ICOT Technical Report: TR-469

TR-469

Tracing Garbage Collection for KL1 on the Multi-PSI/V2 System

by
N. Miyauchi, Y. Kawada (Mitsubishi)
& K. Nakajima

March, 1989

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191 – 5
Telex ICOT J32964

Institute for New Generation Computer Technology

Tracing Garbage Collection for KL1 on the Multi-PSI/V2 System

Nobuhito Miyauchi† Yasuharu Kawada†† Katsuto Nakajima‡

†Mitsubishi Electric Corp. ††SET Corp. ‡ICOT Research Center

Abstract

This paper describes the implementation and evaluation of tracing garbage collection for KL1 on the Multi-PSI/V2. It is important for committed choice language systems to implement effective memory management mechanisms, because the memory consumption speed is quite high. The incremental garbage collector by Multiple Reference Bit (MRB) reclaims single-referenced data in order to keep locality and avoid frequent tracing garbage collection on the Multi-PSI/V2, but cannot collect multiple-referenced garbages. Thus, the tracing garbage collector is indispensable to collect multiple-referenced garbages. The MRB information is maintained by counting reference paths in the tracing garbage collection, so the incremental garbage collector can collect all of single-referenced garbages after that. We evaluated the performance of the tracing garbage collection and the statistics of memory consumption on the Multi-PSI/V2, and confirmed that all of processors can work away without the disturbance of the tracing garbage collection.

1 Introduction

The Multi-PSI/V2 was developed as a pilot model of a parallel inference machine in the Japanese fifth generation computer systems (FGCS) project. The parallel language programming system to execute KL1 is implemented on it. We are now evaluating the performance of the KL1 system and making research on the parallel programming systems, operating systems, and algorithms.

The Multi-PSI/V2 is a non-shared memory multi-processor constructed from up to 64 processors connected by a message passing network. Various application programs and the parallel inference machine operating system, PIMOS, has been developed on it.

As the memory cells are rapidly consumed in the execution of committed choice languages such as KL1, a naive implementation would cause serious performance degradation by the less locality of the memory references and frequent invocations of the tracing garbage collector. To avoid the performance degradation, the Multi-PSI/V2 has currently three types of the garbage collector: an incremental intra-PE garbage collector by Multiple Reference Bit (MRB) [2, 7], a tracing intra-PE garbage collector, and an incremental inter-PE garbage collector [5]. The incremental garbage collector by MRB can collect many garbages referenced by only one pointer. However, it cannot collect them if they are once referenced by two or more pointers. Therefore, the implementation of the tracing garbage collection is indispensable to collect all garbages. We have implemented a tracing garbage collector, using the copying algorithm, which also maintains the MRB in tracing memory. This technique considerably increases the chances to incrementally collect garbages.

This paper reviews the incremental garbage collection by MRB in section 2, and describes the algorithm of the tracing garbage collection in section 3, its implementation on the Multi-PSI/V2 in section 4, and the evaluation of the performance of the tracing garbage collection in section 5.

2 Incremental Garbage Collection by MRB

In the execution of a committed choice language such as KL1 which has no backtracking feature, memory consumption ratio is pretty high as the result of list cells being consumed for process communication by streams. As memory cells are consumed rapidly, tracing garbage collection must be executed much frequently, and the performance shall go down seriously. On the other hand, the statistics of KL1 programs show that most of the data are single-referenced.

So, an incremental garbage collection mechanism for single-referenced garbages is introduced, in order to keep the locality of the memory reference and decrease the frequency of the tracing garbage collection. In this mechanism, the MRB, one additional bit for a pointer, is used to indicate whether the object must be referenced by only one pointer (*off-MRB*) or may be referenced by multiple pointers (*on-MRB*), as shown in Figure 1 and 2 [2].

A single-referenced object become the garbage and is collected, when the path to it is consumed by the dereference, unification, and so on. The memory cell for the collected object is chained to a free list for the reclamation.

Special instructions are introduced to collect garbages and maintain the MRB. KL1

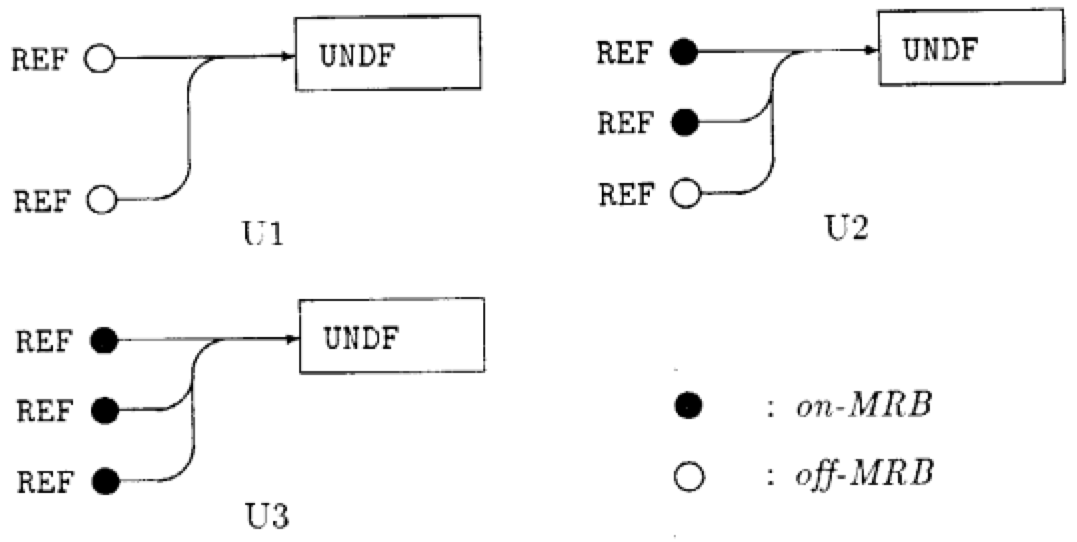

Figure 1: MRB of References to an Uninstantiated Variable

compiler generates instruction codes by analyzing reference information and timing of collecting garbages.

