TR-394

Evaluation of the Effect of Incremental Garbage Collection by MRB on FGHC Parallel Execution Performance

by
K. Nishida, Y. Kimura & A. Matsumoto

June, 1988

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

Submitted to Compcon 89

Evaluation of the Effect of Incremental Garbage Collection by MRB on FGHC Parallel Execution Performance

Kenji NISHIDA* Yasunori KIMURA Akira MATSUMOTO Atsuhiro GOTO

Institute for New Generation Computer Technology (ICOT) †

Abstract

The reduction of bus traffic is the key issue to improve the performance of shared memory multiprocessors based on coherent cache mechanism. Committed choice logic programming languages, such as Flat GHC (FGHC), consume memory area very rapidly, and require large storage size. Such large storage size requirement affects the total system performance not only by frequent global GC, but also by increasing bus traffic especially for memory allocation.

Incremental GC reduces the bus traffic for memory allocation by decreasing the required storage size. It also reduces other bus traffic by decreasing cache misses, since the locality of memory references can be enhanced by reclamation and reuse of memory cells. Therefore, incremental GC is the key issue to improve the total system performance of committed choice logic programming language.

Incremental GC by Multiple Reference Bit (MRB-GC) has been proposed as an efficient incremental GC scheme for FGHC. The required storage size is reduced to 30 % by MRB-GC. The overheads of incremental GC, such as extra memory references for management of reference count, is reduced not to affect the performance of FGHC execution.

This paper evaluates the effect of MRB-GC on the bus traffic of FGHC execution on the shared memory multiprocessor. The evaluation result shows that memory-cache traffic is reduced more than expected from decrement of the required storage size. Yet the problem on the increase of cache-cache traffic by mutual invalidation of cache blocks is revealed.

1 Introduction

The parallel inference machine (PIM) is one of the most important targets of the FGCS project [7]. It consists of about 16 clusters connected by a network. Each cluster includes eight processing elements (PEs) which share one address space and communicate through shared memory (SM) on a common bus. Each PE has coherent

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

† 21F Mitakokusai Bldg., 1-4-28, Mita, Minato-ku, Tokyo, 108, JAPAN, Phone: +81-3-456-3193 Telex: ICOT J32964

cache memory to enable quick access to the shared memory [7]. The kernel language of PIM is designed based on Flat GHC [6, 15, 16]. Flat GHC (FGHC) is a kind of committed choice logic programming languages without backtracking.

While FGHC can describe basic operations such as synchronization and communication between parallel processes without any side effects, destructive assignment is not allowed. Therefore, naive implementation of such committed choice logic programming language consumes memory area very rapidly. As a result, garbage collection (GC) occurs frequently[1]. Cache misses and memory faults occur often during global GC, because the locality of memory references cannot be expected in widely used GC based on marking scheme.

This problem becomes more serious in the FGHC implementation on a shared memory multiprocessor such as the PIM cluster. It is because the global GC by marking scheme has following problems: (1) Bus contentions during global GC by cache misses makes it difficult to perform global GC by multiple processors; and (2) Synchronization among PEs to initiate and terminate global GC is rather costly.

Therefore, incremental GC is a key issue to improve the total system performance. By collecting garbages during execution, not only the memory consumption rate can be reduced, but also the locality of memory references can be increased, so that coherent caches can work efficiently. The overheads of incremental GC, such as extra memory references for reclamation and management of reference count, should be reduced not to degrade the system performance.

A good reclamation rate with low overhead can be expected by reclaiming single referenced memory cells, since it is empirically known that most of the references to memory cells in FGHC are single [9, 11]. Incremental GC by multiple reference bit (MRB) is proposed as an efficient incremental GC scheme (MRB-GC for short) for FGHC [2]. Reference information is maintained by only one bit flag in pointers to reduce the overhead of managing reference count. Memory cells which have multiple references are not reclaimed in MRB-GC,and global GC must therefore be used together. However, MRB-GC has 60 to 70% of reclamation ratio according to our evaluation[9, 12], it can reduce the frequency of global GC to 1/3.

