1974

Applicability of buffered main memory to SYMBOL-IIR like computing structures

Om Prakash Agrawal
Iowa State University

Applicability of buffered main memory to SYMBOL-IIR like computing structures

by

Om Prakash Agrawal

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of
The Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Major: Electrical Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1974

TABLE OF CONTENTS

CHAPTER I. INTRODUCTION

The persistent demand for getting more performance for less cost has brought very rapid and profound changes in the speed of both the processors and the memory systems of computer systems. Over the last few decades the overall memory speed has been improved by a factor of 10 or 20 and the processing power has been consistently increasing at the rate of 100 times per decade (32,33). In spite of technological growth in the field of both the processors and the memory systems, there has been a severe mismatch between the speed of the processors and the main memory. This gap of speed has caused the performance of the present day computer systems to be limited by the speed and the capacity of the memory system.

In order that the performance of the computer system not be limited by the memory, there has been a constant effort to bridge this gap of speed between the processors and the main memory. This constant effort has produced a tremendous impact on the growth of technology and has given birth to a variety of technological and architectural innovations. Out of all these innovations, the concept of "buffering", a term also synonymous with "cache", has been found to be tremendously attractive and appealing. This concept of "buffering", implemented for the first time on a production system in 1968 in the IBM 360/85, a large scale computer

system, gave the user a machine which was capable of operating at the speed of the processor, with the cost of the slow backing store, and whose performance was no longer limited by the speed of the main memory (24).

The philosophy behind the concept of "buffering" is to use a quite fast and relatively small memory in between the processor and the main memory. Serving as a transparent bridge between the processor and the main memory, this "buffer" gives the illusion of a very large main memory operating at the speed of the "buffer", to the users. The concept of "buffering" has been found to be immensely attractive essentially because of two reasons--programming and economics. The programming reason is that most of the conventional users' programs tend to follow the principle of "locality", that is at any instant of time the addressing pattern of a program tends to be localized; the economical reason being that technology of today has not been able to provide a faster, larger and cheaper memory system operating at the speed of the processor. The memory cost seems to be inversely proportional to the speed and directly proportional to the capacity. Hence, larger and faster memories tend to be quite expensive.

"Buffering" tends to provide a localized subset of information at a faster speed and cheaper price to the processor and provides an illusion of a large main memory operating

at the speed of the buffer.

As long as these two characteristics are encountered the approach of "buffering" would seem to be the most effective solution of matching the speed between the processor and the main memory. For these reasons, "buffering" has been found to be immensely cost effective in large scale computer systems. Also, because of conceptual simplicity, the buffered memory approach has even been tried for mini-computers and has been found to be cost performance effective.

However, so far the approach of "buffering" has been tried for mostly conventional, software dominated von Neumann machines, and the "buffer" has been provided for the Central Processor only--which is supposed to be the most indispensable processor in the conventional computer system.

Recently, the consistent decline in the cost of hardware components (due to rapid progress in technology) and the consistent increase in the cost of software has led the computer architects to think of designing highly hardware oriented decentralized computing structures. The SYMBOL-IIR computer is one result of this new hardware/software analysis.

The SYMBOL-IIR computing structure can be characterized as a radical departure from conventional software dominated von Neumann machines. It is a highly decentralized system, organized as a network of dedicated or non-homogeneous or

autonomous processors (also known as Functional Units) each designed to do certain specific tasks. Besides this feature, most of the functions like memory management, system supervision, etc. are done in hardware (36,38). Even though it is a novel architectural milestone in the history of computer systems, the SYMBOL-IIR computer has a speed disparity between its processors and the main memory and between the main memory and the paging disk. These gaps make the system performance intuitively bounded by the speed of the memory system.

This thesis is concerned with the first of the gaps. If speed improvement is desired for the main memory, then there are two alternatives:

1. Either replace the whole main memory by a faster memory or

2. Use a buffered memory approach to improve the performance.

The second alternative is better than the first one because the second approach would tend to achieve about the same performance as the first approach with much less cost.

Problem Definition

The tremendous success of the "cache" approach with conventional computing structures (both large and mini) leads one to think that it would work for any kind of computing structure. Even though one would think of using the same approach in unconventional machines, no one has attempted so far to illustrate the designing of buffered memory systems for unconventional structures. Various interesting questions which are yet to be answered are:

1. Whether a "buffer" approach would be cost-performance effective also for an unconventional architectured computing structure.

2. Whether the special architectural organization of the unconventional machine would have any effect at all on the organization and the management of the "buffered" memory, and if so, what whould be the optimum "buffer" configuration and

3. If buffering is the best solution for improving the speed of an unconventional machine, how should it be organized and managed?

Specific Goals

Specific goals of this dissertation which are further elaborated later are:

1. To analyze the effect of the architectural organization of an unconventional computing structure upon the

design and management of its "buffered" memory.

2. To study whether in a multidedicated, non-homogeneous time-sharing computing system the buffer should be organized as a homogeneous unit or as a heterogeneous unit.

3. To study whether the buffer should be partitioned into equal sizes for different classes or whether the amount of buffer allotted to a particular class should vary depending upon its need and demand.

4. To study the overall organization and management of "buffered" memory.

5. To carry out a cost, speed and performance analysis for various buffer sizes, and various algorithms to see whether or not it is cost effective.

Significance of the Problem

The primary significance of this work is to provide additional insight into and shed additional light on several key problems in the design and management of "buffered main memory" for highly unconventional computing structures. The SYMBOL-IIR computer, because of its physical existence and availability serves only as a model or a vehicle. Emphasis has been placed on computing structures, having the similar major architectural philosophies as that of SYMBOL-IIR, and the attempt has been made to make the solution as general as possible. The chief significance of the problem could be

stated as follows:

"Given a computing structure like SYMBOL-IIR how should the buffered memory be designed and managed so that it is efficient, cheap and flexible enough?"

Hence, the title of the dissertation is "Applicability of buffered main memory to SYMBOL-IIR like computing structures" and not "Applicability of buffered main memory to the SYMBOL-IIR computer".