3 Method of Tracing Garbage Collection

3.1 Necessity of Tracing Garbage Collection

Once MRB of a pointer is turned on to indicate multiple-referenced (*on-MRB*), the MRB cannot be turned off even if the pointers decrease and their object becomes to be referenced by the single pointer.

It is the reason that the increment and decrement of reference paths cannot be managed by 1 bit reference information. Consequently the tracing garbage collector must collect garbages which have ever been multiple-referenced.

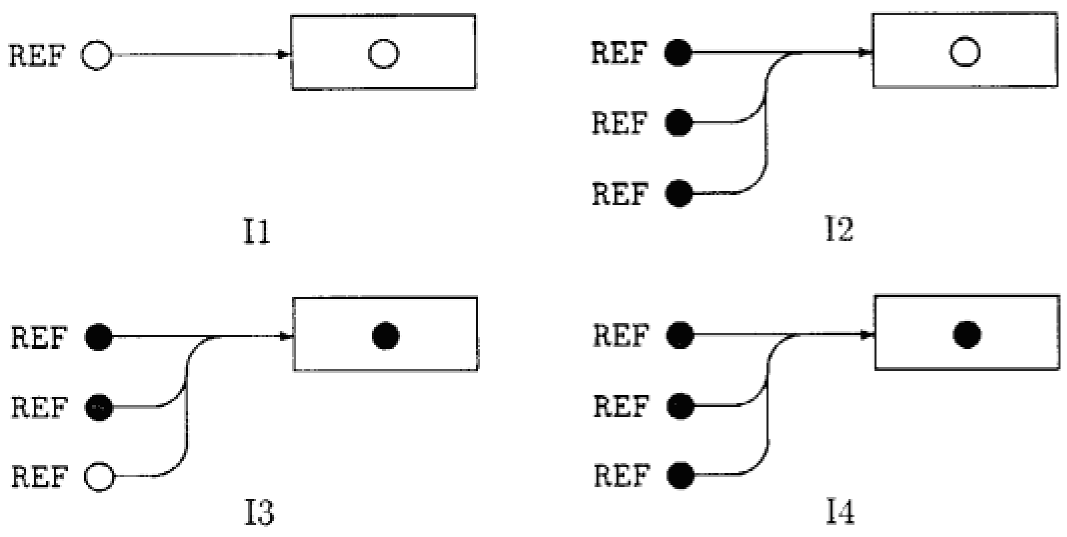

Figure 2: MRB of References to an Instantiated Variable

Moreover, the tracing garbage collection is indispensable to reclaim circular structures which cannot be collected by any reference counting, and to resolve the fragmentation of the free lists.

3.2 Copying Algorithm

Copying algorithm adopted in our garbage collector is faster than marking and compacting algorithm. That is, it sweeps only active data, while marking and compacting algorithm sweeps whole memory area. This advantage should overcome the drawback of the available memory size for switching of accessible memory area, a half of the actual memory, especially in the systems which have large scale main memory, such as the Multi-PSI/V2.

In this algorithm, data cells are allocated to a half of the heap area, the active heap, until it is used up and the tracing garbage collector are invoked. The tracing garbage collector only traces active data from marking roots, and copies them to another half of the heap area, which will be the next active heap.

Generally one bit of a memory cell is used as a GC bit to indicate whether it has been already copied or not.

3.3 Dereference and Maintenance of MRB in Copying

A naive garbage collector will copy all active cells, including invisible pointers, without any modification. In a committed choice language system, overwriting of the dereference result is allowed for the garbage collector. That is, it reduces the size of active cells by eliminating intermediate invisible pointers and the cost of copying their cells. Moreover, the reduction of the reference chain length should improve the performance of the dereference. Almost of reference pointer cells can be collected by dereference of tracing garbage collection.

As to the maintenance of MRB, if a pointer with *on-MRB* is found in the reference chain, a naive method will turn on the MRB of the root of the chain to concern with possibility of multiple-reference. Although this method is simple, it will turned on the MRB of the single-reference pointers whose objects once were referenced by multiple pointers.

In our tracing garbage collector, the MRB of those pointers are turned off by the following method. When an object is found for the first time, it and its pointer are copied to the new area, and the MRB of the pointer is cleared, regardless of the MRB of the pointers in the reference chain. So, if there are not other pointers referencing the object, the MRB of the pointer is off indicating that the object is referenced by the single pointer.