This paper evaluates the effect of MRB-GC on the performance of FGHC on shared memory multiprocessors with coherent cache memory. MRB-GC is expected to reduce not only the frequency of global GC but also the bus traffic. The bus cycle count is measured by cache simulation based on the memory references of the FGHC pseudo parallel emulator. The evaluation result indicates that cache-cache traffic constitutes the majority of bus traffic, although memory-cache traffic is greatly reduced by MRB-GC.

2 Storage Management for FGHC

This section briefly describes the memory reference characteristics of FGHC and the effect of incremental GC on the performance of FGHC implementation on shared memory multiprocessor with coherent cache.

[1]In sequential PROLOG, this problem is not so serious because of the backtracking feature [14].

```
foo :- true | generator(0,Y), consumer(Y,Z).

generator(X,Y) :- true |
        X1 = X + 1, Y = [X|Y1], generator(X1,Y1).
generator(X,Y) :- X > 10000 |
        Y = [].          < termination for generator >

consumer([X|Y1],Z) :- true  |
        Z = X + Z1, consumer(Y1,Z1).
consumer([],    Z1) :- true |
        Z1 = 0.          < termination for consumer >
```

Figure 1: Example of Stream Communication in FGHC

2.1 Memory Reference Characteristics of FGHC

The execution of FGHC is based on a communication between goals using a stream. Figure 1 is an the example of communication in FGHC. In this program, generator allocates a variable cell, X1, and a cons cell, [X|Y1], to create a stream, Y, and consumer receives stream Y. The cons cell unified with [X|Y1] in consumer (which is allocated in generator) is never be referenced in consumer after the unification. Similarly, the variable cell unified with X will never be referenced after the unification in consumer. If there exist no other references to these variable cell and cons cell, these memory cells can be reclaimed incrementally in consumer.

Since FGHC does not allow destructive assignment, generator keeps allocating new memory cells until it terminates. Therefore, naive implementation of FGHC without incremental GC consumes memory area very rapidly. Such rapid consumption of memory area requires a large storage size, or it causes frequent GC. Frequent cache misses also occurs, because the locality of memory references is deteriorated. As a result, the total system performance is seriously affected.

2.2 Incremental Garbage Collection by Multiple Reference Bit

In the example in Figure 1, memory cells used in a stream are created in generator and referenced only in consumer, so that they can be reclaimed in consumer, but usually there may exist another reference to those memory cells. Thus, it is necessary to ensure that there is no other reference to the memory cell (the cell is single referenced) to reclaim a memory cell.

Implementing a reference count on for each memory cell requires an extra word for each memory cell, and, it also requires extra references to each memory cell although FGHC employs many pointer operations which does not need to refer the memory cell. Since most of the memory cells in FGHC are considered to be single referenced, a good reclamation rate can be expected by reclaiming only memory cells that has been single referenced.

Incremental garbage collection by *Multiple Reference Bit* (MRB-GC for short) [2] is proposed as an efficient incremental GC scheme for FGHC. MRB is one-bit pointer tag which holds multiple reference information, and the references to memory cells

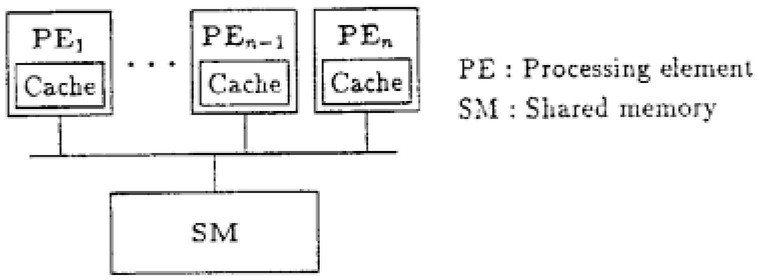

Figure 2: A Shared Memory Multiprocessor

just for management of reference count can be avoided. MRB reduces the overhead of reference count management, thereby reducing reference information to one bit (*single* or *multiple*). As a result, MRB cannot manage the reference count of *multiple* referenced cells. Thus, MRB-GC cannot reclaim memory cells once they are multiple referenced, although it may return to single referenced memory cells.

