TR-327

Locally Parallel Cache Design Based on KL1 Memory Access Characteristics

by

A. Matsumoto, T. Nakagawa, M. Sato,
Y. Kimura, K. Nishida & A. Goto

November, 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

Locally Parallel Cache Design Based on KL1 Memory Access Characteristics

Akira MATSUMOTO, Takayuki NAKAGAWA, Masatoshi SATO,
Yasunori KIMURA, Kenji NISHIDA and Atsuhiro GOTO

Institute for New Generation Computer Technology (ICOT)

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 (comprised of about 8 processing elements) with shared global 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.

This paper describes a locally parallel cache and hardware lock mechanism to enable quick and exclusive accesses to the shared global memory. The most important issue is how to reduce common bus traffic. A write-back cache protocol having five cache states designed for KL1 execution on each PIM cluster is described. A hardware lock mechanism is attached to the cache on each processor. This lock mechanism enables efficient word by word locking, reducing common bus traffic by using the cache states. Finally, locally parallel cache and hardware lock mechanisms, and memory access characteristics of KL1 are evaluated with measurements obtained by software simulation.

1. Introduction

ICOT is promoting the research and development of the parallel inference machine PIM[Goto]. PIM has a hierarchical structure with a dozen or more clusters (Figure 1). Each cluster consists of eight or more processing elements (PE) which communicate through shared global memory (GM) over a common bus. A parallel logic programming language KL1 (Kernel Language 1[Kimu]) which is based on GHC (Guarded Horn Clauses[Ueda]), is the target language of PIM.

Focusing on KL1 parallel execution in each cluster, quick and exclusive accesses to shared data is one of the key issues. We introduced a locally parallel cache mechanism to quickly access shared data. Several cache protocols have been proposed in the literature[Arch,Bita,Good,Katz,Papa,Swea]. Each protocol aims to solve the so-called cache coherency problem and reduce common bus traffic.

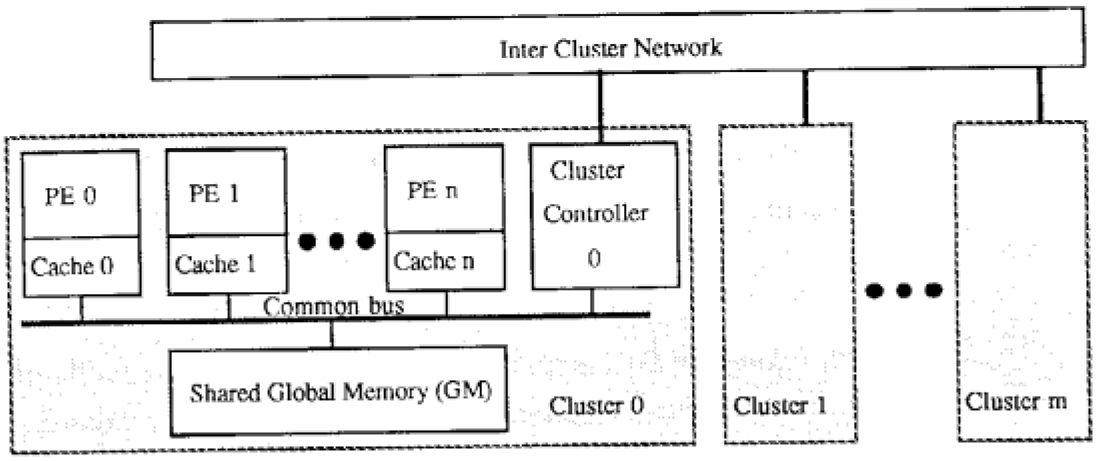


Figure 1. Configuration of Parallel Inference Machine (PIM)

Several multiprocessor systems are commercially available. Some of them, e.g. Balance 8000/21000[Sequ], have shared global memory with coherent cache, and aim to increase performance under the multi-user UNIX environment. However, the parallel processing in a PIM cluster differs from these general-purpose multiprocessors. The parallel processing granularity in Balance system is very large, e.g. UNIX processes. On the other hand, parallel processes in PIM are one or more goal reductions as mentioned in section 2.2. The kernel language KL1 based on GHC[Ueda] is the system programming language in PIM instead of the conventional language, e.g. C. The single-assignment manner of KL1 causes the different memory access characteristics from programs written in conventional languages. Therefore, we should clarify what kind of cache protocol is most suitable for KL1 parallel execution.

Cache design and evaluation is done as follows. First, all memory access histories are retrieved from a pseudo parallel KL1 simulator, and stored as an address trace file. Next, the memory access histories are input into a cache memory simulator. Then the memory access characteristics of KL1 sample programs are evaluated from various viewpoints. Finally, the KL1 language processor as well as the cache protocols are redesigned.

The next section describes parallel processing in a PIM cluster and clarifies the memory access characteristics. Then the key issues of the parallel cache and lock mechanisms for a PIM cluster are discussed followed by the detail of cache and lock protocols. Finally, the memory access characteristics of KL1 are evaluated by a KL1 language simulator and cache memory simulator.

2. Parallel Processing in a PIM Cluster

2.1 Data structures in KL1 execution

KL1 is a logic programming language enabling parallel programming. Clauses in KL1 programs are selected in a pattern-driven manner as in Prolog, however, unification of logical variables are performed in a single assignment manner[Ueda]. Parallel processing is described in KL1 programs as follows: programmers can describe various processes of flexible size in KL1, communications among such processes are realized using logical variables, and KL1 has simple language principles for parallel process synchronization.