Approaches to Dissertaion Study

The problem has been approached as follows:

1. A study of the virtual memory system and the architectural organization of the SYMBOL-IIR and SYMBOL-IIR like computing structures and its effect on the design and management of the "buffered" memory is undertaken.

2. A program mix to generate suitable address sequences for computing the effective hit-ratio for various buffer configurations and algorithms, by simulation, is selected.

3. Finally a cost performance analysis of buffered system for SYMBOL-IIR like systems is made.

Significant Results

Three alternative ways of buffering SYMBOL-IIR like computing structures are investigated in detail. These three ways are--a buffer for the whole system, buffers for the terminals, or buffer for each dedicated processor. It is shown

that allocation of dedicated, sharable buffer space to each dedicated processor results in a very cheap way of improving performance significantly.

It is also demonstrated that besides the principle of "locality", the architectural organization of the whole system affects tremendously the design and management of the whole buffered memory system.

It is also shown that as long as large high speed memories are significantly more expensive than slower ones and the principle of "locality" is observed for for most of the users programs, buffering is the cheapest way for improving performance--even for highly unconventional computing structures like SYMBOL-IIR.

CHAPTER II. HISTORICAL PERSPECTIVE AND RELATED RESEARCH

Innovations for Speeding up the Memory

Various approaches have been implemented in various systems to reduce the effective access time for data and instructions to less than the full memory cycle time.

The approaches which are more common are the use of:

1. Multiple interleaved memory banks with single entry points,
2. Multiple interleaved memory banks with multiple entry points, and
3. The use of scratch pad

One of the major disadvantages of organizing memory as a single module consisting of a homogeneous collection of addressable storage is that it imposes a speed limitation, particularly in high speed systems, because a single module is internally a single bus scheme, that is only one access can take place at a time. Also, when the single module is serving a request, all other requests for it have to be locked out thus resulting in a large delay. Hence, large systems have attempted to reduce memory delays by dividing total storage into banks or modules of storage where each bank now contains a subset of the memory addresses. The partitioning of memory into several banks allows all the banks to operate simultaneously, delivering words to requestors (processors) asynchronously, without much lock out

or interference. Since each bank can still serve only one request at a time, lock out can still occur when the requests contend for access to the same bank.

The effectiveness of banking memory therefore depends in large measure on the distribution of addresses being generated by asynchronous units in the system, and also on the ability of the storage control system to queue requests on the busy modules.

The banking organization could be classified as either "high bit" banking or "low bit" banking depending upon whether the selection of a memory bank depends upon the high order or low order bits of an address. High order banking sort of partitions memory functionally. This scheme was implemented in Univac 1107 and 1108 where memory was sort of divided into I (Instruction) and D (Data) banks. Also known as FIFO and LIFO organizational structure, this was used in RCA's BIZMAC computer. The LIFO store technique was also employed in the Aeroneutronic Logic Evaluator, Burroughs B5000, the English Electric KDF-9 and the Ferranti Atlas computers.

Low order bit banking, also known as conventional interleaving, arranges the address structure so that adjacent words are stored in adjacent modules. Each independent module contains its own address decoding, driving circuits, data read out sense hardware and data register. By enabling multiple accesses to proceed concurrently storage bandwidth

is increased.

The number of modules that can operate concurrently is a measure of the speed improvement over a single module system. The depth of interleaving required to support a desired concurrency is a function of the storage cycle time, the processor memory request rate, and the desired effective storage cycle time.

Multiple interleaved memory banks with multiple entry points is the scheme of interleaving so that different memory modules can be accessed from different entry points. These different entry points could be connected to separate processors in a multiprocessing system. Now, the processors communicate with the memory through different entry points, through different communicating address and data registers.

The technique of scratch pad

Another approach of improving the performance of memory systems is the use of high speed control or scratch pad memories. This approach uses a small number of registers which are used to form a very small high speed memory and are used for storing temporary or intermediate results, frequently used data, constants and short subroutines which need to be iterated.

Pitfalls of the above approaches and the motivations for the buffered memory systems

Efforts such as that of pipelining (increase of overlap) greater local storage buffering (look ahead), multiple programming, parallel processing, deeper storage interleaving, more sophistication in the handling of branches, virtual memory, time-sharing and other improvements in the processors have been nice technological innovations for improving the performance of the computer system. However, all these attempts are only partially successful in bridging the gap between the speed of the processor and that of the main memory, and in making the computer performance not memory bounded.

One of the most important ingredients of a high throughput machine is to provide it with a very large main storage capacity, preferably operating at the speed of the processor, so that the machine will not be bounded by the speed of the memory and the processor could then issue a memory request at each and every processor cycle. However, it has not been feasible to provide a large main storage with a cycle time commensurate with the processor speed. Fastest memory devices are the most expensive per bit storage and the slowest memory devices are the cheapest per bit storage.

Because of these cost-speed trade-offs all the previous attempts of matching the gap between the processor speed and main memory speed has had resulted in a

microsecond/millisecond multilevel hierarchical virtual memory storage system, even though it was realized that to match the gap perfectly a nanosecond/microsecond hierarchy is the only solution.

This concept of nanosecond/microsecond hierarchy is nothing new. Its implementation had not been feasible only because of the lack of a suitable technology. In fact as early as 1962, Bloom, Cohen and Porter (8) had proposed a technique known as "Look aside" memory to improve the logic to memory speed ratio by the use of an associative memory. Lee had tried to simulate the concept of implementing "Look aside" memory and found that by using an associative memory of about 256 words of 100 nanosecond cycle time, an effective cycle time of about 350--400 nanoseconds could be obtained in a memory system--whose main memory cycle time was 1 microsecond (21,22,23). Wilkes, in 1965, had proposed a similar concept like this as a "slave" memory, by proposing a fast core memory acting as a slave to a slower core memory in such a way that in practical cases the effective access time was nearer to that of the fast memory than to that of the slow memory (44).