Although MRB-GC must be used together with global GC because of its reclamation ability, it can reduce the global GC frequency by reducing the memory consumption rate, thereby improving the total system performance of FGHC execution. Reduction of the memory consumption rate also improves the cache characteristics by enhancing the locality of memory references. Thus, performance improvement by reducing cache misses is also expected. MRB-GC on FGHC requires free-list management for memory cells [9]; however, further enhancement of locality of memory references is expected if free-list is performed last in first out.

2.3 MRB-GC Effect on Shared Memory Multiprocessor

PIM cluster is a shared memory multiprocessor with coherent cache memory (Figure 2). A coherent cache protocol [1, 5] includes *broad-casting* and *invalidation*. When write operation occur to a shared cache block on one PE, The cache memory based on broad-casting sends a new data on common bus to update all the shared cache blocks on other cache memories. On the other hand, the cache memory based on invalidation sends a invalidate command on common bus to invalidate all the shared cache blocks on other cache memories. A coherent cache protocol based on invalidation is examined for PIM [10]. In this protocol, a missed cache block can be transferred from other cache memory, not from the shared memory, if any other cache memory has the cache block. This protocol is called *cache-cache data transfer*.

The key issue for the performance of shared memory multiprocessor is to reduce bus traffic. The bus traffic of such parallel processing systems are includes memory-cache traffic(such as *swap-in* and *swap-out*) and cache-cache traffic. If the free-list for MRB-GC is common to all PEs in the system, cache-cache transfer may occur by fetching the top of free-list in every reclamation and reuse of a memory cell. Thus, MRB-GC for FGHC parallel execution provides an independent free-list for each PE.

MRB-GC can reduce the bus traffic as follows.

Reduction of Required Storage size: Reduction of the required storage size effects reduction of swap-in operations for newly allocated memory cells. Swap-out operations are also reduced on caches with finite column number (smaller than the required storage size).

Enhancing the Locality of References: By managing free-lists last in first out, references to reused memory cells can hit the cache memory.

There are problems to cause the increase of cache-cache traffic, such as:

Invalidation: Since reclamation to a free-list generates a write operation, bus traffic for invalidation increases because of incremental GC.

Cache-Cache Data Transfer: When one cache block consists of plural memory cells, it can be shared by a number of PEs, because each cell can be reclaimed by a different PE. In that case, PEs invalidate each other's free-lists in every reclamation and reuse of memory cells. As a result, invalidation and cache-cache data transfer increases[2].

3 Evaluation Procedure

The MRB-GC effect is evaluated from the viewpoint of reduction of bus traffic. The measurement was done through the cache simulation on the memory reference information generated by the FGHC pseudo parallel emulator.

3.1 Cache Simulation

3.1.1 FGHC Pseudo Parallel Emulator

FGHC pseudo parallel emulator runs on the KL1b [8] code which is a WAM-like abstract machine code for FGHC. The FGHC emulator provides pseudo parallel execution of FGHC through one goal reduction to be a processing granule, and switching PE at the end of a goal reduction.

MRB-GC is implemented on the variable cells and cons cells area (heap area), and the data records (such as goal records) can be fully reclaimed since they can be guaranteed to be single referenced by implementation. The FGHC pseudo parallel emulator is set up using compile-time option GC-on (enables MRB-GC) and GC-off (disables MRB-GC). When GC-off is set, memory cell reclamation and memory references for reclamation are suppressed, although memory references for MRB maintenance, which are only needed for reclamation, are not suppressed.