The following data structures are used in KL1 goal reduction. Parallel goals are represented by goal records and their environments. Goal records includes atomic goal arguments or pointers to their environments consisting of logical variable cells or structures. The reducible goal records are stored as 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[Sato]. The meta-call records form a tree-like structure, 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[Sato]. Each processing element has its own ready queue. From the logical viewpoint, goal records are not shared even if they are stored in the physical shared global memory. On the other hand, goal environments, meta-call records, and suspension records are shared among processing elements. Clauses in KL1 programs are compiled into WAM[Warr] like machine instructions, called KL1B[Kimu]. Each processing element dequeues a goal record from its ready queue, then performs goal reductions by executing the corresponding instructions, accessing to the goal environment in the shared global memory.

2.3 Memory access characteristics and requirements for cache

Parallel goals are distributed in a PIM cluster. Then each processing element executes goal reductions. There are many different features in memory accesses from multiprocessor systems like Balance system[Sequ]. First, because parallel goals share logical variables, the processing elements communicate more often with each other through the logical variables than the usual parallel processing on Balance system. Therefore it is important for locally parallel cache to have an efficient cache-to-cache data transfer mechanism as well as to work as a cache of shared global memory. Next, there are many exclusive accesses to communicate through shared logical variables[Sato]. Table 5 shows memory access characteristics of a benchmark program. It shows about 3.5% of total memory accesses are the read operations with locks. However, we can expect that the exclusive memory accesses seldom conflict with each other[Sato]. Therefore we should design a lock mechanism which works efficiently, at least when lock does not conflict with other locks. Finally, because of the single-assignment feature of KL1, data write frequency is higher than conventional languages[Smit]. For example, Table 5 shows the data write frequency is almost comparable to the read data frequency. So, it is necessary to decrease to write back from cache to shared global memory.

3. Locally Parallel Cache Design for PIM

3.1 Key issues in write back cache design

In a shared global memory architecture, reducing common bus traffic is a more important design factor than miss ratio[Good]. KL1 needs more write accesses than conventional languages as described later. It is clear that a write-back protocol can reduce common bus traffic more than a write-through protocol[Good,Arch]. Write-back protocols are classified into four groups as shown in Table 1[Arch,Bita,Papa].

Table 1. Classification of write back protocols

	Invalidation	Broadcast write
No write-back in case of transferring a dirty block	BERKELEY[Bita] 5 states model	DRAGON[Arch]
Write-back in case of transferring a dirty block	ILLINOIS[Papa]	FIREFRY[Arch]

The horizontal classification is: in the case of writing to a shared cache block, invalidating cache blocks of other processing elements (invalidation method), or broadcasting the written data to other processing elements simultaneously (broadcast-writing method). The invalidation method lowers common bus traffic in the case when latency of shared block write-accesses is low. On the other hand, the broadcast-writing method is better in the case when many processing elements frequently write data in the same shared block[Arch,Bita]. Considering the single assignment feature of KL1, most logical variables are shared by two KL1 goals. In other words, most inter-processing element communications are one-to-one with relation to heap and goal records accesses. It means the broadcasting is not necessary in most of KL1 parallel processing. On the other hand, all processing elements often read meta-call records to check logical results (e.g. success/failure) of parallel goals. However, the memory accesses to meta-call records are few as in Table 5. Therefore, the invalidation method is suitable for PIM cache.

The vertical classification of Table 1 is: in the case of transferring modified blocks between processing elements, no writing back to shared global memory, or writing-back to shared global memory. It is obvious that no writing-back to shared global memory can lower the busy ratio of shared global memory modules. On the other hand, writing back to shared global memory can decrease modified cache blocks. As mentioned in section 2.3, not only write frequency is higher than conventional languages, but also cache-to-cache data transfer often occurs in the parallel processing of KL1. Therefore, the no-writing back to shared global memory protocol is suitable for PIM cache.

3.2 Optimization for KL1

In normal write operations, a fetch-on-write strategy is used. i.e., a cache block is allocated in the cache on both read and write misses. In the parallel processing of KL1, new data structures are created dynamically by using heap area monotonously[Kimu,Sato]. Therefore, it is not necessary to fetch-on-write when new cache block is allocated for a new data structure, i.e. processing element can write data in a cache block without fetching the contents of shared global memory. We call this kind of write operation as "direct write[1]". Although the "direct write" operation should be used with the restrictions (see section 4.1), it is effective for the parallel processing of KL1 (see section 5.8).

When KL1 goals are distributed for load balance of processing elements, a processing element sends goal records to another processing element using a communication buffer . A one-write-one-read rule is strictly kept for the communication buffer by two processing elements. In this case, the contents of buffer both in sender's cache and in receiver's cache are useless after the receiver processing element reads the contents. In other words, by invalidating the sender's cache block

[1] This kind of write commands is used in a sequential inference machine PSI[Taki].

after cache-to-cache transfer and by purging the receiver's cache block after the receiver processing element finishes reading, meaningless swap-out and swap-in can be avoided. To accomplish this, we provide a new CPU command called "exclusive read (see section 4.1)". This "exclusive read" command is used with the "direct write" command. For example, the sender processing element creates a goal record in communication buffer area by "direct write" commands without fetching contents of shared global memory. Then the receiver processing element reads in the contents of buffer by "exclusive read". Although the "exclusive read" should be used carefully not to confuse cache coherency (see section 4.1), it can reduce common bus traffic by avoiding useless swap-in and swap-out.

3.3 Key issues in lock operations of KL1

Lock operations are essential in shared global memory architectures. The KL1 language processor uses lock operations for heap area and meta-call record area accesses[Sato]. Although actual lock conflicts seldom occur, the lock latency of a shared environment is high as described in section 2.3. Therefore it is necessary to introduce a hardware lock mechanism.