This concept of nanosecond/microsecond hierarchy was also implemented in embryonic form in other computers like Ferranti Atlas (20) and ETL-Mk-6 (39). In addition to the main immediate access stores of ferrite cores, Atlas had also

several thousand words of an entirely novel type of storage to which access was extremely fast (0.2 microsecond compared to that of 0.75 microsecond of ferrite core). This store consisted of a wire mesh with small ferrite plugs inserted in the spaces, the contents of the store being determined by the presence or the absence of the plugs. However, this was essentially used as a read-only storage and used essentially for storing subroutines and a large number of analytical functions only.

ETL Mk-6 also used a memory hierarchy of 3 levels consisting of a drum, core, and a tunnel diode memory of 250 nanosecond cycle time. The fastest memory was partitioned into a program stack, arithmetic stack and index registers--out of which the latter two only were accessible to the programmer. The idea of program stack was essentially to contain spaces for short loops.

All these techniques had been proposed and implemented in embryonic form only in certain computers. They had not been implemented in any large scale computer system because of lack of suitable technology. With the advancement of technology and with the availability of monolithic memory technology, the nanosecond/microsecond hierarchy was implemented for the first time in the IBM 360/85, a large scale computer system, in 1968 and was termed as "cache" system. In a cache based computer system, a fast and small memory

known as cache, interposed between the Cental Processor and the main memory, serves as the transparent bridge between their speeds. It is transparent in the sense that it is invisible to the user that is sort of hidden from him (Cache means hidden), and hence is not addressable by him. However its purpose is to make available to the processor, the pool of information currently being needed by it. However as the buffer memory is quite small it can't hold a large amount of information. The "cache" gives the illusion of having a large main memory operating at the speed of the "cache". Hence the processor tends to operate with a memory of cache speed but with a cost of that of main memory. This configuration has analogies with other systems employing memory hierarchies such as paged virtual memory systems. In contrast with this latter, however

1. A cache based memory system has a hierarchy of nanosecond/microsecond level, where as a paged virtual memory system has microsecond/millisecond hierarchy.

2. Cache deals with smaller blocks of data.

3. Cache provides a smaller ratio of memory access times (5 or 10 to 1 rather than 100 to 1) and because of this characteristic, a processor remains idle (that is does not switch to another task) while blocks of data are being transferred from main memory to cache and

4. A cache based system enhances the effective speed,

where as a paged virtual memory system tends to enhance the apparent size of the memory.

The reason cache memory works so nicely with a conventional computer system architecture is essentially one of programming--generally users programs tend to follow the principle of "locality", that is addressing pattern of a program is not distributed uniformly through the whole memory capacity, but at different intervals of time it tends to be localized to a particular subset of memory. This principle is known as "locality of reference". Usually programs tend to have two kinds of locality of reference--"spatial locality" and "temporal locality" (25). This "locality of reference" principle illustrates that if a block of words is brought from main memory to cache, then the probability of words in cache to be used next by the program are very high and then they would be accessed at cache memory speed instead of the main memory speed. This property has had significant impact upon the architectural design of computer systems. This very principle has given birth to the concept of the "Working Set" model of program behaviour (11) and also to the principle of paging in virtual memory systems.

Paged virtual memory systems tend to achieve the speed of main memory at the cost of the auxiliary memory, whereas cache memory systems tend to achieve the speed of cache memory at the cost of the main memory and the auxiliary memory.

Related Research

Trying to implement a "buffer" memory for a large scale computer system, for the first time, Liptay showed that a "cache" memory of 16K bytes of storage (extendible to 24K bytes or 32K bytes) and of 80 nanosecond cycle time, operating with a main memory of 512K to 4096K bytes and cycle time of 1.04 microsecond, was equivalent in performance to 81% of the performance (in average) of a memory system consisting of 512K to 4096K bytes operating at the "cache" speed (24). "Great effort and thousands of hours of machine time have been expanded in proving the feasibility of a buffer technique for the IBM system/360 model 85" (10). Since its inception in 360/85, cache memory has been used in 360/195, 370/155, 165 and 195 models, and there have been numerous investigations for the applicability and design considerations of buffered memory systems for various kinds of computer systems.

Conti (10) and Mattson et al. (27,28) have proposed and developed various techniques for evaluating hierarchical storage systems (including high speed buffer storage) and their system effectiveness. The stack algorithm proposed by Mattson et al. (27,28) can be used essentially for evaluating variable class, variable page size, multilevel memory systems. Arora and Wu (1) have tried to study the performance of a cache memory system by analyzing statistical

quantification of instructions and operand traces. They have also tried to investigate parameters which could allow simulation of various non-existing environments. Mattson (26), Kaplan and Winder (19), and Meade (29,30) have conducted extensive studies on the effects of buffer size and block size on the hit ratio for a variety of computing environments. Meade (31) has discussed about designing various parameters of a buffered memory system in an excellent article in Electronics. Bell and Casasent (3,4) tried to investigate the applicability of buffered memory systems for mini-computers and came to the conclusion that a performance gain of 5 or more can be achieved at the cost increase of 2 or less for PDP 8/E computer systems. Pohm et al. (34,35) have tried to investigate the applicability of buffered memory systems for both large and mini-computer systems, and have demonstrated that buffering a 500K bytes main memory would result in a 300% improvement in performance with a cost increase of 8%. Bersamian and DeCegama have studied system design considerations of cache memories on a multiprogramming system and have come to a conclusion that "hit probability may significantly decline in multiprogramming systems with high program switching rate. As a result the performance efficiency of cache may be negligible" (5).

In a nutshell, in all the studies taken so far buffering seems to be immensely cost-performance attractive for both

large and mini computer systems. However most of the research for buffering has been concentrated on conventional computing systems, and buffer space has been provided for the Central Processor only. As far as the author knows, only in a super Japanese computer an effort has been made to provide separate buffer area for the basic processor and the I/O Processor (of the Central Processor) (9). The speed and capacity of the buffers for these two different processors are however different.

One of the major goals of this dissertation is to investigate the applicability of separate dedicated buffer space for different dedicated processors of a multidedicated processors computing system.