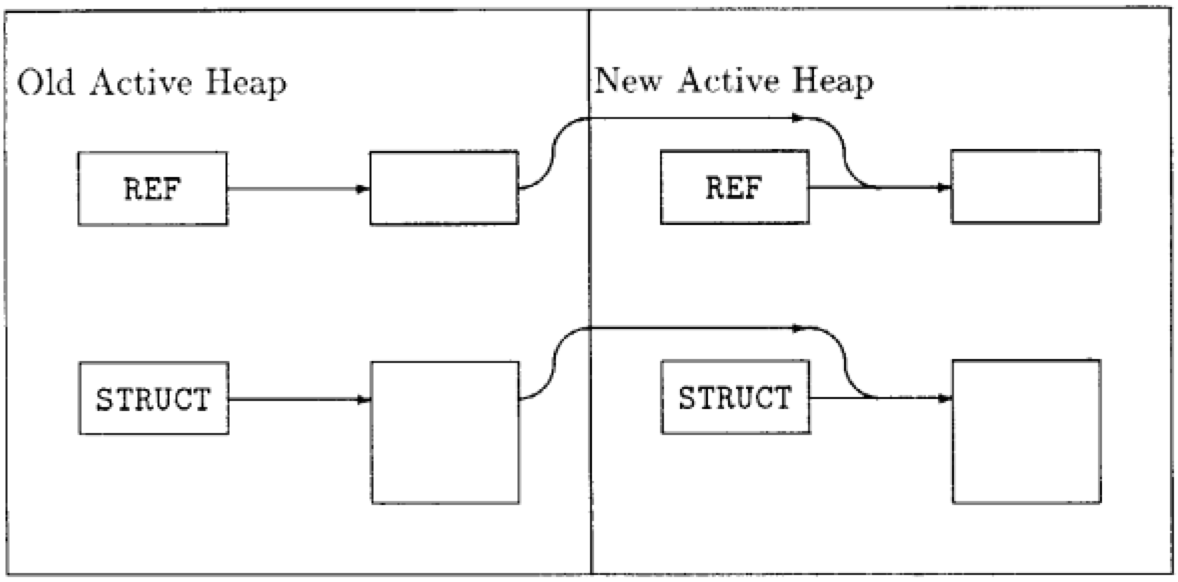

Figure 3: Maintenance of Pointers on Normal Copying GC

The MRBs of those pointers are turned off by the following method. When an unmarked object is found in the old active heap as the result of dereference, the object and a pointer to it are copied to the new active heap. The MRB of the pointer is made off, regardless of the MRBs of the pointers in the reference chain in the old heap. So, if there are not other pointers to the object, the MRB of the pointer is *off-MRB* to indicate that the pointer references to the object.

If there are other pointers, that is, marking process finds the object again, the MRB of all the pointers including the first one should be turned on. In order to turn on the MRB of the first pointer, the object in the old area is replaced with the pointer to the first pointer in the new area, as shown in Figure 4, instead of the pointer to the object in the conventional method as shown in Figure 3.

The MRB of each type cells are maintained as follows.

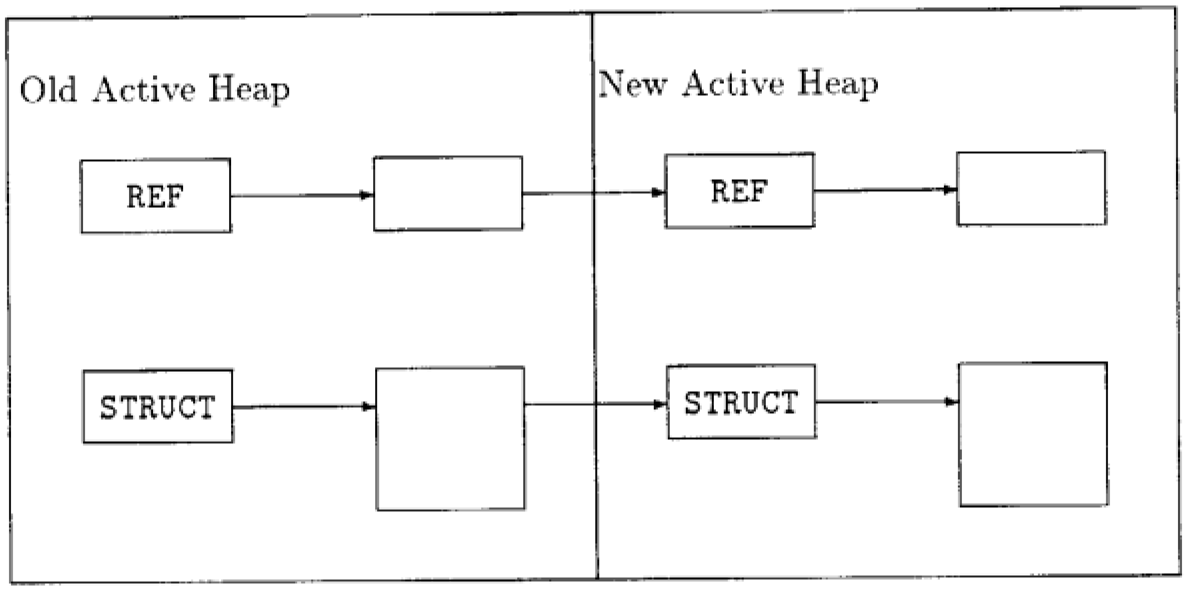

Figure 4: Maintenance of Pointers on MRB Maintenance Copying GC

Pointers to Atomic Cells The pointers to atomic data cells, such as atoms and integers, are removed, because those data are directly written in the location of the pointer in new active heap. In consequence, maintenance for their MRB is not required.

Pointers to Structure Data Cells The MRBs of pointers to structure data cells are turned off in case of single-reference and turned on in case of multiple-reference according to the above method. The invisible pointers to the pointer cells are removed as those to atomic data.

Pointers to Uninstantiated Data Cells As shown in Figure 1, the MRB of the pointers to unbound variables are turned off in the case of single- and double-reference, and turned on in the case of multiple-reference more than two. Note that it is allowed to turn off one pointer's MRB in latter case [2].