The test program for our evaluation is the Bottom Up Parsing (BUP) program. The BUP program has 10 top goals, each of which parse an independent sentence concurrently. BUP runs about 120,000 goal reductions, about two million instruction steps and 4.4 million memory references.

[2]The effect of invalidation on the hit ratio is analyzed in [4].

3.1.2 Cache Configuration

The cache simulation is applied on two cache configurations for comparison, such as:

256 columns, 4 sets, 4 words/block: Expected to be implemented on PIM.

16K columns, 4 sets, 1 word/block: Maximum size allowed in cache simulator. The main purpose of this configuration is to analyze the effect of mutual invalidation of shared cache blocks. This configuration omits swap-out by approximating the cache of infinite size, and omits sharing cache blocks which do not hold shared data.

3.2 Measurement Items

From our evaluation of the bus cycles on parallel execution of FGHC [10], accesses to heap area constitutes the majority of bus cycles. Since MRB-GC is employed for heap area, the measurement is focused on bus cycles performed for heap area. The measurement items for this evaluation are as follows.

1. **Total Bus Cycles Performed for Heap Area:**
   Total bus cycles include memory-cache traffic and cache-cache traffic.

2. **Memory-Cache Traffic:**
   The memory-cache traffic is affected by the size of the required storage.

   Swap-in occurs at least proportionally to the required storage size, and is not affected by the number of PEs or the size of the cache memory.

   Swap-out occurs if the size of the cache is smaller than the required storage size, never occurs in caches of infinite size.

   Since the majority of memory-cache traffic is swap-in, and swap-out is usually hidden by other bus traffic, therefore, the bus cycles for swap-in are counted.

3. **Cache-Cache Traffic:**
   To determine cache-cache bus traffic, two kinds of bus traffic are measured.

   (a) **Invalidation**

   (b) **Cache-Cache Data Transfer**
   Cache-cache data transfer includes the required bus traffic for communications through shared data and the effect of mutual invalidations of shared cache blocks. The effect of mutual invalidations of shared cache block can be omitted by using cache configuration of 1 word/block.

4 Evaluation Result and Discussion

4.1 Total Bus Cycles for Heap Area

Figure 3 shows the total bus cycle count for heap area. MRB-GC reduces bus cycle count especially for a small number of PEs. This is achieved mainly by reducing memory-cache traffic such as swap-in. The effect of the MRB-GC on bus cycle reduction degrades as the number of PEs increases. The major cause of the degradation

