数理解析研究所講究録
第618巻 1987年 97-109

On the Working Set Concept for Data-flow Machines: Consideration on List Processing on a Hierarchical Structured Memory

永松 竜夫　　砂原 秀樹　　所 真理雄
Tatsuo Nagamatsu, Hideki Sunahara and Mario Tokoro

Department of Electrical Engineering
Keio University
Yokohama 223 JAPAN

Abstract

In previous papers[9,7,8], the working set concept of instructions for data-flow machines was introduced and evaluated.

This paper discusses a method of list processing on data-flow machines based on the working set concept. A hierarchical structured memory and several policies to manage the working set of structured data are proposed and evaluated.

1 Introduction

In recent years, non-numerical processing such as in AI applications has become important. Non-numerical processing includes operations on structured data such as lists and trees. Thus, computer systems which can execute the operations of structured data at a high speed are necessary.

Data-flow architecture is one of the solutions for the requirements of high speed computation. Various data-flow architectures have been proposed[10]. Some data-flow machines aim at high speed list processing[3,4,11]. Data-flow architectures have advantages over conventional and other parallel computing architectures in that they have capabilities for automatic detection and utilization of maximum parallelism in a program at execution time. However, there are some disadvantages in execution with a finite number of resources (e.g., execution units and memories). Thus, schemes to utilize maximaly the parallelism that is provided by resources are required.

As an answer to this requirement, the working set concept for data-flow machines proposing a data-flow architecture with hierarchical instruction memories has been introduced[9,7,8]. In data-flow programs which do not include branches, loops, and function calls, **the principle of locality** does not apply as it does in conventional machines because in such machines each instruction is executed only once. Instead, a working set concept based on the **simultaneity of execution** of a program can be established.

The execution of an instruction returns the result value with its id or a list of destinations. However, when structured data is required, it is not possible to return values as we can in case of scalars; values are too large to be sent. Thus, methods are proposed in which the execution of an instruction returns only the result id; the result value is stored in the memory for structured data called the structured memory[2,1,5]. When the structured memory becomes large, it is economical to provide a two-level structured memory. Thus, the working set for data should be considered.

In this paper, several policies to manage the working set of structured data are proposed. Evaluation is made of these policies.

2 The Working Set Model for Structured Data

To execute the operation of structured data efficiently, two kinds of the structured memory have been proposed. One method is called the **reference count** method[2,1]. In this method, the structured data are stored in terms of a binary tree since it can express any form of structured data and can modify data dynamically. The other method is called the **I-structure**[5]. The I-structure has been developed to express an array of scientific computations and to permit pipelining execution between the producer and the consumer of the structured data. Since the structured data such as lists and trees are changed dynamically, we have selected the **reference count** method.

Dynamic modification of the structured data causes a complex access pattern. It is difficult to know the access pattern of the structured data before execution. There are, however, clues which will allow one to anticipate it. The structured data such as trees are accessed from the root to the leaves. Thus, the results of the operation on the structured data give a hint of the working set model for the structured data. For example. if a cell A is presently timing accessed, a cell B which is pointed to by cell A will be accessed in the near future. The working set model based on **the structure of the data** can be considered. However since access to the structured data occurs in parallel, it is not sufficient to make an efficient working set of the structured data. For this reason, a working set model based on several characteristics of the structured data should be made. In this paper, the working set model based on the read and write operations of the structured data and getting a new cell is proposed.

3 A Model for Data-flow Machines

Figure 1 shows a model for data-flow machines which have hierarchical instruction memories and structured memories. Each component of the machine can be briefly described as follows:

EUs: *Execution Units*
Each execution unit can execute an instruction, except for the list operation. All the execution units can execute in parallel.

PIM: *Primary Instruction Memory*
This memory contains active instructions. An instruction can take in input data only when it is placed in this memory.

SIM: *Secondary Instruction Memory*
This memory contains dormant instructions.