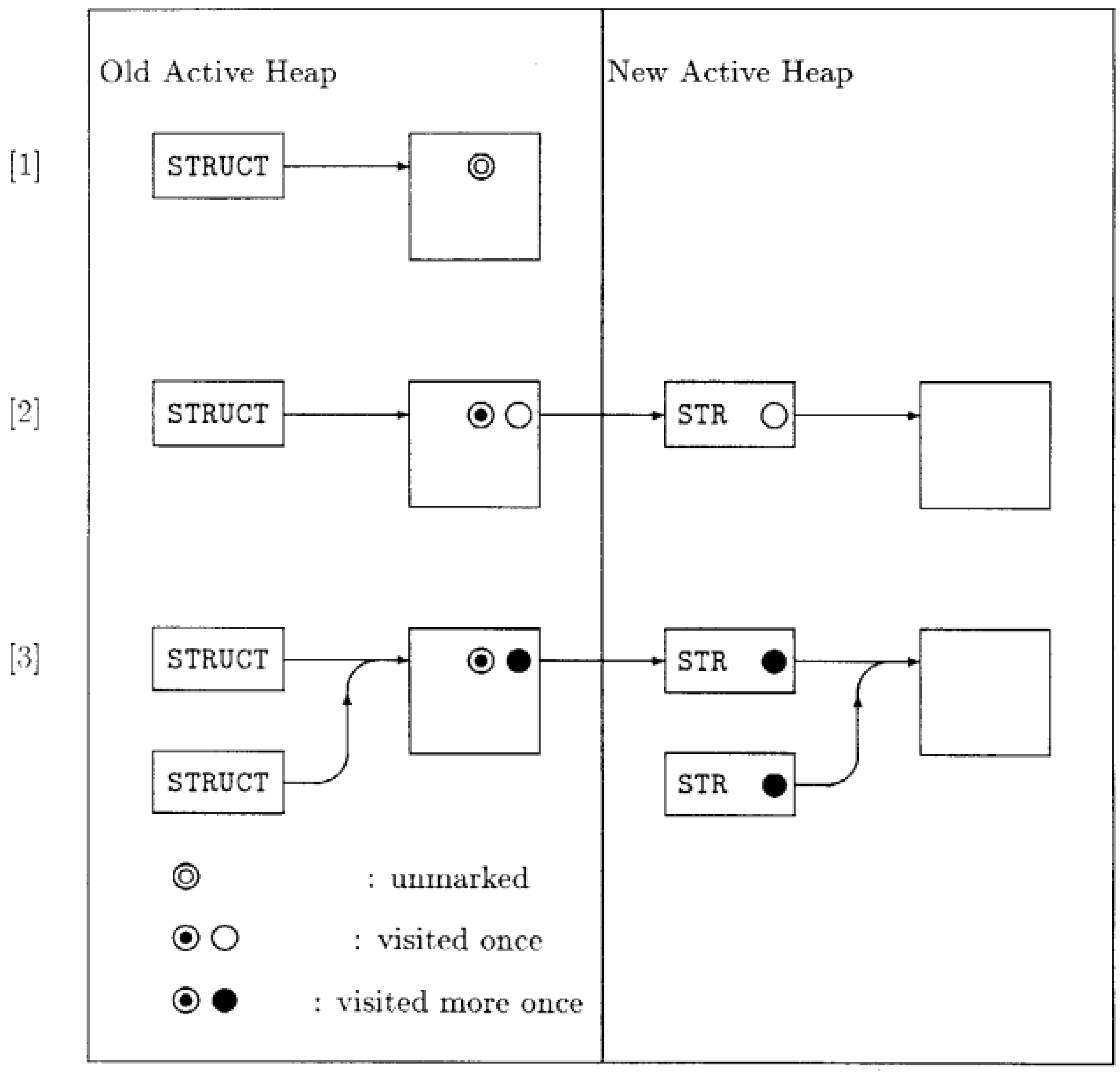

Figure 5: MRB Maintenance of Pointers to Structure Data Cells

Thus, the first and second pointers are copied as *off-MRB* pointers, and the MRB of the first pointer is turned on when the third pointer is copied.

To identify the four different states, unmarked, marked once, twice and more than twice, the GC bits and MRB of the object in old area are used.

4 Implementation of Tracing Garbage Collection on the Multi-PSI/V2

In the implementation on the Multi-PSI/V2, active goals are chained to a list, called goal stack. Thus, the roots of the marking are the arguments of the goals in the goal stack. There are various data types, other than described above, such as suspended goals and variables which hook them, special data for a constant time merger, pointers to data in other processors, and instruction codes.

Export and import data tables are implemented for the inter-PE garbage collection [5]. Each entry of the export table, corresponding the data referenced by other processors, is one of the root of the marking. Each entry of the import table, corresponding the reference to the other processor, is used to request counting down the reference count, when it becomes the garbage by the intra-PE garbage collector. The count down request will remove the export table entry when the reference count becomes zero, and the data corresponding to the entry may be collected by the intra-PE garbage collector.

There are several free lists according to the size of cells. When a new data is created and the corresponding free list is empty, the list is extended by consuming the heap area. If the size of the extended list is too small because of the short of the available heap area,