CHAPTER III. APPLICABILITY OF BUFFERED MAIN MEMORY TO SYMBOL-IIR LIKE COMPUTING STRUCTURES

Introduction to the SYMBOL-IIR Computing Structure

Before considering the applicability of buffered main memory to SYMBOL-IIR like computing structures it is a good idea to discuss briefly the existing SYMBOL-IIR computing system. The SYMBOL-IIR computing system is a time-sharing virtual memory computer system, consisting of 7 dedicated or autonomous processors all sharing the same virtual memory (Fig. 1). The access to virtual memory of these dedicated processors is controlled by another dedicated processor called the Memory Controller. The purpose of this processor is to allocate memory access dynamically to different processors on a priority basis. Any time a processor needs access to main memory, it raises its priority line telling Memory Controller that it needs access to memory. If it happens to be the processor having highest priority trying to access memory at that time, the access to memory is granted. However, if other processors with higher priorities than it are also trying to access the memory at the same time, then it has to wait until all the requests for higher priority processors are satisfied. The priorities of these different dedicated processors are fixed, that is they are static and do not vary dynamically. The priorities are assigned on the basis of relative importance of each dedicated processor.

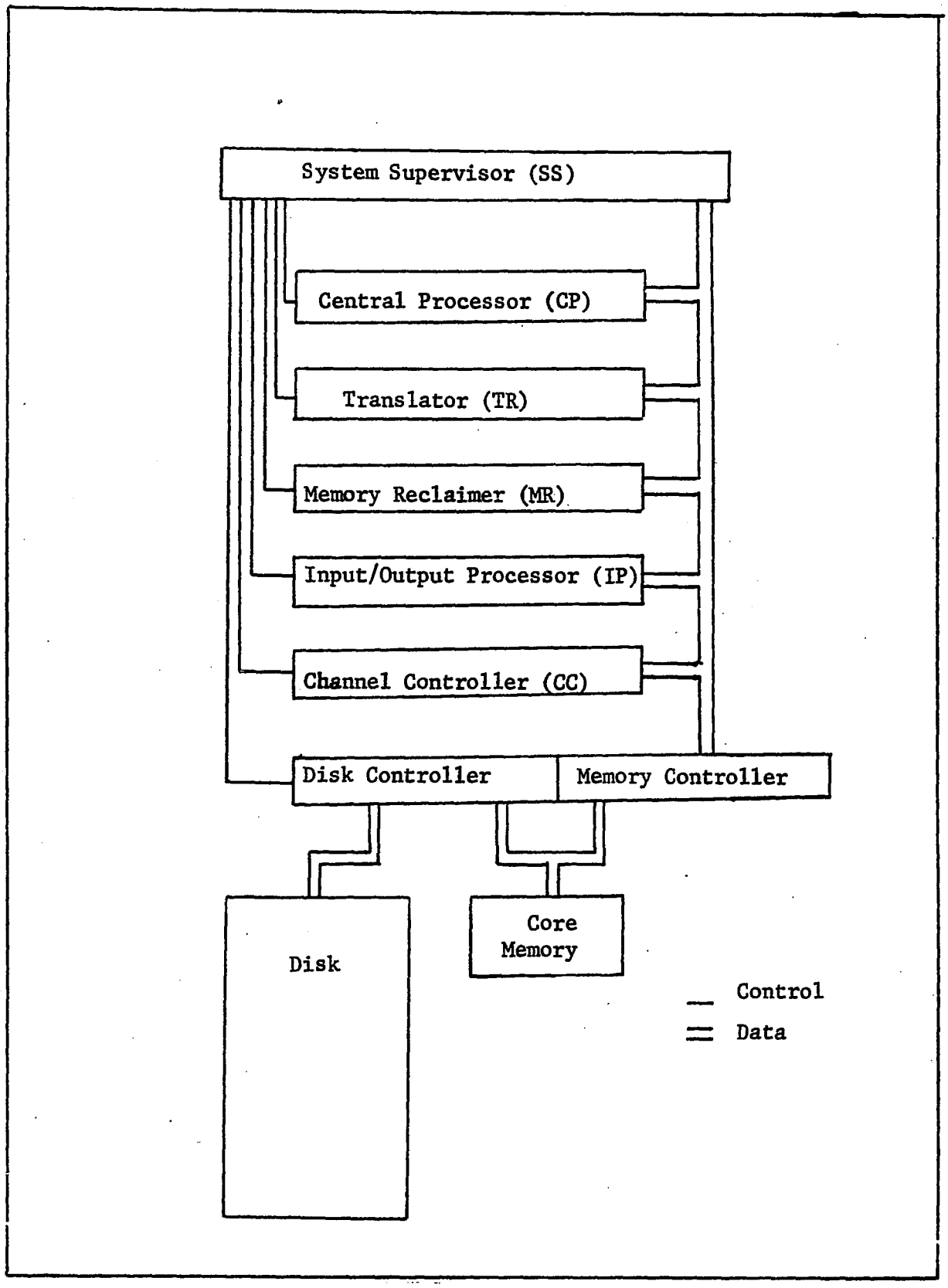

Fig. 1. SYMBOL-IIR System

The SYMBOL-IIR computing structure is a radical departure from conventional software dominated von Neumann machines. It is a highly hardware oriented machine, and its major architectural philosophy is to implement, in hardware, a variety of features which are usually implemented by software in conventional computing structures. Some of these interesting features are dynamic memory allocation and reclamation, dynamically variable field lengths and structures, automatic memory management and time-sharing supervision, direct symbolic addressing, alpha numeric field manipulation, direct text editing, etc. (36,38).

Virtual Memory System of the SYMBOL-IIR

In a buffered virtual memory system, the architectural organization of the virtual memory system has tremendous impact upon the organization of its buffered memory. Hence before discussing the buffered virtual memory system of SYMBOL-IIR like computing structures, the virtual memory system of the SYMBOL-IIR computing system, and some of its characteristics are discussed. Virtual memory system of the existing SYMBOL-IIR computer is organized as a two level paged memory system--the first level being a core memory consisting of 32 pages, and the second level being a disc with total capacity of 800 pages. Out of 32 pages of physical core one page is reserved for system usage and is known as the system header page, and three pages are used for

terminal header information. The rest of the pages of the core are used as virtual data pages by different processors for different terminals (36,38) (Fig. 2).

