization decreases to about 50% and network bottlenecks are avoided. The maximum number of PEs in the cluster can be increased to eight, but below the saturation point (one or two PEs), the later strategy is 10-20% slower than the former strategy because the PEs must execute additional communication operations. Since the final goal of a parallel computer is to provide high performance with a large number of processors, the later strategy should be used despite its poor single-processor performance.

4.3. Programming Model for the Cluster Structure

In a cluster structure, throughput-dominant programming is needed to hide communication latency and provide high performance. If the program has latency dominant characteristics, however, processor utilization decreases as the number of PEs in each cluster increases. This is because communication requests from PEs in a cluster are serialized on the CC and communication latency increases with the number of PEs. Furthermore, if the program has no communication locality, so that communication overhead of a cluster increases with the number of PEs, system performance decreases as the number of PEs increases. The logic simulation program evaluated in this paper is one such program, for which even if the cluster has several PEs only one PE in the cluster should be executed.

To avoid the performance degradation that occurs with a latency-dominant program, the algorithm should be changed to a throughput-dominant algorithm. For example, a time-warp algorithm suitable for parallel execution has been proposed [6] for the logic simulation program. Synchronization through the time manager is not used in this algorithm, so that parallel execution of simulation and communication is possible.

5. CONCLUSION

This paper clarified the characteristics of the cluster structure by analyzing and measuring the performance of a cluster-structure parallel computer. A cluster-structure architecture reduces the amount of hardware needed to construct a large-scale parallel computing system, but throughput bottleneck and increasing latency in the network interface must be overcome in designing a high-performance parallel system.

The main results obtained from our investigation are as follows.

(1) In a cluster-structure parallel computer, the maximum number of processors in each cluster is limited by the communication overhead (the ratio of operations required for communication to those required for program execution). System performance saturates when the number of processors exceeds the limitation.

Since the communication overhead for KL1 inference operations on the PIM/c is as much as 30%, the number of PEs in a cluster is limited to three if the communication processor (CC) performs all of the communication operations.

(2) To prevent bottlenecks in the communication processors, some communication operations should be distributed to program-executing processors in the same cluster. This enables the maximum number of processors in a cluster to be increased.

On the PIM/c, the workload on the CC can be sufficiently reduced by distributing the network receiving operations to the PEs so that the maximum number of PEs in a cluster can be increased to eight.

(3) The programming algorithm on a parallel computer should be designed to hide communication latency, because processor utilization decreases when a processor has to wait because of this latency. Latency hiding is indispensable in a cluster structure because the communication requests from processors in the cluster are serialized in the network interface.

ACKNOWLEDGMENTS

We thank Dr. Shunichi Uchida, Director of the ICOT Research Center, for his guidance and support, and we thank Dr. Takashi Chikayama and other ICOT members for their helpful discussions. This research was sponsored by the ICOT.

REFERENCES

[1] N. Hamanaka, J. Nakagoshi, and T. Tanaka, "Reducing Network Hardware Quantity by Employing Multi-Processor Cluster Structure in Distributed Memory Parallel Processors," *Parallel Processing: CONPAR 92 - VAPP V*, Springer-Verlag 634, 1992, pp. 25-30.

[2] D. Lenoski, J. Laudon, K. Gharachorloo, W. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. Lam, "The Stanford Dash Multiprocessor," *IEEE Computer*, Vol. 25, No. 3, 1992, pp. 63-79.

[3] S. Uchida, "Summary of the Parallel Inference Machine and its Basic Software," *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pp. 33-49.

[4] T. Chikayama, "Operating System PIMOS and Kernel Language KL1," *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pp. 73-88.

[5] T. Nakagawa, N. Ido, T. Tarui, M. Asaie, and M. Sugie, "Hardware Implementation of Dynamic Load Balancing in the Parallel Inference Machine PIM/c," *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pp. 723-730.

[6] H. Date, Y. Matsumoto, K. Kimura, K. Taki, H. Kato, and M. Hoshi, "LSI-CAD Programs on Parallel Inference Machine," *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pp. 237-247.

[7] M. Asaie, T. Nakagawa, T. Tarui, N. Ido, and M. Sugie, "Evaluation of Load Balancing Strategy using LSI Routing Program on Parallel Inference Machine PIM/c," *Proceedings of Joint Symposium on Parallel Processing 1994* (in Japanese).