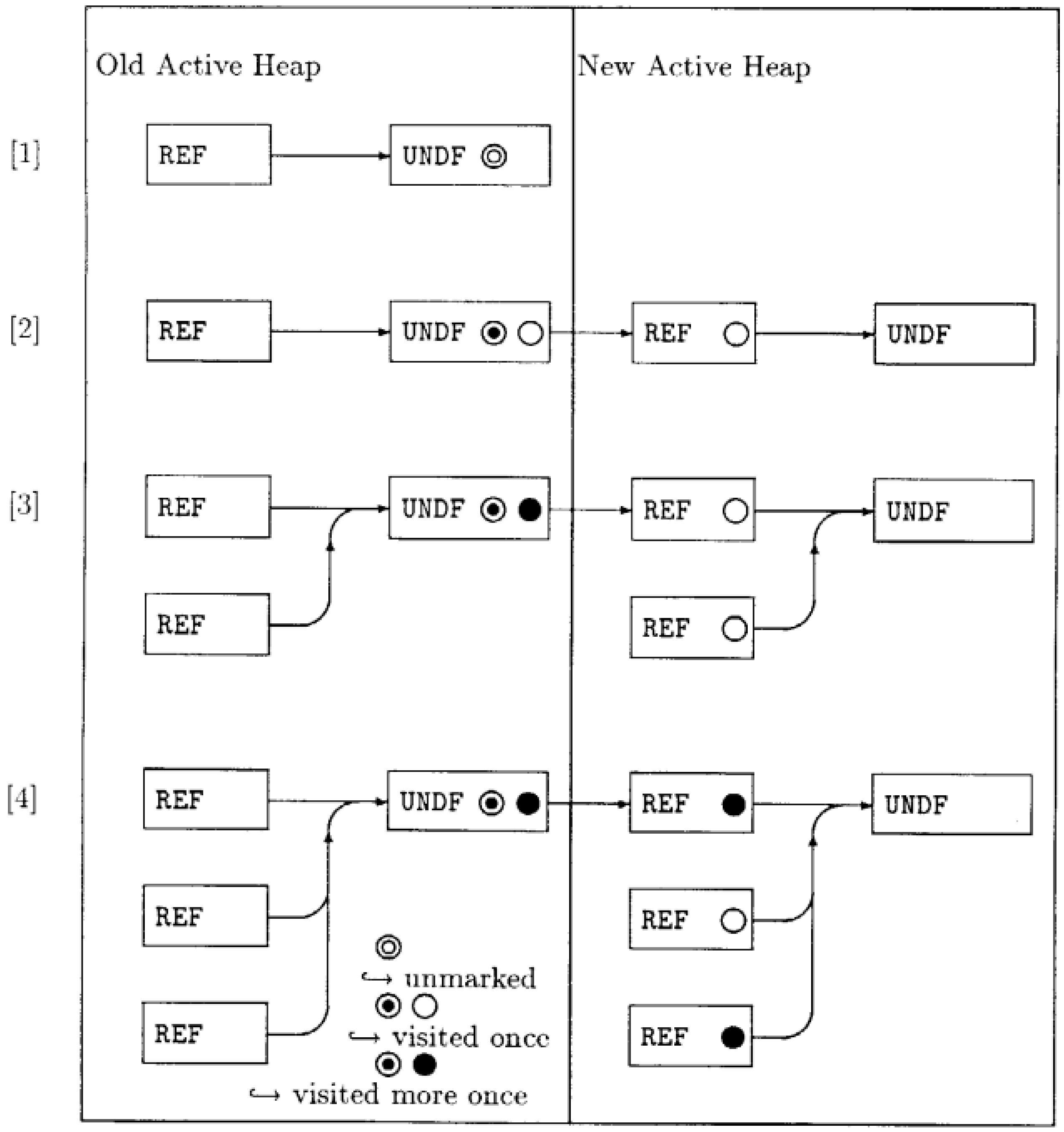

Figure 6: MRB Maintenance of Pointers to Undefined Data Cells

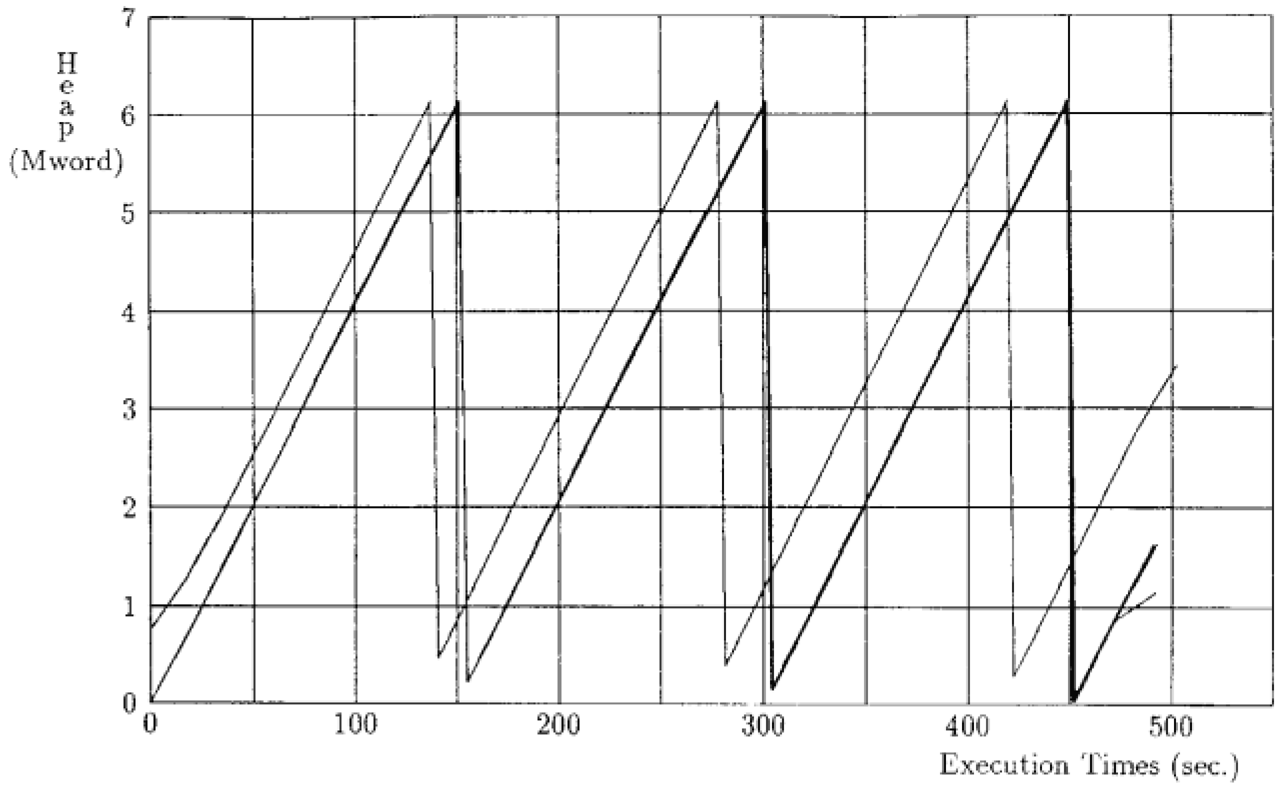

Figure7 (a) : History of the Heap Consumption (**Pentomino**)

the garbage collector is invoked.

5 Evaluation on the Multi-PSI/V2

We evaluated consumption speed by measuring the interval time of the invocation of the tracing garbage collector, the amount of active data after the garbage collection, and the execution time of the garbage collector on the 4-processor Multi-PSI/V2. The statistics of four processors are shown in Figure 7. Initially, the Code data are storaged in the processor whose number is zero.

We used the following five benchmark programs, one is simple and others are considerably large, for the evaluation. They are executed under the control of the PIMOS.

Queens :It a simple benchmark which computes all solutions to the 13-queens problem. The evaluation on the sequential KL1 emulator shows that the most of active data cells are invisible pointer cells. The memory consumption ratio of all the non-idle processors is constant.

Pentomino : It computes all solutions to the 8×5 packing piece puzzle by exhaustive search. The memory consumption ratio of all the non-idle processors is constant.

Tsumego : It solves sub-problems of the Japanese board game, "Go", similar to the checkmate problems of the chess. The scheme of dynamical load balancing is implemented by software and one processor is the manager of load balancing which was found to be not always busy. One of difficult problems is solved three times continuously in this measurement.