Page organization

Each virtual page is 256 words, and is organized as 32 groups, where one group is equal to 8 words and is the basic quantum of memory space allocated dynamically by the memory controller (37,45). Since memory space is allocated dynamically, the groups allotted to a processor by the memory controller can cross logical page boundaries, and since extensive list manipulation is done in the system to keep track of virtual memory space, each group has a group link word associated with it. The group link word contains information about the forward link of the present group to the next group and also the backward link of the group. These links are maintained dynamically by the memory controller. In SYMBOL-IIR group link words of a group are maintained in the same virtual page as the group itself. Choosing the group size to be 8 words and also having the group link words in the same page makes each virtual data page of SYMBOL-IIR look like (Fig. 3). The virtual data page has 28 groups of data, 28 group link words and 4 words of page header information.

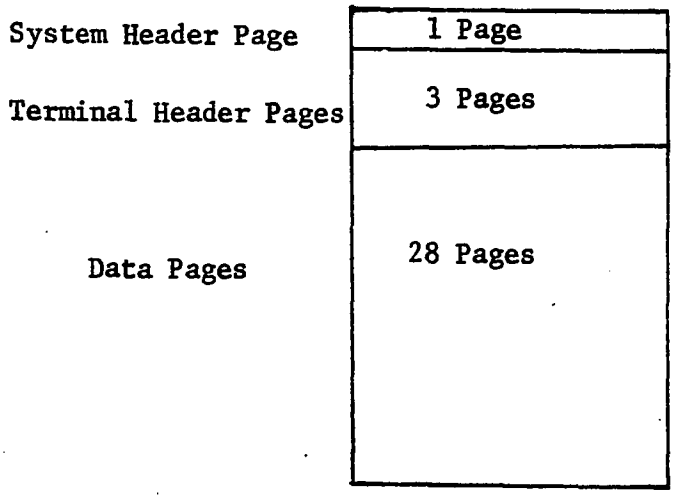

Fig. 2. Main Memory of the SYMBOL-IIR

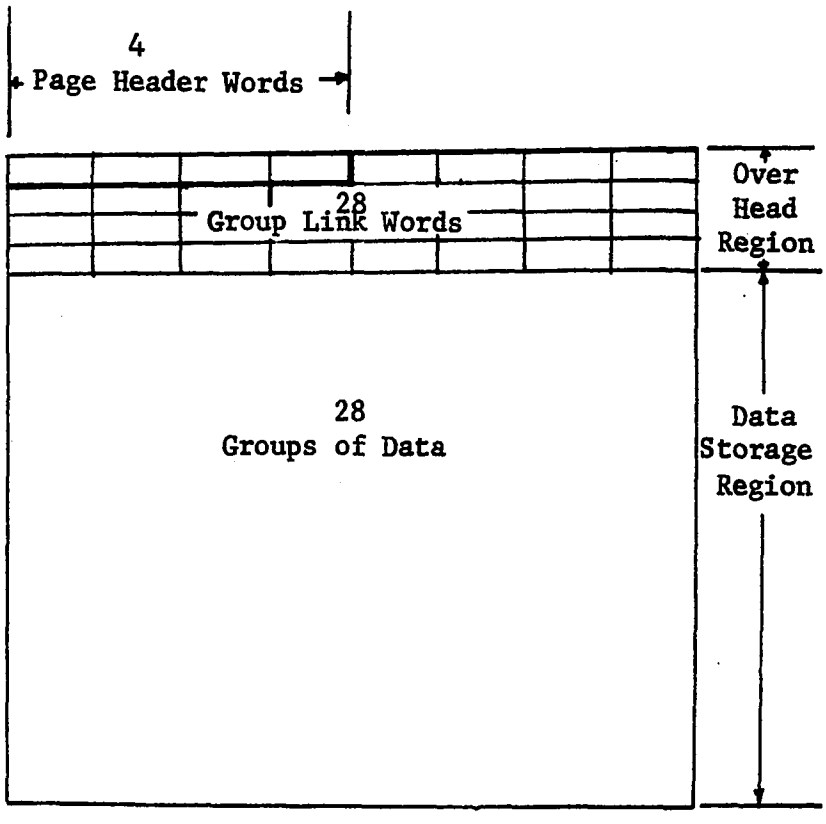

Fig. 3. Page Organization of the SYMBOL-IIR

Page header information contains information about who owns the particular virtual page, which next virtual page it is linked to, whether or not space is vacant in the page and how much of it etc. This information is used extensively by the Memory Controller for dynamic memory allocation and reclamation.

Some characteristics of the Memory System of SYMBOL-IIR

One important characteristic of the SYMBOL-IIR computer structure is the segregation of virtual memory pages according to their use. All virtual memory pages of SYMBOL-IIR are used by different dedicated processors for different terminals, and these are essentially used for one of three purposes. The pages are used for source page lists (the list of pages used for source string only), object page lists (the list of pages used for object string only) or for name-table and data page lists (the list of pages containing the name-table and data).

Another important characteristic of the SYMBOL-IIR computer structure is the principle of "no-sharing" of information by different terminals, that is each terminal is the sole and exclusive owner of its three page lists once it aquires them from the system, until it releases them to the system to be used for some other purposes. However, a terminal needs the service of various dedicated processors for the completion of its job. Hence, even though no sharing of in-

formation between terminals is allowed the processors share information belonging to a terminal. These philosophies have very profound effects on the techniques of buffer space allocation as mentioned later.

Since we are mostly concerned with SYMBOL-IIR like computing structures, we do not intend to answer, here, questions like why sharing of information between terminals was not allowed in the system. We just consider systems which have similar major characteristic philosophies as that of SYMBOL-IIR.

Introduction to SYMBOL-IIR like Computing Structures

SYMBOL-IIR like computing structures could be characterized as computing structures having similar major architectural philosophies as that of the SYMBOL-IIR computer, and as far as the problem of applicability of buffered main memory to SYMBOL-IIR like computing structures is concerned, they could be viewed as in Fig. 4.