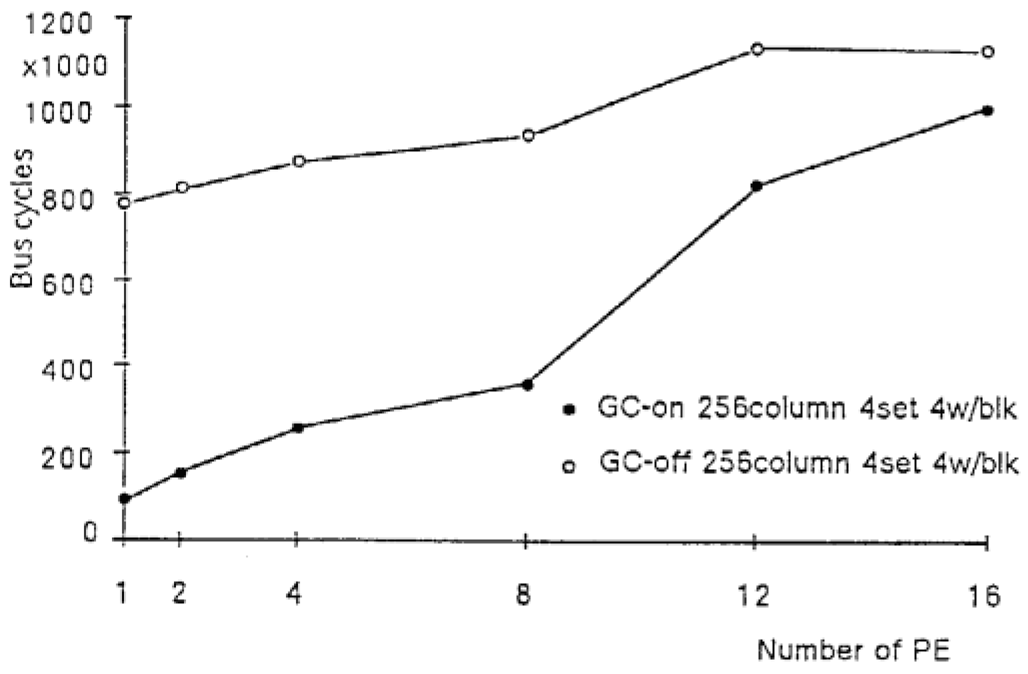

Figure 3: Bus Cycle Count for Heap Area

Table 1: The Required Storage Size

PE#	GC-on	GC-off	on/off%
1	65581	230830	28.4
2	65799	227808	28.8
4	65701	225255	29.2
8	66441	226149	29.4
12	66445	219366	30.2
16	66832	216324	31.0

is increase of cache-cache data transfer. For 16 PEs, about 3/4 of bus cycle is used by cache-cache data transfer in GC-on, while it is about 1/3 used in GC-off.

4.2 Memory-Cache Traffic

Table 1 shows the the required storage size, this result indicates that the required storage size is reduced to 30% by MRB-GC. The reduction of memory-cache traffic is expected from the result, since the reduction of the required storage size implies the reduction of swap-in operations for allocation of memory cells.

Figure 4 shows the bus cycle count for swap-in. This result indicates that the reduction of swap-in is more effective than reduction of the required storage size. While the required storage size is reduced to 1/3 by MRB-GC, swap-in is reduced to about 1/25 for one to eight PEs and to 1/4 for 12 to 16 PEs. Although the reduction of swap-in for newly allocated cells are expected only for the reduction