PAX : It is a natural language processing subsystem. It analyzes natural language sentences and makes the parse tree. This program is based on the bottom-up parser using the layered stream method. Forty same sentences each of which is composed of 85 words are parsed in this measurement. There are many active data in the execution. Memory consumption ratio frequently changes because the circumstances of the execution is complicated for matching various grammer.

Best Path:It finds the minimum cost paths for all the network nodes on a network with about ten thousand nodes. Each network branch has a non-negative cost. About ten thousand nodes processes of the network are allocated as 4 nodes groups of squares divided by 4 to 4 processors statically. The farther node area of each processor from a start point is, the longer total execution time of it is.

Main characteristics about the measurement are as follows: memory consumption

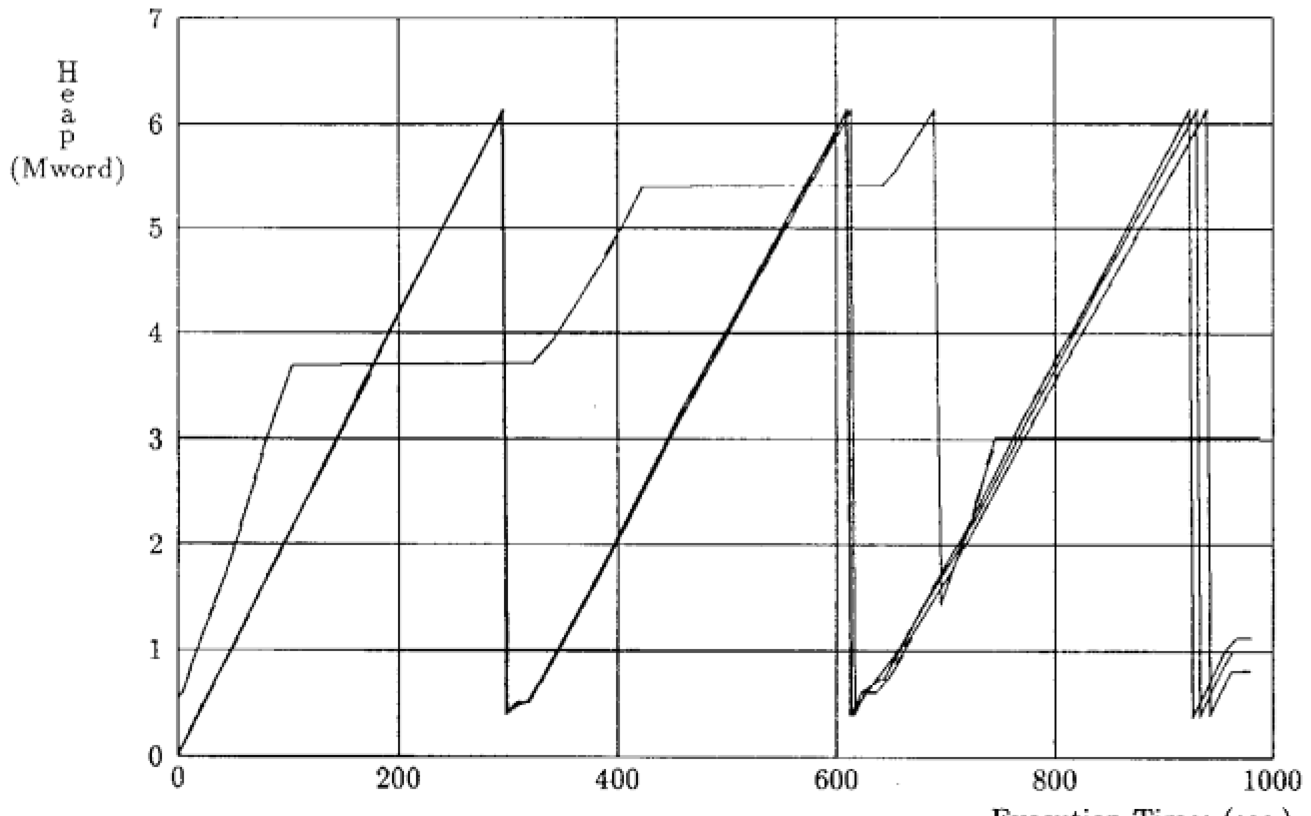

Figure7 (b) : History of the Heap Consumption (**Tsumego**)