The virtual memory system of a computing structure like SYMBOL-IIR could be characterized as a paged memory system consisting of two levels of memory--the first level consisting of relatively fast core of M number of pages and the second level consisting of a drum or disc of N number of pages (Fig. 4). Let us call the core cycle time of the first level or main core memory as Tmcyc and the auxiliary store access time as Taux. For SYMBOL-IIR like computing struc-

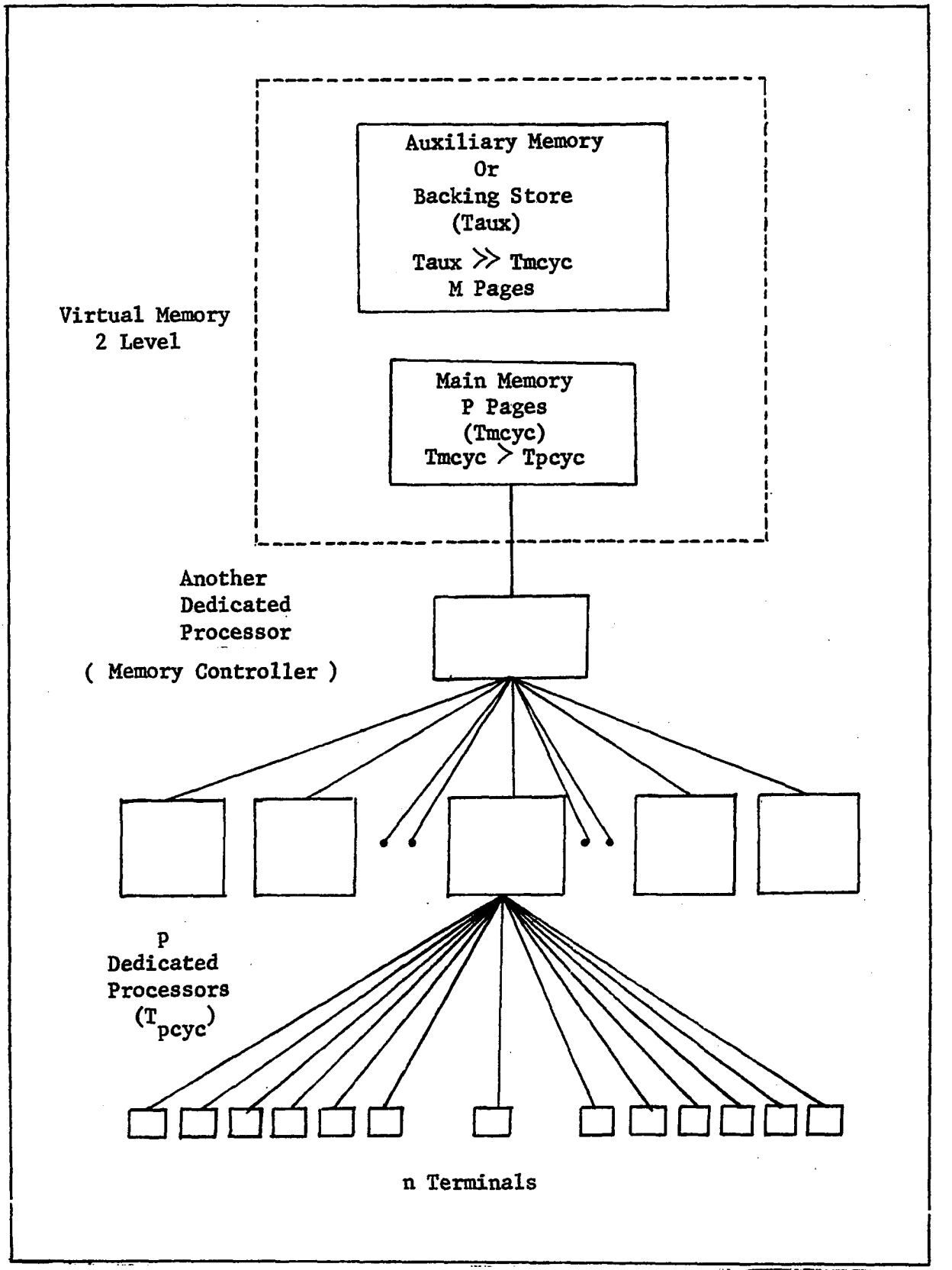

Fig. 4. SYMBOL-IIR like Computing Structures (2 level)

tures Tmcyc is in the range of microseconds and Taux is in the range of milliseconds. So, Taux is about 1000--5000 times that of Tmcyc, that is there is a speed disparity of about 1000--5000 between the first level and the second level.

Let all the processors of SYMBOL-IIR like computing structures have the same processor cycle time and let this be denoted by Tpcyc. For a SYMBOL-IIR like computing structure, there is a speed discrepancy of about 10 between the processor cycle time (Tpcyc) and the main storage cycle time that is Tpcyc = 10 Tmcyc.

Need for buffering

In SYMBOL-IIR like computing structures the main fast memory serves as a buffer for the relatively slow auxiliary memory. However, as noted before, there is a wide speed discrepancy between the two memory speeds, and also there is a big speed disparity between the basic processor cycle time and the main memory cycle time. Even though the virtual memory system tends to give an illusion of infinite transparent main storage to different users, there is a big difference in the effective speeds between the hits (i.e. the time when the required page is in the main memory) and the misses (i.e. the times when the required page is not in the main memory and has to be brought from auxiliary memory) This might involve a transfer of a page from core to auxiliary memory first be-

fore the required page is brought from the auxiliary memory to main memory. This speed gap between the two levels of the memories and between the processors and the main memory of SYMBOL-IIR like computing structures make the speed of the whole computer system inherently bounded by the speed of the memory.

For SYMBOL-IIR like computing structures the ideal case would be to have one large storage unit with cycle time (Tmcyc) equal to Tpcyc. The processors can then issue storage requests on any and every processor cycle. However, because of the cost/speed trade-off considerations it is not physically feasible to provide a storage system in commensurate with the processor cycle time. The ideal solution of achieving greater speed with minimum cost is to provide buffering by using two memories--one called the buffer or cache, being small, cheap and fast enough to match the speed of the processor and situated physically very close to the processors for quick accessibility and the other relatively cheap, and slow main memory but able to transfer a large amount of data into the small buffer main memory in a single cycle.