The lock operation, in general, needs more bus cycles to exclusively control among the processing elements like a write operation. Then we decided to use busy-wait locking, because the busy-wait locking scheme allows locking and unlocking to occur in zero time[Bita]. The busy-wait locking can decrease bus cycles by two cache states: *exclusive* and *lock-waiter*[Bita]. However, if the lock information is held in the cache directory (cache-state locking[Bita]), there are some concerns. For example, it is difficult to simultaneously lock more than the number of cache sets and to distinguish every locked word in the same cache block. Therefore we introduced a lock directory in addition to the cache directory. The lock directory contains a locked address and states.

4 Locally parallel cache and lock mechanisms for PIM cluster

4.1 CPU commands for memory operations

The following five memory operation commands are prepared (excluding lock related operations which are discussed in the next section).

(1)R(address): Read

Read data from a specified address of shared global memory.

(2)W(address): Write

Write data to a specified address of shared global memory.

(3)DW(address): Direct Write

A direct-write command (DW) writes data to unused memory area without fetching a cache block to avoid swap-in overhead. DW must be used keeping the following restriction. DW can be applied only when a cache block entry never exists in a cluster. When a processing element allocates a new data structure in heap or creates a new goal record, it writes data by DW into unused memory area incrementally. Then the cache controller interprets the DW commands as the following two ways: When the memory address is just a cache block boundary and then cache-miss occurs, a new cache block is allocated and the data is written into the block without fetching from global memory. In this case, it must be clear that other processing elements in a cluster does not have the corresponding cache block. This restriction is necessary to guarantee the cache coherency. On the other hand, when the memory address is not a cache block boundary, or when the address hits the cache, the DW command is automatically replaced with a usual write command by the cache controller.

(4)ER: Exclusive Read

Exclusive Read (ER) command is used when the contents of a cache block are not necessary after the processing element reads it in registers, e.g. when the processing element reads one-write-one-read communication buffers or goal records.

ER acts three different ways according to the relative position of a cache block as follows: When the target address is not the last word of a cache block and a cache misses, a cache-to-cache transfer from a supplier processing element occurs. In this case, the supplier cache block is invalidated after the data block transfer. When the target address hits and it is the last word, after the last word of a cache block is read, the cache block in the receiver cache is forcibly purged as the same way as read purge (RP) described below. In other cases, ER is automatically replaced with a usual read operation.

(5)RP: Read Purge

The RP command is used when a cache block cannot be purged by ER. That is when word number of reading area is not multiples of cache block word size. In this case the last word of reading area is read by RP.

4.2 CPU commands for lock operation

We prepared an explicit unlock (U) operation in addition to lock and read (LR) and write and unlock (UW) operations proposed by [Bita].

(1)LR(Address): Lock and Read
Read data from a specified address of shared global memory with lock operation.

(2)UW(Address): Write and Unlock
Write data to a specified address of shared global memory with unlock operation.

(3)U(Address): Unlock
Unlock a specified address.

4.3 Five cache states

On a sequential machine, (i)Valid/Invalid and (ii)Clean/Modified states are required to implement a write-back protocol[Hwan]. In the case of the locally parallel cache for PIM, we added a third field (iii)Exclusive/Shared to guarantee coherency and reduce common bus traffic as previously proposed protocols[Arch,Bita,Katz,Papa,Swea]. We distinguish between the following five cache states:

(1)EM: Valid, Exclusive, Modified
The block is exclusive and modified. It is necessary to swap-out.

(2)EC: Valid, Exclusive, Clean
The block is exclusive and clean (unmodified). It is not necessary to swap-out.

(3)SM: Valid, Shared, Modified
The block is modified and maybe shared. It is necessary to swap-out.

(4)S: Valid, Shared
The block is maybe shared. It is not necessary to swap-out.

(5)I: Invalid
The block is invalid or unused.

4.4 Three lock states

We distinguish between the following L and LW states proposed by [Bita]. In addition, an E state is used in a lock directory:

(1)L: Lock
The address is locked by this processing element.

(2)LW: Lock-Waiter

The address is locked by this processing element. In addition, one or more processing elements are waiting for unlock.

(3)E: Not Locked (Empty)
Unused entry.

4.5 Bus commands and a response for memory operations

There are three bus commands and one response to implement a locally parallel cache mechanism having five cache states. There is no additional bus command and response for DW, ER and RP.

(1)F(address): Fetch
A request to fetch a cache block from other processing elements or shared global memory.

(2)FI(address): Fetch and Invalidate

A request to fetch a cache block from other processing elements or shared global memory, and to invalidate cache blocks of all other processing elements including the supplier processing element of a cache-to-cache transfer.

(3)I(address): Invalidate
A request to invalidate cache blocks of all other processing elements.

(4)H: Hit
A hit response for F and FI requests.

4.6 Bus commands and a response for lock operations

Two bus commands and one response are necessary to implement busy-wait locking[Bita] in addition to above bus commands and a response.

(1)LK(address): Lock

This is a bus command to lock a specified address. If a cache state is *exclusive* (EM or EC), in other words there is no copy of the block in other processing elements, the LK bus command is not broadcast[Bita].

(2)UL(address): Unlock

This is a bus command to unlock a specified address. If the locked address is not in a LW state, in other words another processing element does not refer to the address, the UL bus command is not broadcast[Bita].