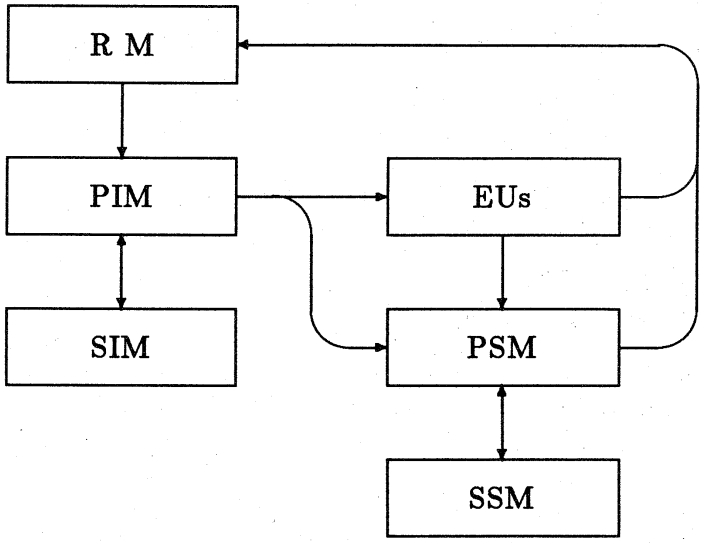

Figure 1: A model of a data-flow machine

RM: *Result Memory*
This memory temporarily stores result data from the EUs until they are taken in by all the instructions that use them.

PSM: *Primary Structured Memory*
This memory contains active data. List operations are executed only in this memory. The PSM also manage the reference count of cells for garbage collection.

SSM: *Secondary Structured Memory*
This memory contains dormant data. The SSM is much larger than the PSM, but only stores data.

4 A Basic Working Set Policies for Structured Data

There are two important points concerning the model of figure 1. One concerns the policies which manage the transfer of instructions between the PIM and the SIM. These policies have been described in previous papers[9,7,8]. The other concerns the policies which manage the transfer of data cells between the PSM and the SSM. There are two kinds of basic policies for structured data: structured data removal policies and structured data fetch policies.

4.1 Structured data Removal Policies

The **structured data removal policies** refer to policies for deciding which cells are to be remove from the PSM so that new cells can be transferred from the SSM. There are four simple policies:

SRP-1: No Policy.
A cell is removed in the order of the PSM id.

SRP-2: First-In First-Out (FIFO).
The cell which has been transferred from the SSM earliest is removed.

SRP-3: Last-In First-Out (LIFO).
The cell which has been transferred last from the SSM is removed.

SRP-4: Least Recently Updated (LRU).
The cell which is the least recently updated is removed.

4.2 Structured data Fetch Policies

The **structured data fetch policies** refer to policies for deciding when and which cells to transfer from the SSM to fill up the free memory cell in the PSM. A cell can be fetched on demand by the list operation. However, to decrease the execution time, a cell should be fetched before the execution of the list operation which uses that cell. Such policies are based on anticipation.

If a *head (car)* operation returns the pointer, the cell which is indicated by that pointer will be accessed in a short time span. The *tail (cdr)* operation also shows the same characteristic. Thus, the cells which are indicated by those pointers can be fetched beforehand in anticipation. There are four basic fetch policies which can be described as follows:

SFP-1: fetch on-demand.
A cell which is needed by the operations is fetched.

SFP-2: fetch on-demand and pre-fetch for head result.
This policy is the same as SFP-1 except that if a *head (car)* operation returns the pointer, the cell which is indicated by that pointer is fetched.

SFP-3: fetch on-demand and pre-fetch for tail result.
This policy is the same as SFP-2 except that the *tail (cdr)* operation is used instead of the *head (car)* operation.

SFP-4: fetch on-demand and pre-fetch for head and tail result.
This policy is the same as SFP-2 except that the *tail (cdr)* operation is also used with the *head (car)* operation.

5 The Hierarchical Structured Memory

5.1 Organization of Structured Memory

To discuss the hierarchical structured memory, the organization of the structured memory must be described in more detail. Figure 2 shows the organization of the hierarchical structured memory. In this model, each cell of the PSM can execute the operation on structured data in parallel. The cells are transferred with a *reference count block (ref)* and a *data block (head and tail)* between the PSM and the SSM.