Hence, the buffered virtual memory system of SYMBOL-IIR like computing structures could be thought of as a three level memory system--the first level being the very fast, small capacity buffer, the second level being relatively

slow, medium capacity main memory, and the third level being the slowest, very large capacity backing store (Fig. 5).

Main Memory Organization of SYMBOL-IIR like Computing Structures

As mentioned before, one of the important philosophies of SYMBOL-IIR like computing structure virtual memory system is the segregation of virtual memory pages according to their use. Also each virtual page sort of consists of two parts--data groups and data group linking words. One of the reasons for providing group linking words and page header words along with data groups in the same page is the simplicity of the address translation scheme.

However, there is no reason why main memory could not be segregated into different modules. Hence, trying to retain the same basic concepts of SYMBOL-IIR and at the same time partitioning the main memory, let us organize the main memory of SYMBOL-IIR like computing structures as follows:

The main memory is thought of as partitioned into three separate major modules--called data module, data linking module, and system and terminal header module respectively.

The System and terminal header module could contain essentially the system and terminal header information like the first 4 pages of the SYMBOL-IIR computer. Since these pages have to be resident in core most of the time, segregating them in a separate module helps to access them in parallel

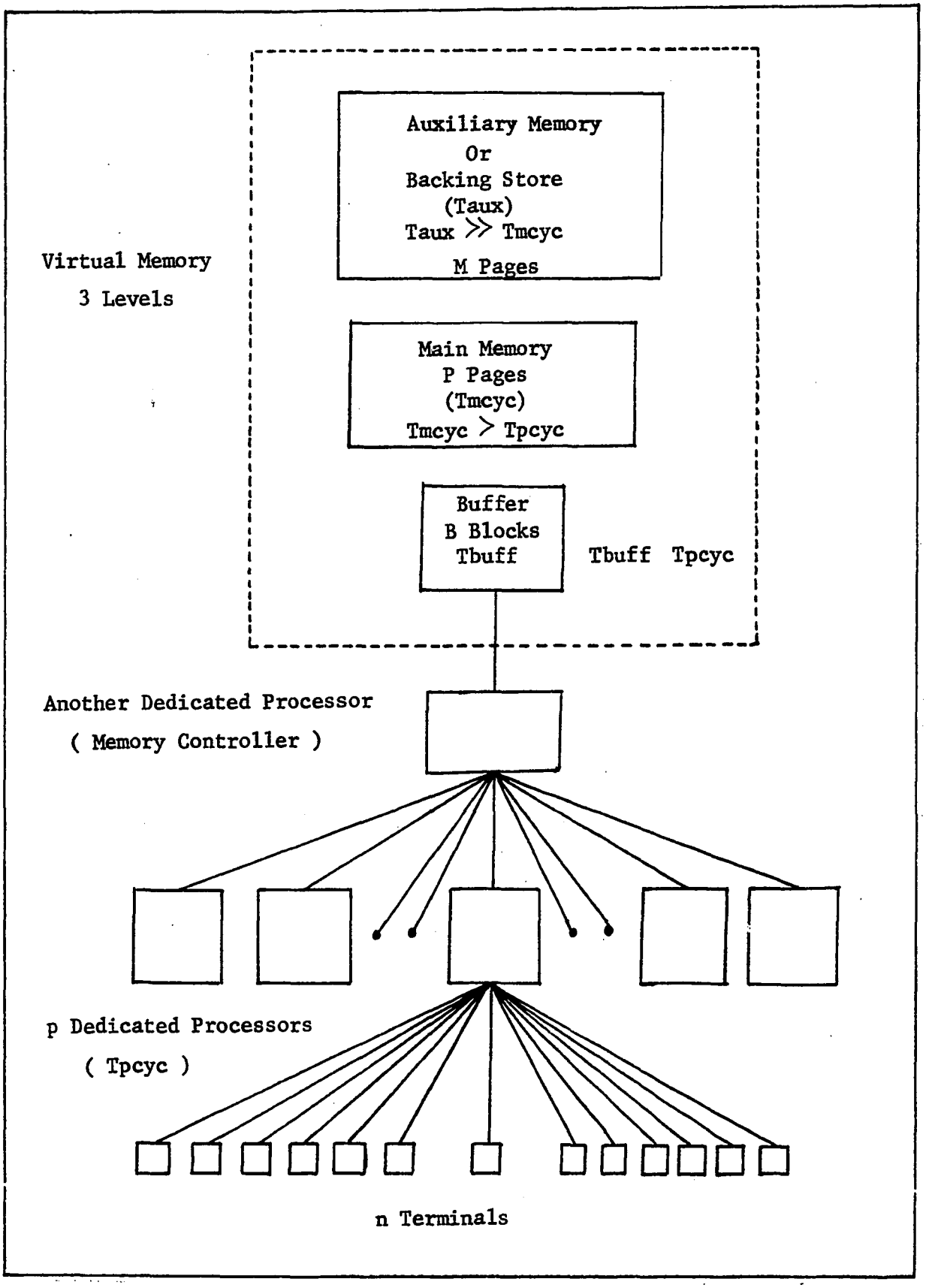

Fig. 5. SYMBOL-IIR like Computing Structures (3 level)

with other information. Also, since it would not be too big, it could be designed as a separate fast memory module.

The data linking module will be the module containing the page header words and data group link words of all the virtual pages of the system. The main reason for this segregation is to achieve simultaneously almost all the important information needed by different processors and to achieve more parallelism. It is thought that segregating these data into different modules and providing the capability of accessing all the data at the same time can reduce the overall computation time and minimize the number of accesses to each module.

Data module(s) will be module(s) containing only data. Hence in essence the whole main memory system of SYMBOL-IIR like computing structures would look as that in (Fig. 6).

Buffer Management and Organization of SYMBOL-IIR like Computing Structures

If a buffer is to be provided for SYMBOL-IIR like computing structures, an important question one has to answer is how the buffer should be organized and managed.