Table 2. State transition table of 5 states model (CPU commands)

State \ CpuCmd	EM	EC	SM	S	I	Remark
R(a)	-/EM	-/EC	-/SM	-/S	SO,F(a)/ H(d)->S GM(d)->EC	Read Other Cache Hit Swap-in from GM
W(a)	-/EM	-/EM	I(a)/EM	I(a)/EM	SO,FI(a)/ H(d)->EM GM(d)->EM	Write Other Cache Hit Swap-in from GM
DW(a)	MCHK	MCHK	MCHK	MCHK	SO/EM	Direct Write
RI(a)	-/EM	-/EC	MCHK	MCHK	SO,FI(a)/ H(d)->EM GM(d)->EC	Read Invalidate Other Cache Hit Swap-in from GM
RP(a)	-/I	-/I	MCHK	MCHK	SO,FI(a)/I	Read Purge
LR(a)	-/EM	-/EC	LK,I(a)/ EM	LK,I(a)/ EM	SO,LK,FI(a)/ H(d)->EM GM(d)->EC	Lock Read Other Cache Hit Swap-in from GM
UW(a)	(UL)/EM	(UL)/EM	MCHK	MCHK	SO,(UL),FI(a)/ EM	Unlock Write Swap-in from GM
U(a)	(UL)/EM	(UL)/EM	MCHK	MCHK	(UL)/I	Unlock

Abbreviations:

XXX/YYY: issuing a XXX bus command and transits to a YYY state
F(a),FI(a),I(a): issuing F,FI,I bus commands with block address, a
H(d)->YYY: receiving a cache block data through a cache-to-cache transfer because it hits to an other processing element cache. The cache state transits to a YYY state
GM(d)->YYY: receiving a cache block through a swap-in from Shared Global Memory (GM) and transits to a YYY state
MCHK: machine check (This state is detected as a hardware error.)
SO: if a cache entry which should be used is in a modified state, swap-out to Shared Global Memory will occur
(UL): if a cache entry which should be used is in a lock-waiter (LW) state, an unlock (UL) bus command will be issued.

Table 3. State transition table of 5 states model (BUS commands)

State \ BusCmd	EM	EC	SM	S	I	Remark
F(a)	H(d)/SM	H(d)/S	H(d)/SM	H(d)/S	-/I	Fetch
FI(a)	H(d)/I	H(d)/I	H(d)/I	H(d)/I	-/I	Fetch Invalidate
I(a)	-/I	-/I	-/I	-/I	-/I	Invalidate

Abbreviations:

XXX/YYY: issuing a XXX response and transits to YYY state
H(d): sending a cache block data through a cache-to-cache transfer because it hits in another cache.

(3)LH: Lock Hit

[illegible] a response to the F, FI, and LK bus commands. The LH response shows that the referred address is locked. The requesting processing element which received the LH response starts busy-waiting. The requesting processing element retries a memory reference after it receives the UL bus command. The common bus is never used during busy-waiting cycles[Bita].

4.7 Features of the locally parallel cache and lock mechanisms

A state transition table for CPU commands is shown in Table 2, and a table for bus commands is shown in Table 3. The features of our protocol are summarized as follows:

(1) invalidate other caches in the case of writing to shared blocks
(2) no writing-back to shared global memory in the case when a dirty block is transferred on the common bus
(3) any cache block can become the supplier of a cache-to-cache transfer
(4) the modified (M) field does not move to a destination cache block, in the case of the cache-to-cache transfer of normal read operations. But the modified field is transferred in the case of the write or read lock operations
(5) busy-wait locking.

5. Evaluation of KL1 memory access characteristics

5.1 Evaluation methodology

We have developed a pseudo parallel KL1 simulator on a sequential machine. Accuracy of the simulator is at the reduction[1] level. This pseudo parallel KL1 simulator executes one reduction of a processor, then switches to a next processor. We think that the accuracy of the simulator is sufficient to compare cache protocols specified for KL1 execution. In the future, however we plan to make a machine cycle level simulator for further evaluation.

Evaluation was done in the following steps. First, we added probes to collect all memory access histories in the reduction level KL1 simulator. A record for one memory access consists of processing element number, access mode (such as read/write/lock), area name (We distinguished following six areas: heap, instruction, goal record, suspension record, meta-call record, communication buffer), and address. Next, the simulator executed a benchmark program and generated memory access histories to a trace file. Finally we input memory access histories of a benchmark program to a cache memory simulator and evaluated memory access characteristics from various viewpoints. We assumed that all memory areas were stored in one shared global memory. We did not assume a local memory in this simulation.

5.2 Configuration of locally parallel cache

Cache memory capacity for each processing element was assumed to be 4 kilowords. The

[1] See section 2.

standard configuration for each cache memory is 4 sets, 256 columns, with 4 word blocks. We believe this size is a realistic value for a PIM processing element. The number of processing elements was assumed to be eight, which is the target number of processing elements in a cluster. A common bus is used for swap-in from shared global memory, swap-out to shared global memory, cache-to-cache transfer between processing elements, and invalidation. Major assumptions of hardware parameter are as follows:

(1) width of the common bus is a word (a word consists of tag and data part). An address bus and a data bus are not distinguished. Therefore it is assumed that an address cannot be sent with data during the same cycle.

(2) it takes eight cycles to access shared global memory. However the swap-out write operation to shared global memory is hidden by a subsequent memory operation.

(3) the common bus is not freed until one memory operation is completed.

We assumed the following six common bus access patterns:

(1) swap-in from shared global memory with swap-out takes 13 cycles

(2) swap-in from shared global memory without swap-out takes 13 cycles

(3) cache-to-cache transfer with swap-out takes 10 cycles

(4) cache-to-cache transfer without swap-out takes 7 cycles

(5) only swap-out takes 5 cycles (This access pattern appears only in DW.)

(6) invalidation of other processing elements' cache blocks takes 2 cycles.

The nominal bus usage ratio is an estimate of the common bus busy ratio:

$$\text{Nominal bus usage ratio} = \frac{\text{common bus usage time}}{\text{execution time}}$$

Common bus usage time was calculated multiplying bus cycles (described above) by bus cycle time (assumed 50 nanosecond). Execution time was calculated dividing reduction counts by target performance (assumed 200K [RPS: Reduction Per Second]). Therefore nominal bus usage ratio means bus usage ratio excluding bus waiting time because of bus contention. Although the nominal bus usage ratio is a rough estimate, it is sufficiently accurate to evaluate the design parameters.