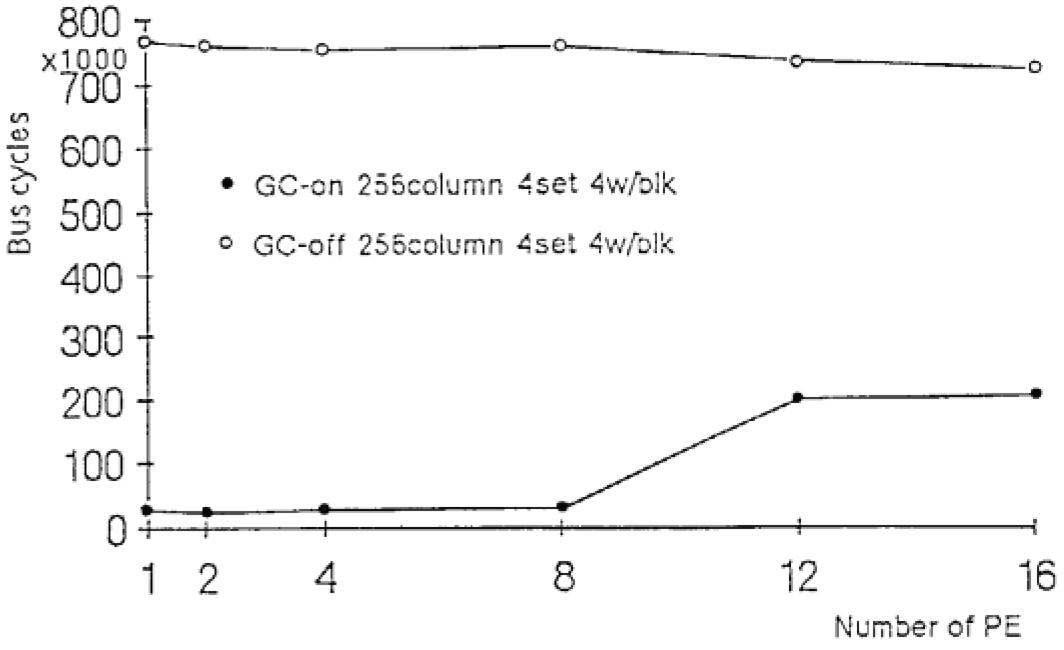

Figure 4: Bus Cycle Count for Swap-In

that of required storage size, further reduction of swap-in is accomplished. This result means that MRB-GC reduces not only swap-in but also swap-out.

The swap-in count for GC-on increases by multiples of 12 and 16 PEs, although swap-in for GC-off does not show any significant change. This result indicates that swap-out forced by other traffic (such as cache-cache traffic) increases for 12 to 16 PEs. Since the result in table 1 does not show significant difference, it is not the result of an increase in required storage size. In GC-off, the swap-out forced by swap-in occurs even with a small number of PEs.

4.3 Invalidation

Figure 5 shows the bus cycle count for invalidation. This result shows the invalidation command is increased twice by MRB-GC. It also shows that the invalidation command count is not affected by mutual invalidation of shared cache blocks, since there is no significant difference between the counts for 4 words/block and 1 word/block.

4.4 Cache-Cache Data Transfer

Figure 6 shows the cache-cache data transfer. This result shows that the cache-cache data transfer is increased twice by MRB-GC. The count of cache-cache data transfer GC-on comes close to to the count of swap-in of GC-off (figure 4) at 16 PE. The result of 1 word/block indicates that the communications through shared data increases as the number of PEs increases.

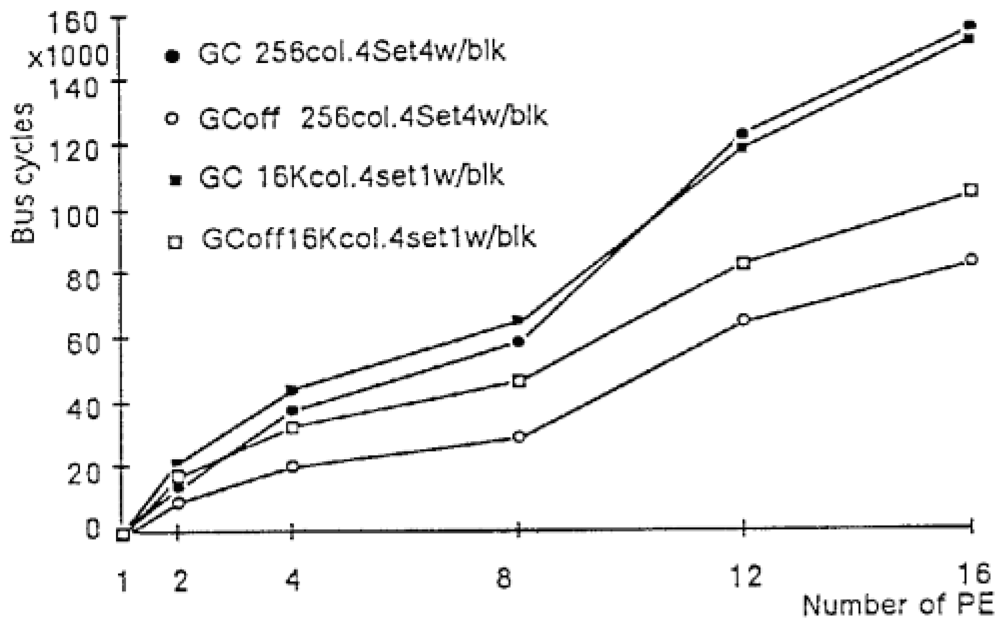

Figure 5: Bus Cycle Count for Invalidation

An increase of cache-cache data transfer in 4 words/block is considered a result of mutual invalidation of a cache block. In GC-off, the effect of mutual invalidation is comparatively lower than in GC-on, since number of the live cells in one cache block is considered to be fewer than that of GC-on. In addition, mutual invalidation of free-list greatly increases the cache-cache data transfer, because it is very frequently updated by GC. Cache-cache data transfer uses more than three quarters of the bus cycles in GC-on for 2 to 16 PEs.