In [3], the reference count management overhead is discussed. This overhead becomes excessive if the reference count is updated in each instruction. However, the reference count update frequency can be reduced with the reference count update instructions which explicitly update

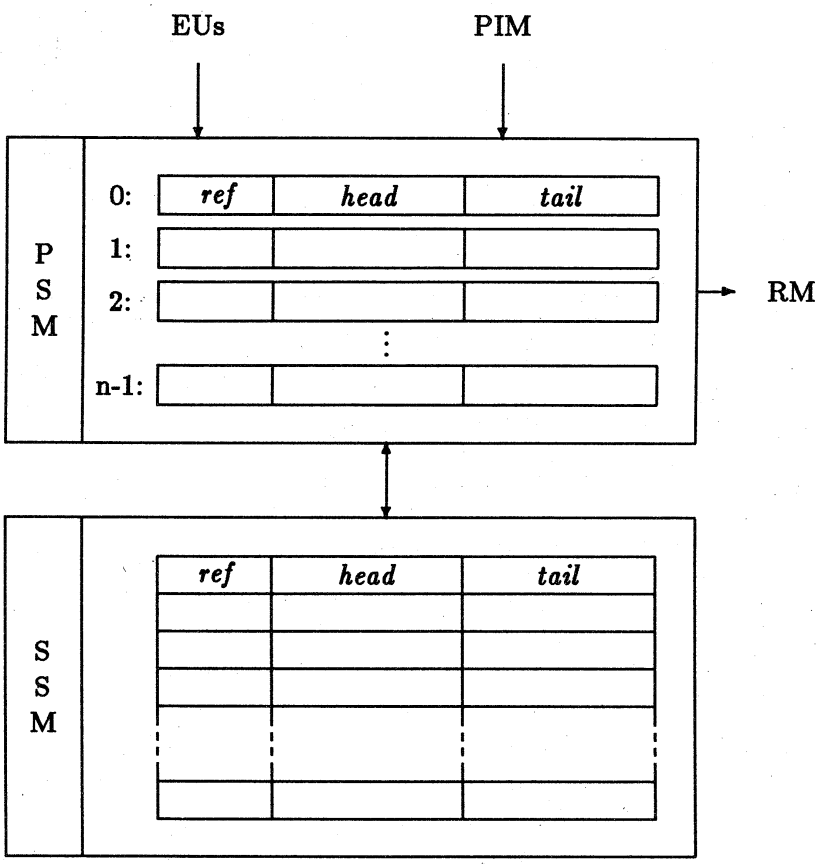

Figure 2: Organization of the hierarchical structured memory (dependent cell model)

a reference count. The reference count update instructions are executed at the beginning and the end of functions and blocks[1]. By this method, the execution time can be reduced by 30%[6].

In the hierarchical structured memory, the frequency of the transfer between the PSM and the SSM affects the execution time. The larger number of transfers between the PSM and the SSM causes a longer execution time. Thus, the method to decrease a number of transfers is required.

The strategy for garbage collection proposed by [3] is not suitable for the hierarchical structured memory since the reference count update is concentrated at the beginning and the end of functions and blocks. It causes an increase in transfers between the PSM and the SSM.

The function of the PSM can be divided into two parts. One part consists of the operations on structured data such as *head (car)*, *tail (cdr)*, and *cons*. The other part is the update of a reference count. Thus, we propose a new organization for the structured memory as shown in figure 3.

To avoid confusion, the organization of figure 2 will be called the **dependent cell model**, and figure 3 it will be called the **independent cell model**. In the independent cell model, the

[1]In this discussion, a functional programming language called **VALID** is assumed. The features of **VALID** are (1) a block structure, like { ··· } of the C and locality of value name, and (2) the single assignment rule.