5.3 Characteristics of a benchmark program

We selected a simple BUP (Bottom Up Parser) program as a benchmark . The BUP program generates parse trees by analyzing natural language sentences. We think that this kind of program will become one of the PIM application programs. General characteristics of the BUP program are shown in Table 4. We plan to increase the varieties of benchmark programs in the future.

Table 4. Characteristics of a benchmark program

Benchmark program	BUP
Static	
Line number of the source program	3.2K
Number of static KL1B* instructions	3.3K
Dynamic	
KL1B* instruction words executed more than once	1.9K
Executed KL1B instructions	545.4K
Number of total reductions	35.8K
Number of suspensions	1.6K
Number of total memory references	1,312.9K

* KL1 Base: Machine instruction of KL1

Table 5. Memory access characteristics

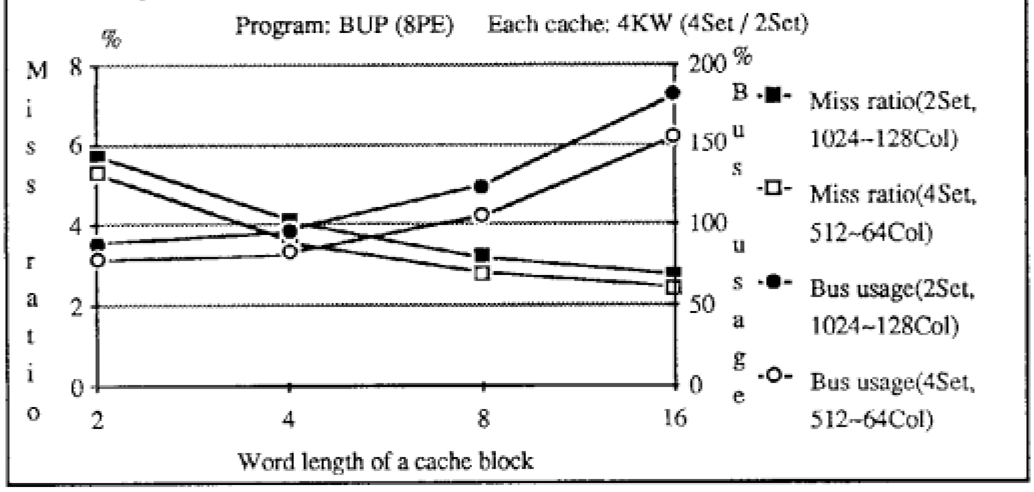
area	Write operation	Read operation	Read with lock	Total
Environment (Heap)	7.5%	4.3%	2.9%	14.7%
KL1B instruction	0	49.5%	0	49.5%
Goal record	13.9%	13.8%	0	27.7%
Suspension record etc	1.1%	1.1%	0	2.2%
Meta-call record etc	0.6%	2.7%	0.6%	3.9%
Communication buffer	1.0%	1.0%	0	2.0%
Total	24.1%	72.4%	3.5%	100.0%

The Table 5 shows memory access characteristics of the BUP program. About 50% of the memory references are used for fetching instructions. This value is high because our simulator does not use clause indexing[Warr]. Data write frequency is 50%. This value is almost the same as in Prolog execution[Tick]; however, it is higher than conventional languages[Smit]. Access frequency to the shared heap area is about 15% of all memory accesses.

5.4 Nominal bus usage ratio and cache miss ratio

Figure 2 shows the relation between the nominal bus usage ratio and cache miss ratio, in the case when the capacity of a cache data array is the same. Four sets, 512 columns with 2 word blocks is the best configuration in Figure 2 from a viewpoint of nominal bus usage which is more important than miss ratio. However 2 word blocks use about twice the size of cache address array as compared to 4 word blocks. Moreover the difference in nominal bus usage ratio between 2 and 4 word blocks is relatively small. Therefore we selected 4 word blocks. In the case when word length of a cache block is fixed at 4 words, the number of bus cycles are compared in the following two configurations. The first configuration is 4 sets, 256 columns, and the second configuration is 2 sets, 512 columns. The number of used bus cycles of the 2 sets configuration is about 16% more than the 4 sets configuration. Therefore the 4 sets configuration is most effective.

Fig 2. Cache miss ratio and nominal bus usage ratio



5.5 Cache capacity and nominal bus usage ratio

Figure 3 shows the bus usage ratio as a function of cache capacity. There are two kinds of memories in the cache. One is an address array that contains a cache directory . The other memory is a data array that contains cache data. To reduce machine cycle time, the address array should be on an LSI chip. In this situation it is important to reduce the chip size of the address array. The desired way is to fix the capacity of the address array and increase the capacity of data array. However if the size of a cache block is more than 8 words, the nominal bus usage ratio becomes larger in spite of the increase in the capacity of the data array. Therefore the best size of a cache block is 4 words.