5 Conclusion

This paper evaluated the effect on the reduction of the number of bus cycles by MRB-GC. This evaluation shows the importance of incremental GC for FGHC by reducing the required storage size. Reduction of the required storage size not only reduces global GC frequency but also reduces of swap-in operations. MRB-GC reduces memory-cache traffic such as swap-in and swap-out. It is effective for any number of PEs, yet the increase of cache-cache traffic reduces the effect somewhat for large number PEs.

Cache-cache traffic, especially cache-cache data transfers is increased by MRB-GC, which is mainly caused by mutual invalidation of shared cache blocks. The problem of mutual invalidation of shared cache blocks becomes a major problem in parallel execution of incremental GC, since incremental GC tends to increase the number of shared memory cells in a cache block, by increasing the number of live cells in one cache block.

Summarizing the effect of MRB-GC on the number of bus cycles:

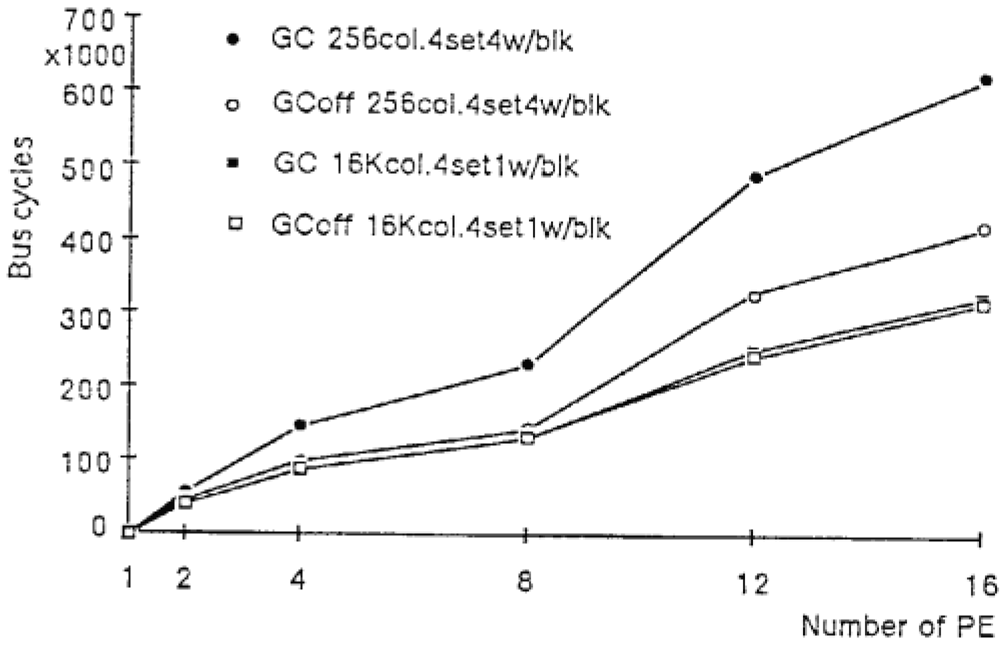

Figure 6: Cache-Cache Data Transfer

- MRB-GC reduces the memory-cache traffic by reducing the required storage size.
- Increase of the cache-cache data transfer may seriously affect the benefits of MRB-GC in bus cycle reduction. It is mainly caused by an increase of mutual invalidation of shared cache blocks.

Although the reduction of bus cycles degrade as the number of PE increases, it is still lower than that of GC-off. Thus the total system performance can be improved by MRB-GC by reducing the frequency of global GC.