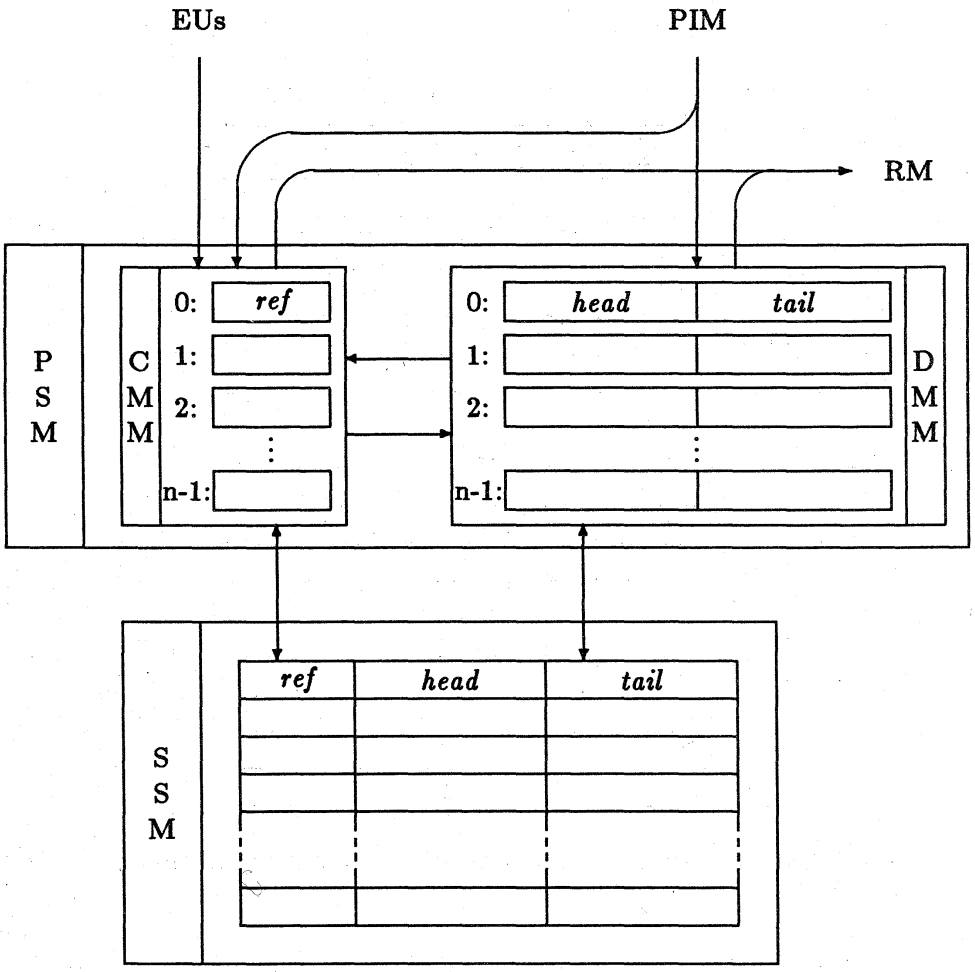

Figure 3: Organization of the hierarchical structured memory (independent cell model)

PSM is divided into two components. These components can work independently. The cells are individually transferred between the CMM and the SSM, and between the DMM and the SSM. These components of the PSM can be briefly described as follows:

CMM: *Reference Count Management Module*
The reference count is updated in this module. If the reference count becomes zero, the CMM requests the DMM to release the cells.

DMM: *Data Management Module*
The operations on the structured data are executed in this module (e.g., *head (car), tail (cdr), and cons*). When the content of the released cell is a pointer to other cell, the DMM requests the CMM to decrement the reference count of the specified cell.

There are three kinds of new strategies for hierarchical structured memory: the strategies for garbage collection, the strategies for writing data, and the strategies for getting a new cell.

