TR-247

Parallel Cache and Hardware Lock Mechanism for PIM Cluster

by

A. GOTO, A. MATSUMOTO, T. NAKAGAWA

M. SATO and H. SHIMIZU

March, 1987

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

(Summary for ICPP'87 Short Papers)

Parallel Cache and Hardware Lock Mechanism for PIM Cluster

Atsuhiro GOTO Akira MATSUMOTO Takayuki NAKAGAWA

Masatoshi SATO Hajime SHIMIZU

Fourth Research Laboratory,
Institute for New Generation Computer Technology (ICOT)

Mita-Kokusai Building 21F., 4-28, Mita 1, Minato-ku, Tokyo 108 JAPAN,
Tel: 03(456)3193 Telex: ICOT J32964

CSNET: goto%icot.jp@relay.cs.net
ARPA: goto%icot.uucp@eddie.mit.edu
UUCP: ihnp4!kddlab!icot!goto

January 13, 1987

Abstract

The parallel inference machine (PIM) is now being developed at ICOT. PIM consists of a dozen or more clusters, each of which is a tightly-coupled multiprocessor with shared memory and a common bus. KL1, a parallel logic programming language based on GHC, is executed using a shared heap model on each PIM cluster.

A parallel cache and hardware lock mechanism is being investigated to enable quick and exclusive accesses to the shared memory. The most important issue is how to reduce common bus traffic. The data access characteristics of KL1 execution are examined first. Then the write-back cache protocol having six cache states designed for KL1 execution on each PIM cluster is described. Next the hardware lock mechanism is attached to the cache on each processor. This lock mechanism enables word by word locking efficiently, reducing common bus traffic by using the cache status. Finally, the cache and lock mechanism is evaluated using a software simulator, and its availability is confirmed.

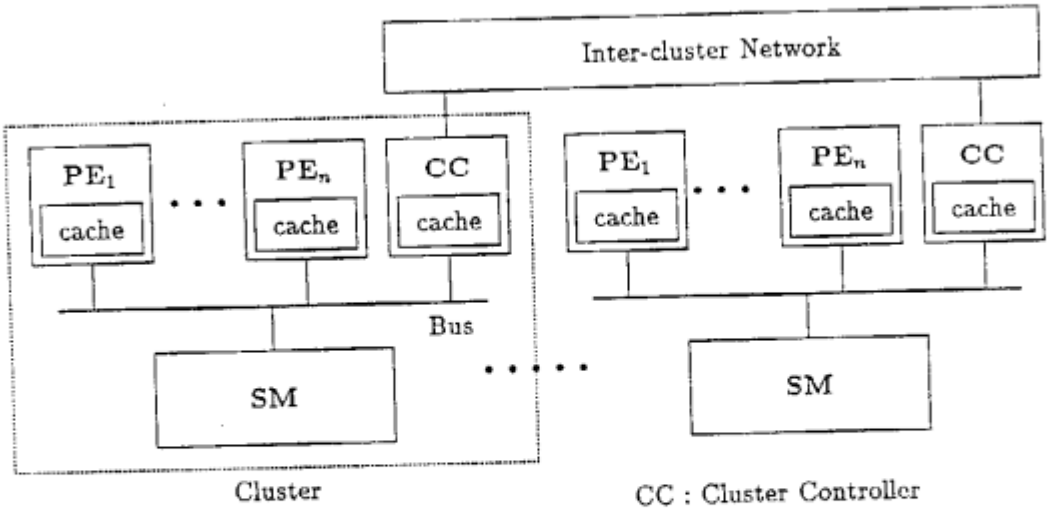

Figure 1: PIM Overview

1 Introduction

The parallel inference machine PIM[6] is now being developed at ICOT. PIM has a hierarchical structure with a cluster concept (Figure 1). Each cluster consists of eight or more processors (PE) which communicate through shared memory (SM) over a common bus. The PIM target language is the parallel logic programming language KL1, based on GHC[9]. We now have two KL1 parallel execution models for PIM: the message-oriented model[7,8] for intercluster parallel execution and the shared heap model[3] for the tightly-coupled multiple processors in each PIM cluster.

Focusing on KL1 parallel execution in each cluster, quick and exclusive accesses to shared data are the key issues. Parallel cache mechanisms are key elements in providing quick data access. Several cache protocols have been proposed so far[1,4,5], each of which aims to solve the so-called *cache coherence* problem. The next step is to clarify what kind of coherence protocol is suitable for KL1 execution.

This paper first briefly analyzes KL1 parallel execution by the shared heap model. Then the parallel cache and lock mechanism for KL1 is given, followed by the evaluation using software simulation.