Fig3. Cache capacity and nominal bus usage ratio

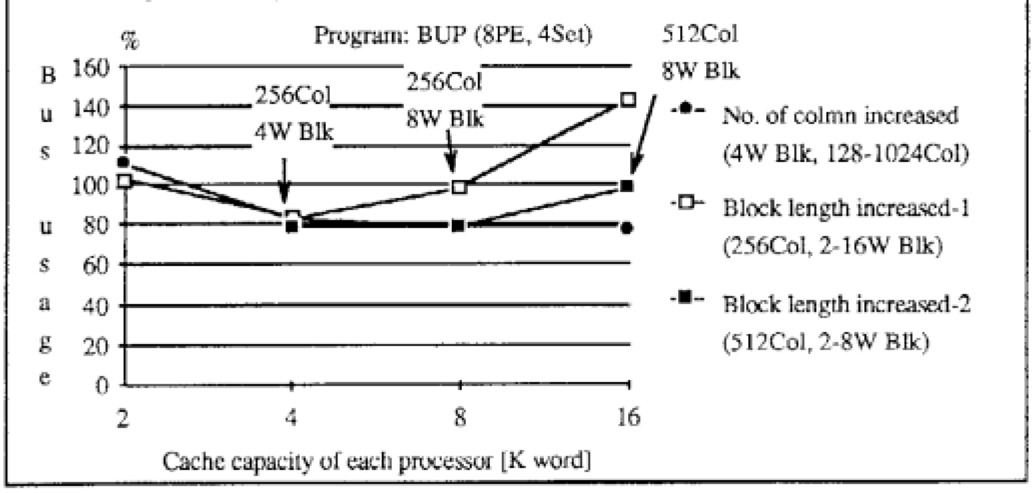
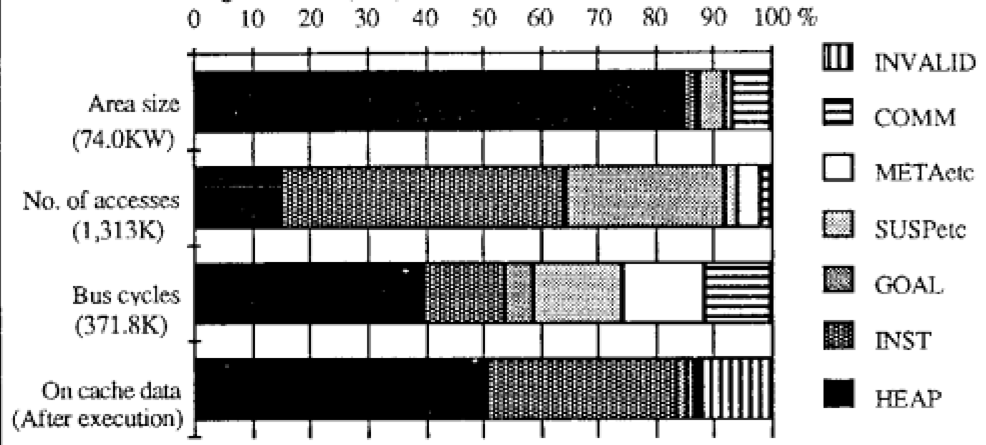


Fig. 4 Memory access characteristics of each area
Program: BUP (8PE) Each cache: 4KW (4Set, 256Col, 4W Blk)



5.6 Memory access characteristics of each area

We distinguished between the following six areas in the KL1 simulator: heap area, instruction area, goal record area, suspension record area, and communication buffer area. Figure 4 shows area size, number of references, number of bus cycles, and contents of cache memory snapshot after execution of each area.

Heap area occupies about 85% of dynamic memory space. However, the number of heap memory accesses is only about 15% of all memory accesses. Therefore access characteristics of the heap area have lower locality compared to other areas. Actually nearly 40% of bus cycles are used to access the heap area. On the other hand, instruction area occupies about 2.6% of memory space. However about a half of memory accesses are instruction reads. Goal record area occupies 28% of dynamic memory accesses. This is because a goal record includes its argument list, e.g. atomic arguments or pointers to structures in heap. However, goal record area occupies only 4.5% of dynamic memory space. In addition, goal records are reused again and again by free-lists managing of its own processing element. Then goal record area occupies only about 5% of bus cycles.

Memory access characteristics of the heap area has a large effect on KL1 language processing because address space of the heap area is very large and access locality is very low. We examined more precise characteristics of the heap area as shown in Table 6. Table 6 shows each heap word is accessed three times in average. The number of write accesses and read accesses is almost the same. In addition, about 40% of read accesses were read with lock operations. Therefore it is confirmed that a high speed lock mechanism using hardware support is necessary.

Table 6. Dynamic access characteristics of the heap area

Program (BUP) Number of reductions=35,752	Total words	Heap words per a reduction
Number of dynamically used heap area	62.8K	1.8 [word]
Number of references to heap	192.8K	5.4
Number of writing to heap	98.1K	2.7
Number of reading from heap	94.6K	2.6
Reading with lock operation	38.0K	1.1