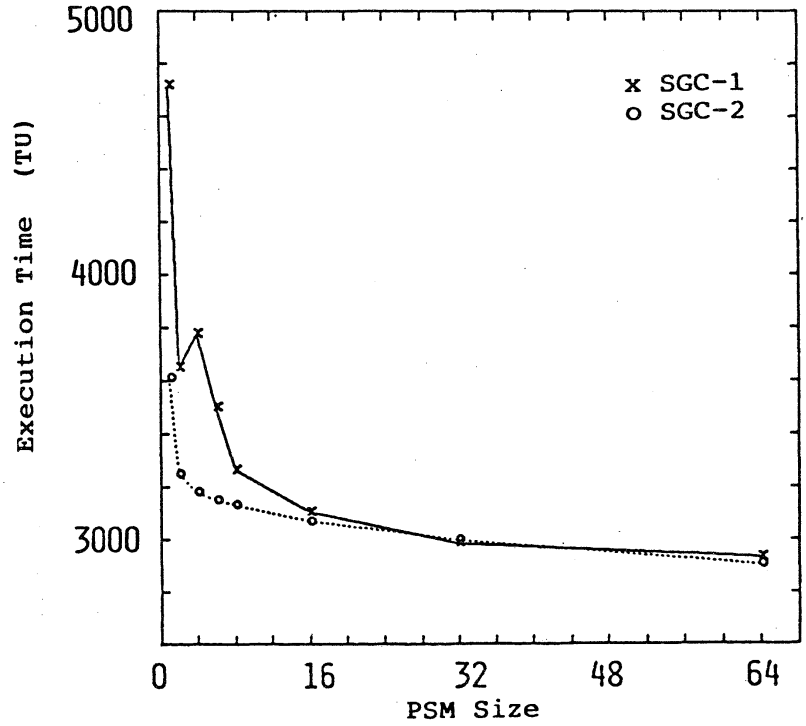

Figure 4: Simulation result for the strategies for garbage collection

5.2 The Strategies for Garbage Collection

The **strategies for the garbage collection** refer to strategies for deciding when the reference count should be updated. From the above discussion, there are two strategies for garbage collection:

SGC-1: update the reference count at the beginning and the end of functions and blocks.
The reference count should be updated at the beginning and the end of functions and blocks based on the **dependent cell model.**

SGC-2: update the reference count in every instruction.
The reference count should be updated in every instruction except when the DMM is busy based on the **independent cell model.**

5.3 The Strategies for Writing Data

In the independent cell model, the number of data writing into the DMM cell is small (maybe twice, that is head data and tail data writing). Thus, the write-through technique of the cache memory seems reasonable. There are three strategies for writing data:

SWD-1: write-back.
When the cell swaps out from the DMM to SSM, the contents of the cell should be written into the SSM.

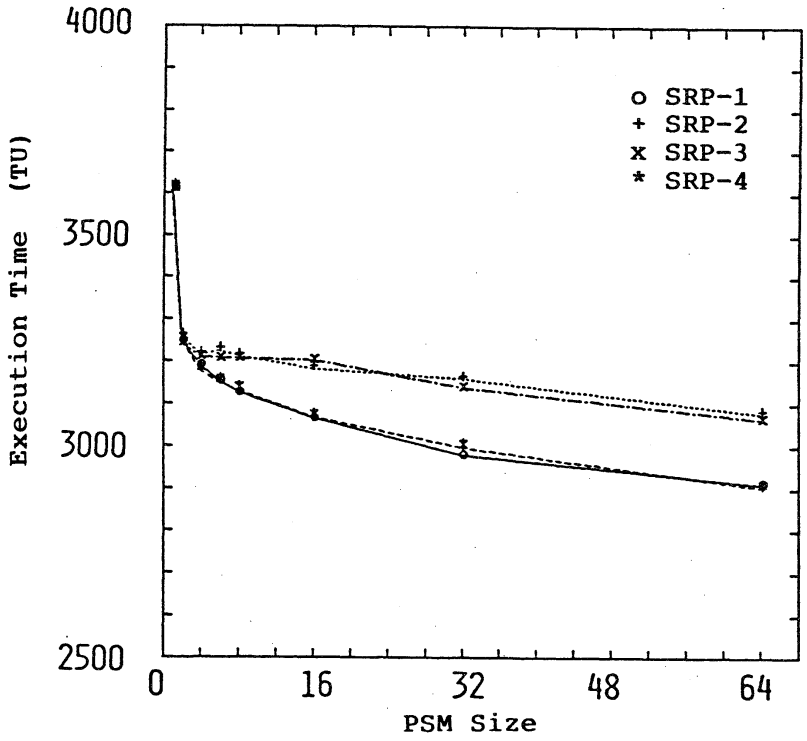

Figure 5: Simulation result for the structured data removal policies

SWD-2: write-through to the SSM when all of the contents are written.
When all of the contents of a cell, that is the data of *head* and *tail*, are written, that cell should be written into the SSM immediately. If there is a cell only one part of which is written on at swap out time, it should also be written into the SSM. Otherwise, it should be discarded.

SWD-3: write-through.
When the contents of a cell are modified, that cell should be written into the SSM immediately. A cell will be discarded when the cell is swapped out.

5.4 Strategies for Getting a New Cell

The operation for getting a new cell can be divided into two parts. The first part is getting an address of an unused (garbage) cell from the CMM. The second part is preparing a new cell on the DMM. An address of a new cell can be returned to the RM after getting an address from the CMM. By this strategy, preparing a new cell and firing of the next instructions can be overlaped.