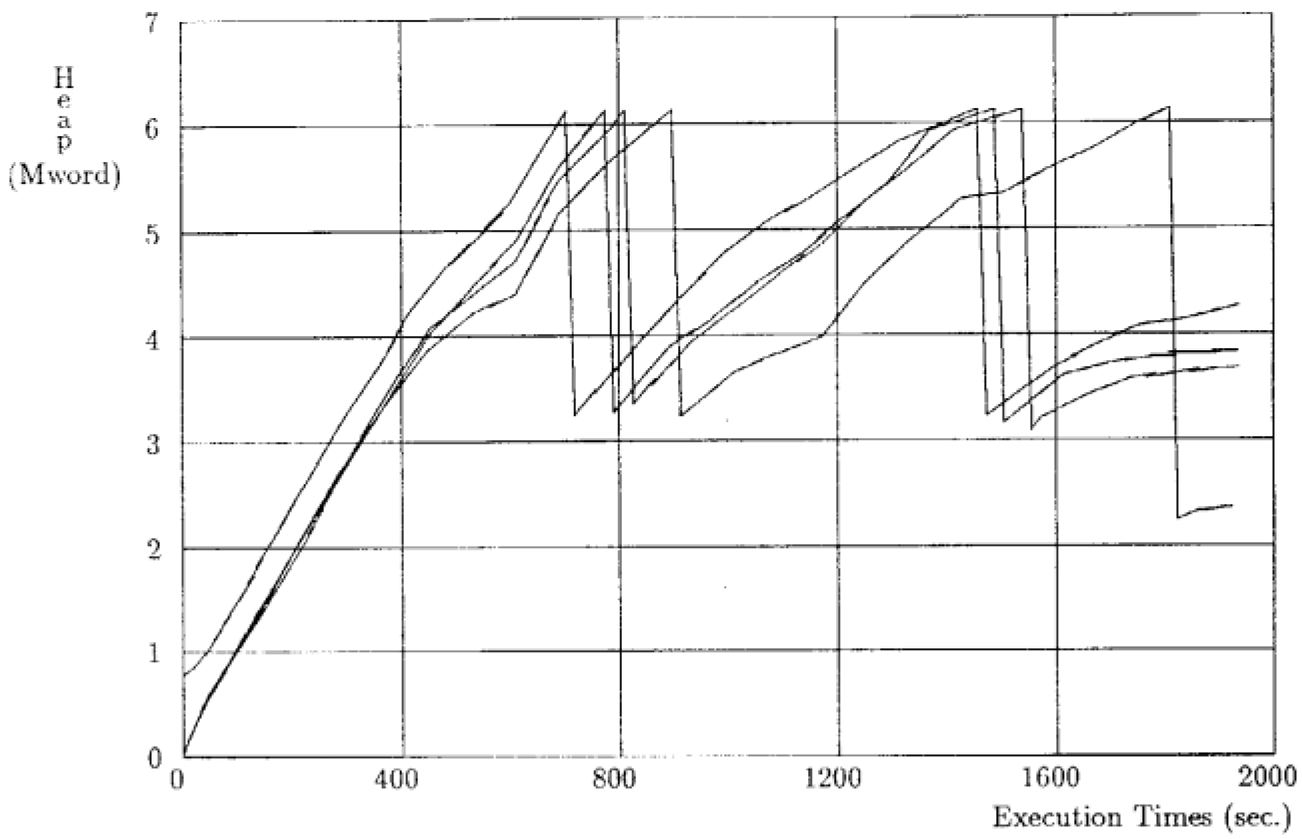

Figure7 (c) : History of the Heap Consumption (**PAX**)

ratio is almost constant according to a program, especially a type of exhaustive search programs like **Queens** and **Pentomino**; the more active data there are in heap area, the longer the execution time of tracing garbage collection.

We measured memory consumption with the lapse of time to execute in case that the maximum size of heap area is smaller than the size of actual heap area, too. The result of **Pentomino** is given in Figure 8.

The tracing garbage collector doesn't reduce the performance of KL1 execution as much as available memory is reduced, as illustrated by the data in Table 1. 'GC' means the frequency of tracing garbage collection, and 'Active' means the average of active data over all tracing garbage collections. The overhead of tracing garbage collection is about

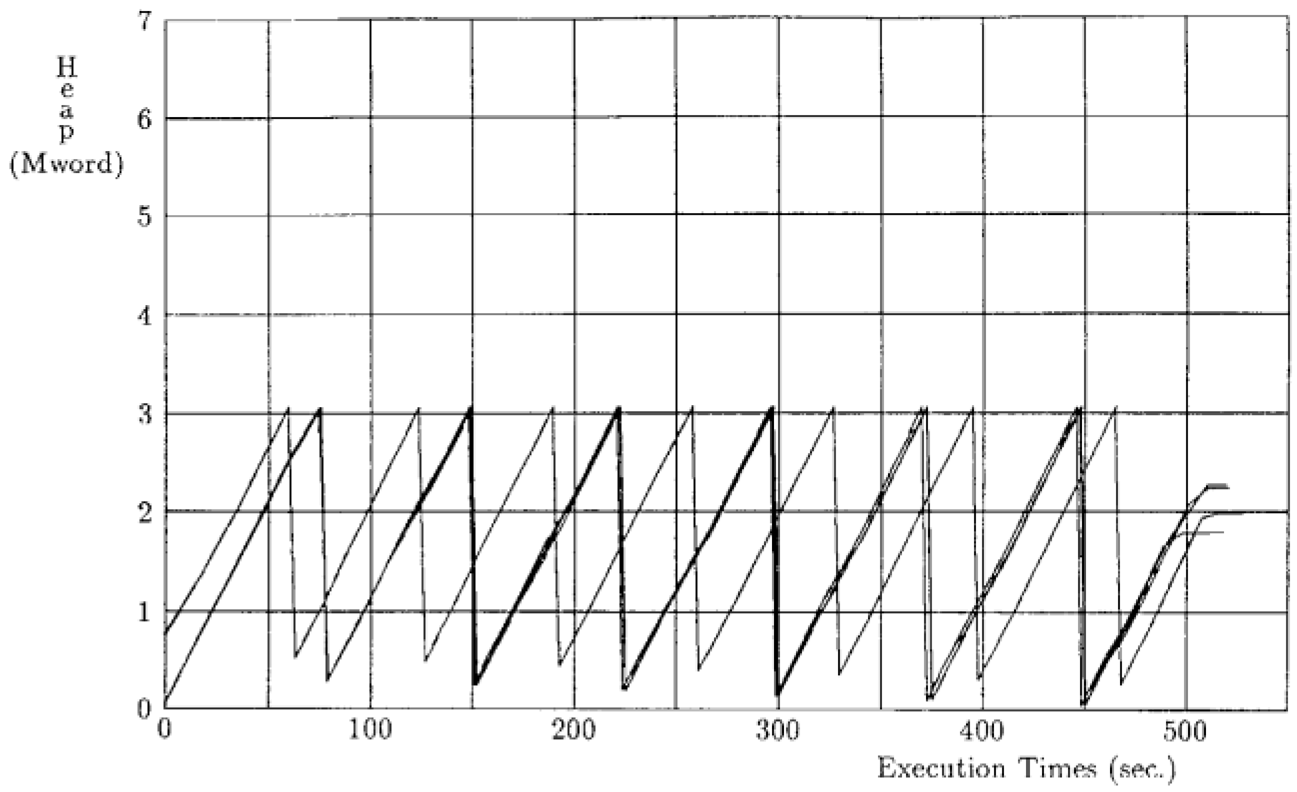

Figure8 : History of the Heap Consumption (**Pentomino**; Heap:$\frac{1}{2}$)

1% ~ 2% in case of 4 processors with 12 Mword of heap memory. (Physical memory ⇒ 16 Mword)

We also measured the overhead by varying the number of processors which work, but couldn't discern the difference of various cases. This shows there are little overhead of the procedure to maintain the information of external reference between processors in the tracing garbage collection.