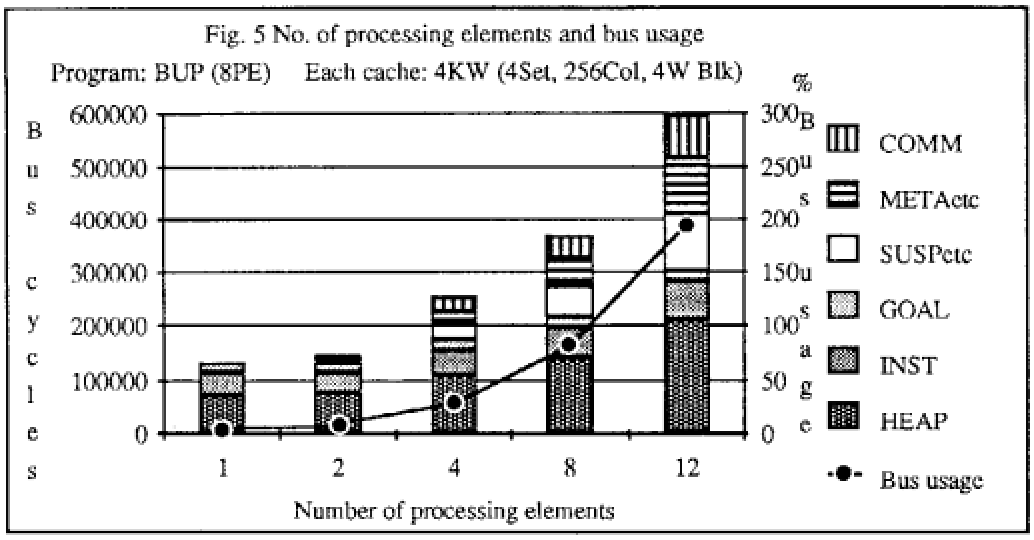


Fig. 5 No. of processing elements and bus usage

5.7 Effect of the number of processing elements

Figure 5 shows the relationship between the number of processing elements and the number of bus cycles. It also shows the relationship between the number of processing elements and the nominal bus usage ratio. In the case of increasing the number of processing elements, the rising ratio of the nominal bus usage ratio is higher than the rising ratio of the number of bus cycles. This is because execution speed is assumed to simply become *n* times faster if the same program is executed by *n* processing elements. There is a prospect of a connection of about four to eight high performance processing elements for one current specification common bus.

The capacity of a cache memory is assumed as 4K words for each processor. Then, if the number of processing elements increases, total capacity of cache memory also increases. Figure 5 shows the bus traffic caused by accessing heap area and suspension area increases as the number of processing elements increases, even if total capacity of cache memories increases. It means that inter processing elements communication is a dominant factor in the parallel processing by more than four or eight processing elements. For example, in case a BUP program is executed by eight processing elements, about 10% accesses of heap memory are accesses to heap area allocated by other processing elements.

Fig. 6 Effect of DW/ER commands

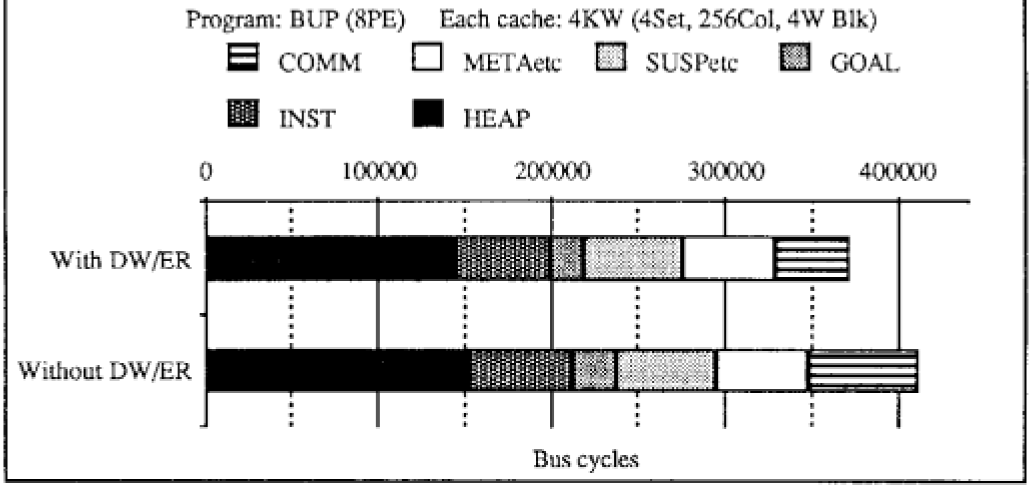
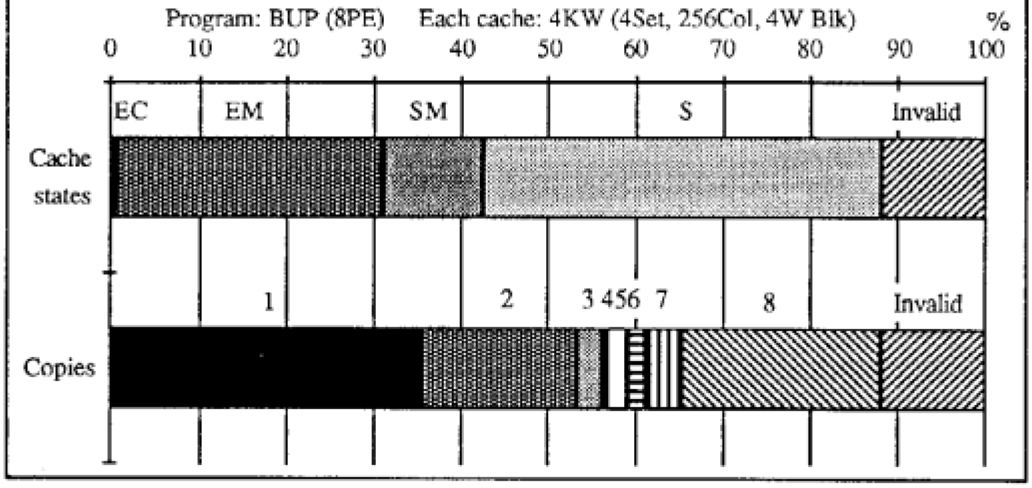


Fig. 7 Cache states snapshot of after execution



5.8 Effect of the DW, ER and RP commands

Direct write (DW), exclusive read (ER) and read purge (RP) commands are used to manage goal records area and communication buffer area as mentioned in section 3.2. Figure 6 shows the DW, RP and ER commands are effective. If DW, RP and ER commands are replaced by simple write and read commands, nominal bus usage ratio increases about 11%.

5.9 Analysis of cache states snapshot of after execution

Figure 7 shows cache states snapshot and the number of processing elements which shared the same cache block after execution was finished. The number of invalid cache block entries comes up to about 12%. Invalidated entries are made when another processor writes data into shared blocks or reads shared data by LR, ER or RP commands. Although these invalid cache block entries seem to be useless, invalid entries can be reused without swap-out overhead.

We examined the number of copied blocks after execution, and found that exclusively used blocks by one processing element was about 36%. On the other hand, about 23% blocks is shared by all eight processing elements. This is because instruction area is shared by many processing elements.