There are two strategies for getting a new cell:

SNC-1: normal get-cell.
Returning the address of a new cell should wait for the completion of the operation of getting the new cell.

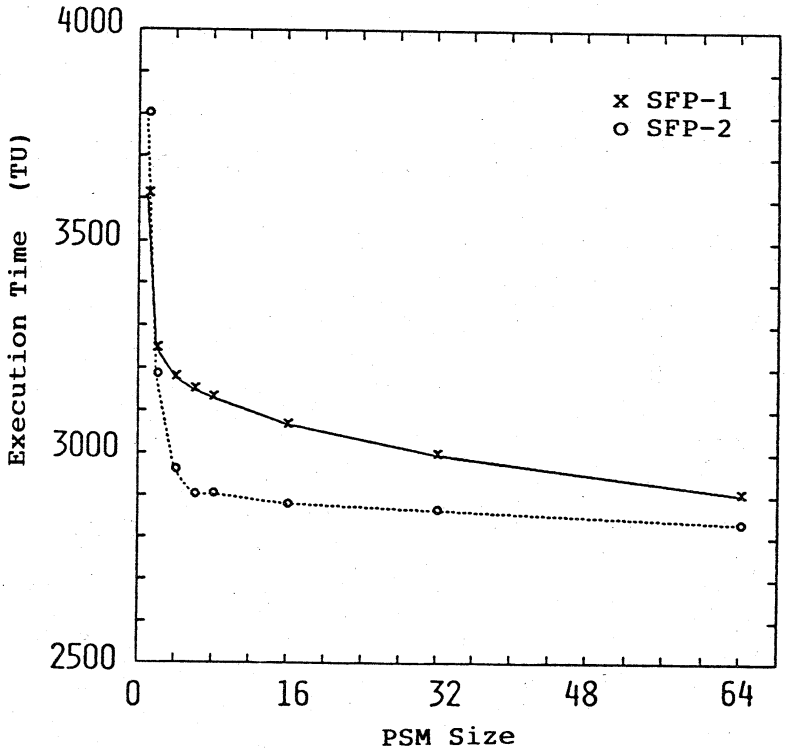

Figure 6: Simulation result for the structured data fetch policies

SNC-2: delayed get-cell.

The address of a new cell is returned after getting an address of an unused (garbage) cell from the CMM. Preparing a new cell on the DMM and firing the next instructions overlap.

6 Evaluation

Computer simulations have been done to evaluate the performance of the proposed policies and strategies. Since the purpose of simulation is to investigate the fundamental behavior of the data-flow programs under a limited resources environment, we have made the following assumptions in the simulations:

1. the execution time of each instruction is constant and is one unit of time [tu],
2. the time to transfer one page between the PIM and SIM is one unit of time,
3. the time to transfer one cell between the PSM and SSM is one unit of time, and
4. all the other consumed time, such as the memory access, is almost zero.

The test program for this simulation is the qsort(40) which sorts a list of 40 random integers by the quick-sort technique. This program consists of 18870 instructions (453 instructions access the structured memory). The get-cell instructions make the 386 new cells while being executed. The number of the accesses to the structured memory is about 4700. The average number of the active cells is 70 (maximum is 119).