Acknowledgment

I wish to thank the research members of the PIM and Multi-PSI project, especially Dr. T. Chikayama, Mr. K. Nakajima, Mr. N. Miyauchi, and Mr. M. Sato, for their useful comments. I also wish to thank the ICOT Director, Dr. K. Fuchi, and the chief of the fourth research section, Dr. S. Uchida, for their valuable suggestions and guidance.

References

[1] J. Archibald and J. Baer. Cache Coherence Protocols: Evaluation Using A Multiprocessor Simulation Model. *ACM Transaction of Computer Systems*, 4(4):273–298, 1986.

[2] T. Chikayama and Y. Kimura. Multiple Reference Management in Flat GHC. In *Proceedings of the Fourth International Conference on Logic Programming*, pages 276–293, 1987. (also in ICOT Technical Report, TR-248).

[3] K. Clark and S. Gregory. Notes on Systems Programming in PARLOG. In *Proc. of the International Conference on Fifth Generation Computer Systems*, pages 299–306, Tokyo, 1984.

[4] Michel Dubois. EFFECT OF INVALIDATIONS ON THE HIT RATIO OF CACHE-BASED MULTIPROCESSORS. In *Proc. of the 1987 International Conference on Parallel Processing*, pages 255–257, August 1987.

[5] J. R. Goodman. Using Cache Memory to Reduce Processor-Memory Traffic. In *Proc. of the 10th Annual International Symposium on Computer Architecture*, pages 124–131, 1983.

[6] A. Goto. Parallel Inference Machine Research in FGCS Project. In *US-Japan AI Symposium 87*, pages 21–36, Nov. 1987.

[7] A. Goto and S. Uchida. Toward a High Performance Parallel Inference Machine –The Intermediate Stage Plan of PIM–. TR 201, ICOT, 1986. (also in Future Parallel Computers, LNCS 272, 299–320, Springer-Verlag).

[8] Y. Kimura and T. Chikayama. An Abstract KL1 Machine and its Instruction Set. In *Proceedings of the 1987 Symposium on Logic Programming*, pages 468–477, 1987. (also in ICOT Technical Report TR-246).

[9] Y. Kimura, K. Nishida, N. Miyauchi, and T. Chikayama. Realtime GC by Multiple Reference Bit in KL1. In *Proceedings of the Data Flow Workshop 1987*, pages 215–222, Oct. 1987. (in Japanese).

[10] A. Matsumoto et al. Locally Parallel Cache Designed Based on KL1 Memory Access Characterestics. TR 327, ICOT, 1987.

[11] N. Miyauchi, Y. Kimura, T. Chikayama, and K. Kumon. Multiple Reference Management by MRB –GC Characteristics on KL1 Emulator–. In *35th Meeting of Information Processing Society*, Sept. 1987. (in Japanese).

[12] K. Nishida, A. Matsumoto, Y. Kimura, and A. Goto. Multiple Reference Management by MRB –Cache Characteristics on KL1 Emulator–. In *35th Meeting of Information Processing Society*, Sept. 1987. (in Japanese).

[13] E.Y. Shapiro. A subset of Concurrent Prolog and Its Iterpreter. TR 003, ICOT, 1983.

[14] K. Taki, K. Nakajima, H. Nakashima, and M. Ikeda. Performance and architectual evaluation of the PSI machine. In *Proceedings of the Second International Conference on Architectual Support for Programming Language and Operating Systems (ASPLOS)*, pages 128–135, 1987.

[15] K. Ueda. Guarded Horn Clauses: A Parallel Logic Programming Language with the concept of a Guard. TR 208, ICOT, 1986. (also to appear in Programming of Future Generation Computers, North-Holland, Amsterdam. 1987.).

[16] K. Ueda. Introduction to Guarded Horn Clauses. TR 209, ICOT, 1986.