2 KL1 Parallel Execution by Shared Heap Model

2.1 Data structures in KL1 execution

The following data/control structures are used in KL1 goal reduction. Parallel goals are represented by *goal records* with argument lists and their *environments*. *Environments* consist of goal argument variable cells. The reducible goal records are arranged in a *ready queue*. Some goals are waiting for the instantiated values of variable cells in order to synchronize with other parallel goals. Such goal records are *bind-hooked* with the variable cells by *suspension records*. The *metacall records* form a *goal tree*, whose leaves are the goal records, to manage their logical results (success/failure).

2.2 Shared heap model

In the PIM cluster, a KL1 program is executed using the following shared heap model. Each processor has its own ready queue. From the logical viewpoint, goal records are not shared even if they are stored in the physical shared memory. On the other hand, goal environments, metacall records, and suspension records are shared among processors. Clauses in KL1 programs are compiled into WAM like machine codes[10], called KL1-B[2]. Each processor dequeues a goal record from its ready queue,

Table 1: Data Access Characteristics in a KL1 Sample Program

	Read	Write	Shared
Code	54%	0%	No
Goal Records	11%	11%	No
Environments	8%	8%	Yes
Suspension Records	0.5%	0.5%	No
Metacall Records	5%	2%	Yes
Total	78.5%	21.5%	

then performs goal reduction by executing the corresponding codes, accessing to the goal environment in the shared memory.

3 Parallel Cache and Lock Mechanism

3.1 Observation about KL1 data access characteristics

Before discussing the hardware mechanism, we have examined the data access characteristics in KL1 execution. From our preliminary simulation, data access characteristics can be roughly summarized as follows. (Table 1)

First the code fetches are read-only operations, so they do not incur the cache coherence problem. Next, goal records are not logically shared. Therefore, even the write operations do not require access exclusiveness.

Environments are shared logically and physically. In addition, KL1 needs more write accesses to environments than conventional languages. Thus, it is very important to maintain the cache coherence and access exclusiveness correctly and efficiently.

As a whole, the shared heap model almost keeps to the single assignment rule. This causes the monotonous consumption of memory areas. When the processor intends to use new environment area, it can simply write the data into its own cache block without fetching a block. In addition, the processor often communicates with other through one-write-one-read buffers. The cache block for such a buffer can be purged immediately after the receiver processor reads in,

Because of the single assignment behavior of KL1, exclusive access to goal environments are necessary only when the processor intends to instantiate undefined variables. However, rather frequent word by word locks may be necessary, even if the operations after locking data are simple and the locks may not conflict so often. We have to provide a simple and efficient hardware lock mechanism.

3.2 Parallel cache with lock directory

Cache performance is usually discussed by the hit-ratio. However, in the shared memory architecture, the common bus traffic is a more important factor. We have designed a parallel cache and lock mechanism (Figure 2), which can reduce the common bus traffic. The basic idea is almost comparable to Bitar & Despain's cache[1], however, it has suitable features for KL1.

The cache mechanism is basically *write-back* cache. The cache directory (CD) maintains six internal states (Table 2) with address tags for each cache block. The six states are made from the four attributes: *valid/invalid, exclusive/shared, origin/copy, clean/modified.*

The lock directory (LD) maintains the locked word addresses with three states, shown in Table 3. Lock operations depend on both the entry state in LD and the corresponding cache block states in CD, so both CD and LD are updated simultaneously by processor commands and bus commands. With this lock directory each processor can lock a lock by word up to the number of the lock directory entries. In addition, the lock directory always snoops the common bus, so the cache block for a locked word can be swapped out. However, when the processor tries to lock more than one locks, the

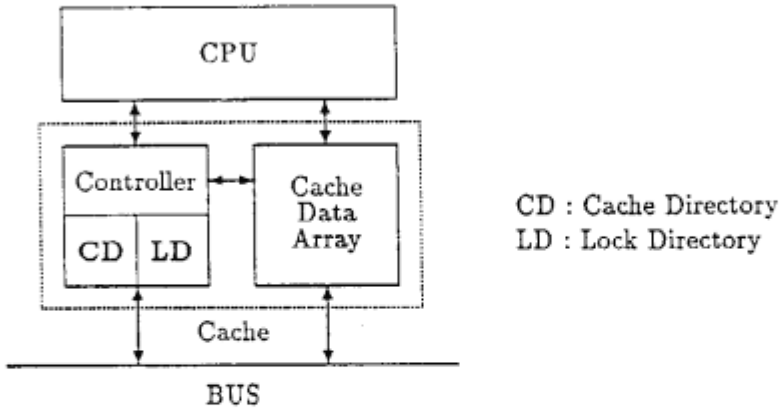

Figure 2: Cache and Lock Mechanism

Table 2: Cache States

Abbr.	Meaning
EC	Valid, Origin, Exclusive and Clean
EM	Valid, Origin, Exclusive and Modified
SC	Valid, Origin, Shared and Clean
SM	Valid, Origin, Shared and Modified
C	Valid, Copy, Shared
I	Invalid or not used

processor has to keep the incremental or decremental order for the locked-word addresses in order to avoid deadlocks.