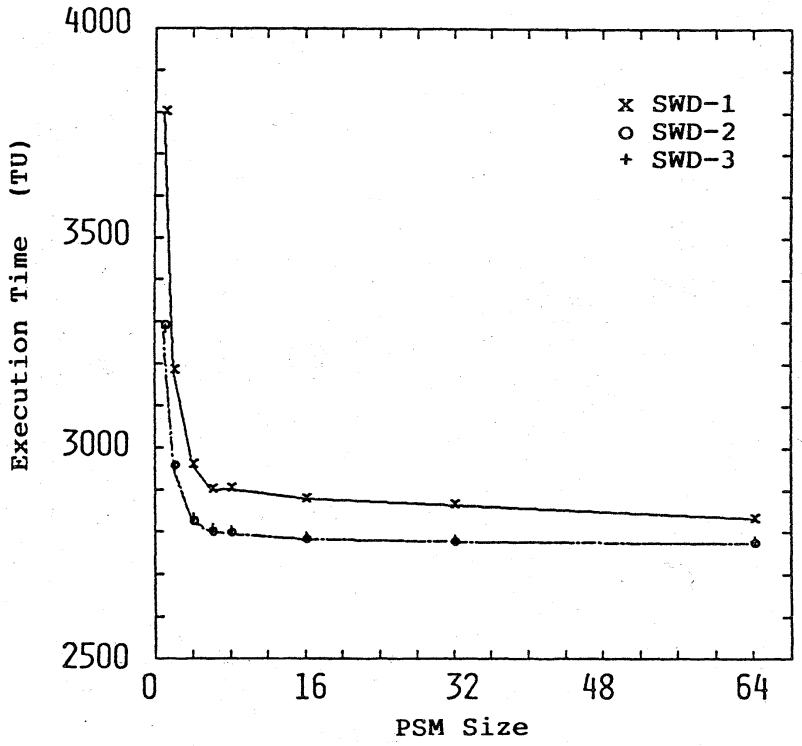

Figure 7: Simulation result for the strategies for writing data

Thity-two execution units are provided. These execution units can execute an instruction with the exception of the list operation. Three pages are provided as PIM, and one page can contain 16 instructions.

6.1 Evaluation of the Strategies for Garbage Collection

To reduce the combinations of the policies and the strategies, these strategies are evaluated first. The strategies for garbage collection (**SGC-1~2**) are compared with the execution time (Figure 4). **SGC-2** decreases the cell swapping time for garbage collection. As a result of this evaluation, **SGC-2** is used in the remaining simulation.

6.2 Evaluation of Structured data Removal Policies

The structured data removal policies (**SRP-1~4**) are compared with the execution time (Figure 5). **SRP-1** and **SRP-4** are more effective than **SRP-2** and **SRP-3**. Since a cell is also swapped in the PSM according to the order of PSM ids by **SRP-1**, **SRP-1** works similarly to **SRP-4**. In the remaining simulation, **SRP-4** is used.

6.3 Evaluation of Structured data Fetch Policies

In the test program qsort(40), the *head (car)* instruction never returns the pointer, so pre-fetch for the head result does not occur. For this reason, **SFP-4** is the same as **SFP-3**. The structured

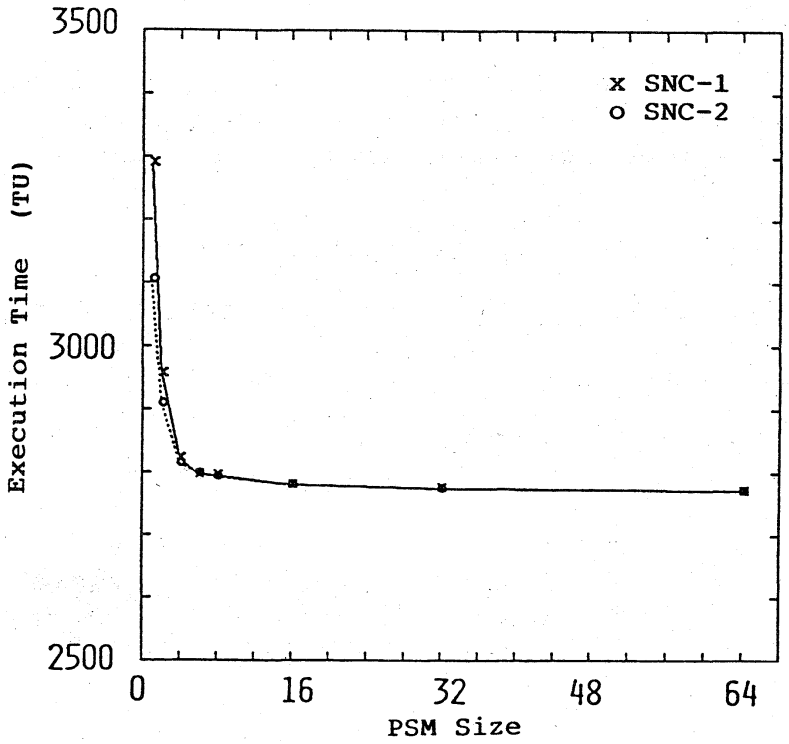

Figure 8: Simulation result for the strategies for getting a new cell

data fetch policies (**SFP-1** and **SFP-4**) are compared with the execution time (Figure 6). The pre-fetch policies work more efficiently than **SFP-1**.