On the other hand, the overhead is estimated to be large in case of such an application program that the memory consumption ratio is high like **Pentomino** and there are many active data like **PAX**.

6 Conclusion and Future Work

As a result of this measurement we concluded that tracing garbage collector is not invoked so frequently that total performance does not go down by it and that incremental garbage collector by MRB is very effective for lower memory consumption ratio and locality. It needs to evaluate the execution of more application programs and the effect of MRB maintenance in the tracing garbage collection.

We must evaluate the performance of memory management totally in comparison with the multi-processor system for KL1 supported only the tracing garbage collector without the incremental garbage collector by MRB. The incremental and tracing garbage collectors by MRB scheme is expected to be more efficient to the performance of the Parallel Inference Machine (PIM) systems as shared memory multi-processor systems, which are being developed in the Japanese FGCS project [15].

Acknowledgments

We would like to thank Dr. Shunichi Uchida, chief of the ICOT fourth laboratory, and the research members of the Multi-PSI and PIM project for their valuable suggestions, who have worked with us in designing and implementing the KL1 system on the Multi-PSI/V2.

Table1: Tracing Garbage Collection Time Overhead

Programs	Heap [Mword]	PE	Run Time [second]	GC	GC Time [second]	Active [Mword]	Overhead
Queens	6.13	0	1104	4	11.3	0.234	1.02%
	(1)	1	1095	5	12.4	0.00347	1.13%
		2	1138	6	14.9	0.00345	1.31%
		3	1154	5	12.5	0.0203	1.08%
	3.07	0	1086	8	22.6	0.234	2.08%
	$(\frac{1}{2})$	1	1103	11	27.4	0.00352	2.48%
		2	1200	12	29.9	0.00360	2.49%
		3	1115	11	27.4	0.00322	2.45%
Pentomino	6.13	0	502	3	9.56	0.386	1.90%
	(1)	1	492	3	10.28	0.139	2.09%
		2	491	3	8.43	0.144	1.72%
		3	493	3	9.52	0.140	1.93%
	3.07	0	528	7	22.2	0.397	4.20%
	$(\frac{1}{2})$	1	518	6	18.5	0.176	3.57%
		2	511	6	17.5	0.174	3.43%
		3	510	6	17.1	0.175	3.35%
Tsumego	6.13	0	745	1	5.72	1.45	0.77%
	(1)	1	966	3	10.29	0.389	1.07%
		2	963	3	10.31	0.391	1.07%
		3	963	3	10.33	0.394	1.07%
	3.07	0	786	4	25.6	1.78	3.26%
	$(\frac{1}{2})$	1	1008	6	18.8	0.275	1.86%
		2	1008	6	18.7	0.269	1.86%
		3	1008	6	18.7	0.267	1.86%
PAX	6.13	0	1923	2	29.6	3.25	1.54%
	(1)	1	1935	2	30.6	3.23	1.58%
		2	1935	2	30.4	3.23	1.57%
		3	1935	2	27.7	2.74	1.43%
	4.60	0	1976	5	71.7	3.44	3.63%
	$(\frac{3}{4})$	1	1976	4	57.4	3.25	2.91%
		2	1956	5	75.6	3.19	3.86%
		3	1976	3	41.8	3.04	2.12%
Best Path	6.13	0	463	0	—	—	—%
	(1)	1	1154	2	23.9	2.29	2.07%
		2	145	0	—	—	—%
		3	541	0	—	—	—%
	4.60	0	802	1	7.23	1.20	0.90%
	$(\frac{3}{4})$	1	1495	3	28.97	1.78	1.94%
		2	233	0	—	—	—%
		3	553	1	9.32	1.52	1.69%

References

[1] K.A.M. Ali and S. Haridi. Global Garbage Collection for Distributed Heap Storage Systems. *International Journal of Parallel Programming*, 15(5):339–387, Oct. 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.

[3] J. Cohen. Garbage Collection of Linked Data Structures. *ACM Computing Surveys*, 13(3):341–367, Sept. 1981.

[4] L.P. Deutsch and D.G. Bobrow. An Efficient, Incremental, Automatic Garbage Collector. *CACM*, 19(9):522–526, Sept. 1976.

[5] N. Ichiyoshi, K. Rokusawa, K. Nakajima and Y. Inamura. A New External Reference Management and Distributed Unification for KL1. In *Proceedings of the FGCS'88*, pages 904–913, 1988. (Also appeared in TR 390, ICOT, 1988.)

[6] 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.

[7] 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).

[8] E.Y. Shapiro. A subset of Concurrent Prolog and its Interpreter. TR 003, ICOT, 1983.

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

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

[11] D. I. Bevan. Distributed garbage collection using reference counting. In *Proceedings of Parallel Architectures and Languages Europe*, pages 176–187, June 1987.

[12] P. Watson and I. Watson. An Efficient Garbage Collection Scheme for Parallel Computer Architecture. In *Proceedings of Parallel Architectures and Languages Europe*, pages 432–443, June 1987.

[13] A. Goto, Y. Kimura, T. Nakagawa, and T.Chikayama. Lazy Reference Counting – An Incremental Garbage Collection Method for Parallel Inference Machines. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 1241–1256, 1988.

[14] K. Nakajima. Efficient Garbage Collection for AI Languages. In *Proceedings of the IFIP WG 10.3 Working Conference on Parallel Processing*, 1988. (Also appeared in Technical Report TR-354, ICOT, 1988.)

[15] A. Goto, M. Sato, K. Nakajima, K. Taki, and A. Matsumoto. Overview of the Parallel Inference Machine Architecture (PIM). In *Proceedings of the FGCS'88*, pages 208–229, 1988.

[16] K. Nakajima, Y. Inamura, N. Ichiyoshi, K. Rokusawa, and T. Chikayama. Distributed Implementation of KL1 on the Multi-PSI/V2. To appeared in *Proceedings of the ICLP'89.*