3.3 Basic actions

We provide various processor commands suited for KL1 execution as in Table 4. The corresponding operations of cache and lock mechanism differ with the CD and LD states. The cache states *EM* and *EC* are used to avoid the bus traffic caused by useless *invalidate* and *lock* commands to other cache. Additionally, the cache status *EC* and *SC* are used to avoid unnecessary *write-back* operations.

The processor commands, *dw* and *rb*, are provided for the monotonous memory consumption mentioned above. When the cache receives *dw* commands with a written word and its address from the processor, the cache checks the address. If the address misses the cache directory but matches the cache block boundary, the word is written directly into a new cache block without fetching a block from the shared memory or other cache. When the processor reads one-write-one-read buffers, the read buffer command *rb* is used. This command fetches a block, invalidating the source block on other cache. The fetched block is purged after the processor finishes reading the content.

The read with lock command *lr* acts as follows. The lock operation on the lock directory depends on the cache block status. When miss-hit occurs, the *lock*, *fetch* and *invalidate* commands are broadcasted

Table 3: Lock directory states

Abbr.	Meaning
L	Locked without waiter
LW	Locked with waiter
E	not used

Table 4: Processor commands

Abbr.	Meaning	Operate On
r	Read	CD
w	Write	CD
dw	Write directly without fetch	CD
rb	Read one-write-one-read buffer	CD
lr	Read with lock	CD,LD
uw	Write with unlock	CD,LD
u	Unlock	LD

through the common bus to query other caches as to whether the word can lock or not and to fetch the block. When the word address hits the cache of *SM*, *SC* or *C*, the *lock* and *invalidate* commands are broadcasted. On the other hand, the cache status is either *EC* or *EM*; there is no other cache which has the locked word. Therefore, the processor simply sets the locked word address in the lock directory without broadcasting any bus command.

4 Evaluation and Discussion

A software simulator was developed for evaluating the above cache and lock mechanism. Then we examined the dynamic characteristics such as bus traffic and cache hit-ratio on several sample programs.

The cache states, *EM* and *EC* can avoid most of the bus commands *invalidate* and *lock*, reducing the bus traffic by about 60~70%. In addition, the processor command *dw* reduces the bus traffic about 30%. On the other hand, *modified/clean* does not reduce traffic so much. This is because most cache blocks containing goal environments are already modified. However, the effect of *modified/clean* depends on the program code. More detailed evaluation is necessary.

5 Conclusion and Further Work

The cache and lock mechanism was designed based on the data access characteristics of KL1 programs. This mechanism shows good performance on KL1 parallel execution by the shared heap model. More detailed hardware design and evaluation will be performed.

Acknowledgment

The research and development described in this article is being conducted mainly by the members of the PIM groups both in the ICOT Research Center and the participating companies. We also wish to thank ICOT Director Dr. Kazuhiro Fuchi and Dr. Shunichi Uchida for valuable suggestions and guidance.

References

[1] P. Bitar and A. M. Despain. Multiprocessor cache sychronization. In *Proc. of the 13th Annual International Symposium on Computer Architecture*, June 1986.

[2] T. Chikayama and Y. Kimura. *Multiple Reference Management in Flat GHC.* Technical Report, ICOT, 1986. To appear as ICOT Technical Report.

[3] M. Sato et al. *KL1 Execution Model for PIM Cluster with Shared Memory.* Technical Report, ICOT, 1986. To appear as ICOT Technical Report.

[4] R. H. Katz et al. Implementing a cache consistency protocol. In *Proc. of the 12th Annual International Symposium on Computer Architecture*, June 1985.

[5] J. R. Goodman. Using cache memory to reduce processor-memory traffic. In *Proc. of the 10th Annual International Symposium on Computer Architecture*, 1983.

[6] A. Goto and S. Uchida. *Toward a High Performance Parallel Inference Machine -The Intermediate Stage Plan of PIM-*. TR 201, ICOT, 1986.

[7] N. Ichiyoshi, T. Miyazaki, and K. Taki. *A Flat GHC Implementation on the Multi-PSI.* Technical Report, ICOT, 1986. To appear as ICOT Technical Report.

[8] K. Taki. The parallel software research and development tool : Multi-PSI system. In *France-Japan Artificial Intelligence and Computer Science Symposium 86*, October 1986.

[9] K. Ueda. *Guarded Horn Clauses.* TR 103, ICOT, 1985.

[10] David H.D. Warren. *An Abstract Prolog Instruction Set.* Technical Note 309, Artificial Intelligence Center, SRI, 1983.