6.4 Evaluation of the Strategies for Writing Data

The structured data writing strategies (**SWD-1~3**) are compared with the execution time (Figure 7). **SWD-2** and **SWD-3** are most effective and show the same effectiveness. Since the number of data writing is smaller than the number of times the data is swapped between the PSM and the SSM, the **write-through** is more efficient than the **write-back**.

6.5 Evaluation of Strategies for Getting a New Cell

The strategies for getting a new cell are compared with the execution time (Figure 8). **SNC-2** is almost the same as **SNC-1** when the size of PSM is large. However when the size of the PSM is small, **SNC-2** decreases the execution time since **SNC-1** causes many cells prepared by get-cell to be removed for the other cells before the *w-head* or the *w-tail* instruction. On this simulation, the network transferring costs from the PIM to the PSM and from the PSM to the RM are regarded to be zero. If these network transferring costs are not zero, the difference of the execution timing between the *w-head* or *w-tail* instruction and the *get-cell* is enlarged. Thus, **SNC-2** becomes more efficient.

7 Conclusion

This paper has proposed a hierarchical structured memory which uses the independent cell model. In this architecture, several policies and strategies for managing the working set of structured data were proposed and evaluated through computer simulation. From the results of the simulation,

- the garbage collection with independent cell model (**SGC-2**),
- removal of the structured data with the LRU policy (**SRP-1** or **SRP-4**),
- pre-fetch the structured data (**SFP-4**),
- the write-through (**SWD-2** or **SWD-3**), and
- the delayed get-cell (**SNC-2**)

worked efficiently. Consequently, the PSM needs only 10% of the of the maximum of active cells in this test program. The relation between the network transferring costs and the efficiency of the strategy for getting a new cell, and the working set size of PSM needed by a program are left for future work.

References

[1] W. B. Ackerman: "A structure controller for data flow computers," PhD thesis, MIT, January 1978.

[2] W. B. Ackerman: "A structure processing facility for data flow computers," In *Proc. of 8th International Conference on Parallel Processing*, 1978.

[3] M. Amamiya, R. Hasegawa, and H. Mikami: "List processing with a data flow machine," In E. Goto, K. Furukawa, R. Nakajima, I. Nakata, and A. Yonezawa, editors, *RIMS Symposia on Software Science and Engineering, Lecture Notes in Computer Science No.147*, pages 165–190, Springer-Verlag, 1982.

[4] M. Amamiya, R. Hasegawa, O. Nakamura, and H. Mikami: "A list-processing-oriented data flow machine architecture," *Proc. of NCC*, 143–151, 1982.

[5] Arvind and R. Thomas: "I-structures: an efficient data type for functional languages," Technical Report, Univ. of California Irvine, 1980.

[6] R. Hasegawa, H. Mikami, and M. Amamiya: "A list-processing-oriented data flow machine architecture its evaluation," *Trans. of IECE(D)*, J67-D(9):957–964, September 1984 (In Japanese).

[7] H. Sunahara and M. Tokoro: "Evaluation of working set algorithms for data-flow machines," In E. Goto, K. Araki, and T. Yuasa, editors, *RIMS Symposia on Software Science and Engineering II, Lecture Notes in Computer Science No.220*, pages 233–260, Springer-Verlag, 1986.

[8] H. Sunahara and M. Tokoro: "On the working set for data-flow machines: policies and their evaluation," In J. V. Woods, editor, *Fifth Generation Computer Architectures*, pages 147–160, North-Holland, 1986.

[9] M. Tokoro, J. R. Jagannathan, and H. Sunahara: "On the working set concept for data-flow machine," In *Proc. of the 10th Annual Symposium on Computer Architecture*, pages 90–97, June 1983.

[10] P. C. Treleaven, D. R. Brownbridge, and R. P. Hopkins: "Data-driven and demand-driven computer architecture," *ACM Computing Surveys*, 14(1):93–143, March 1982.

[11] Y. Yamaguchi, K. Toda, and T. Yuba: "A performance evaluation of a lisp-based data-driven machine (EM-3)," In *Proc. of the 10th Annual Symposium on Computer Architecture*, pages 363–369, June 1983.