6. Conclusions

The paper describes the parallel inference machine (PIM) being developed at ICOT. PIM has a hierarchical structure using the cluster concept. Each cluster has a shared global memory. In PIM, we have found that the most important design factor is reducing common bus traffic. In our experiences, a write-back protocol is necessary because KL1 has many write accesses. We evaluated memory access characteristics of a KL1 benchmark program by simulations. In KL1 execution, the data write frequency is about 50%. Although this value is almost the same as Prolog execution[Tick], it is higher than conventional languages[Smit]. Access characteristics of the heap area have lower locality compared to other areas and each heap word is accessed only three times in average. On the other hand, instruction and goal record area have more locality.

In normal write operations, a fetch-on-write strategy is used. In the parallel processing of KL1, new data structures are created dynamically by using heap area monotonously[Kimu,Sato]. Therefore, it is not necessary to fetch the content of shared global memory when new cache block is allocated for a new data structure. We call this kind of write operation as "direct write". Although the "direct write" operation must be used with restrictions, it is effective for the parallel processing of KL1. When KL1 goals are distributed, a processing element sends goal records to another processing element using a communication buffer. In this case, by invalidating the sender's cache block after a cache-to-cache transfer and by purging the receiver's cache block after the receiver finishes reading, meaningless swap-out and swap-in can be avoided. To accomplish this, we provide a new CPU command called "exclusive read". This "exclusive read" command is used with the "direct write" command. For example, the sender creates a goal record in a communication buffer by "direct write" commands without fetching the contents of shared global memory. Then the receiver reads in the buffer contents by "exclusive read". This can reduce common bus traffic by avoiding useless swap-in and swap-out. If "direct write" and "exclusive read" are replaced by simple write and read commands, the number of bus cycles increases about 11%.

Lock operations are essential in shared global memory architectures. The KL1 language processor uses lock operations for heap area and meta-call record area accesses[Sato]. In a benchmark execution, about 40% of the heap read accesses are read with lock operations. Although actual lock conflicts seldom occur, the lock latency of shared environments is high. Therefore it is necessary to introduce a hardware lock mechanism and we decided to use busy-wait locking, because the busy-wait locking scheme allows locking and unlocking to occur in zero time[Bita].

According to the above discussions, we designed a write-back locally parallel cache mechanism having five cache sates and a busy-wait locking mechanism with a lock directory. We examined the relationship between the number of processing elements and bus cycles. As we assume a high performance processing element, the capacity of the common bus becomes a bottleneck. In the simulations, the nominal bus usage is saturated when the number of processing elements is between eight and twelve. However, to reduce bus-waiting time and get good response time, we should keep the nominal bus usage ratio less than 50%. Therefore there is a prospect of a connection of about four to eight processing elements for one current specification common bus.

In future work, we plan to increase the variety of the benchmark programs and make more precise measurements.

Acknowledgment

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

References

[Arch] J. Archibald, J. Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model", ACM Trans. on Computer Systems, Vol.4, No.4, pp.273-298, 1986.

[Bita] P. Bitar, A. M. Despain, "Multiprocessor Cache Synchronization Issues, Innovations, Evolution", Proc. of the 13th Annual International Symposium on Computer Architecture , pp.424-433, 1986.

[Good] J. R. Goodman, "Using cache memory to reduce processor-memory traffic", Proc. of the 10th Annual International Symposium on Computer Architecture, pp.124-131, 1983.

[Goto] A. Goto, "Parallel Inference Machine Research in FGCS Project", Japan U.S. AI symposium (to appear), 1987.

[Hwan] K. Hwang, F. A. Briggs, "Computer Architecture and Parallel Processing 2.4 Cache Memories and Management", McGRAW-Hill,Inc., ISBN 0-07-Y66354-8, pp.98-118, 1984.

[Katz] R. H. Katz, S. J. Eggers, D. A. Wood, C. L. Perkins, R. G. Sheldon, "Implementing a Cache Consistency Protocol", Proc. 12th Annual International Symposium on Computer Architecture, pp.276-283, 1985.

[Kimu] Y.Kimura, T.Chikayama, "An abstract KL1 machine and its instruction set", 1987 Symposium on Logic Programming, pp.468-477, 1987.

[Papa] M.S.Papamarcos, J.H.Patel, "A low-overhead coherence solution for multiprocessors with private cache memories", Proc. of the 11th Annual International Symposium on Computer Architecture, pp.348-354, 1984.

[Sato] M.Sato, et al., "KL1 Execution Model for PIM Cluster with Shared Memory", International Conference on Logic Programming '87, pp.338-355, 1987.

[Sequ] Sequent Computer Systems. Inc., "Balance Technical Summary", MAN-0110-00, 1986

[Smit] A. J. Smith, "Cache Memories", Computing Surveys, Vol.14, No.3, pp.473-530, 1982.

[Swea] P. Sweazey, A. J. Smith, "A Class of Compatible Cache Consistency Protocols and their Support by the IEEE Futurebus, Proc. of 13th Annual International Symposium on Computer Architecture, pp.414-423, 1986.

[Taki] K. Taki, K. Nakajima, H. Nakashima, M. Ikeda, "Performance and architectural evaluation of the PSI machine", Proc. Second International Conference on Architectural Support for Programming Language and Operation Systems, pp.128-135, 1987

[Tick] E. Tick, "Prolog Memory-Referencing Behavior", Stanford University Technical Report No. 85-281, 1985.

[Ueda] K. Ueda, "Guarded Horn Clauses", ICOT TR-103, 1985.

Warr] D. H. D. Warren, "Abstract Prolog Instruction Set", Technical Report 309, Artificial Intelligence Center, SRI International, Menlo Park, California 94025, 1983.