An efficient scheme of implementing a buffered memory system for SYMBOL-IIR like computing structures involves efficient design of the following strategies:

1. Buffer space allocation strategies
2. Buffer space loading or placement strategies

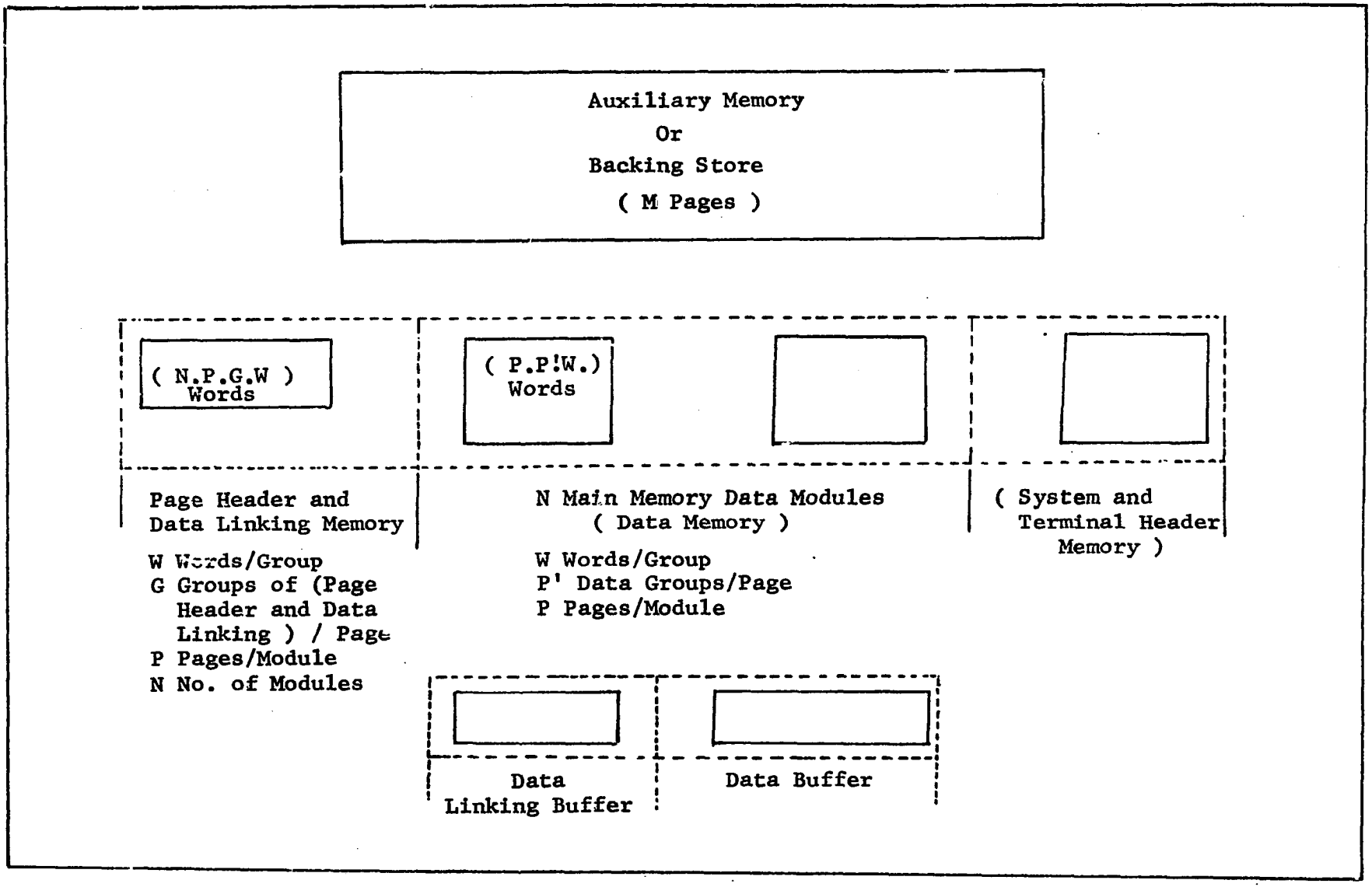

Fig. 6. Virtual Memory Organization of SYMBOL-IIR like Computing Structures

3. Buffer space replacement strategies and

4. Buffer address translation strategies.

The design of these four strategies should be approached with the goal in mind that the buffer should be cheap, small, fast, and flexible enough, and yet should maximize the success references to the buffer.

If technology of today could provide an inexpensive, large, and fast memory (serving as a buffer) then the whole problem of worrying about designing these efficient strategies would not be that important.

Basis for buffer space allocation

In trying to design an efficient space allocation strategy for SYMBOL-IIR like computing structures, one should give some thoughts to the following questions:

1. How the buffer space should be allocated on size and associativity considerations,

2. Whether it should be allocated on a physical or functional basis,

3. Whether it should be organized as a homogeneous unit or it should be organized as heterogeneous units, and

4. How should it be partitioned? Should it be divided into a fixed or variable number of classes? and, if so, how the partitioning of buffer space should be done for different classes, that is should all classes have equal buffer space or should the amount of buffer space allotted to a

class vary depending upon its need.

Let us discuss these problems in some detail.

Size and associativity considerations

Based on the size and associativity considerations one might approach to allocate buffer space for SYMBOL-IIR like computing structures in any of the following three ways:

1. Fixed page size, variable associativity
2. Fixed or variable number of blocks, fixed associativity and
3. Fixed number of blocks, variable associativity.

Fixed page size, variable associativity Taking the approach as is used in conventional computers, one approach might be to organize the buffer space as consisting of a fixed number of pages, the page size being fixed and same as the page size of the backing store. The buffer would appear now as an anonymous pool of page frames, where each page frame of it could be associated with any virtual page of the backing store (variable associativity) (Fig. 7). However, as the information would be transferred between main memory and the buffer in blocks instead of a page (a block being some fraction of a page), this approach will need some sort of validity bits to be associated with different blocks of the page. This approach works fine for conventional machines and has been implemented in various machines (24). However, for SYMBOL-IIR like computing structures this approach does not

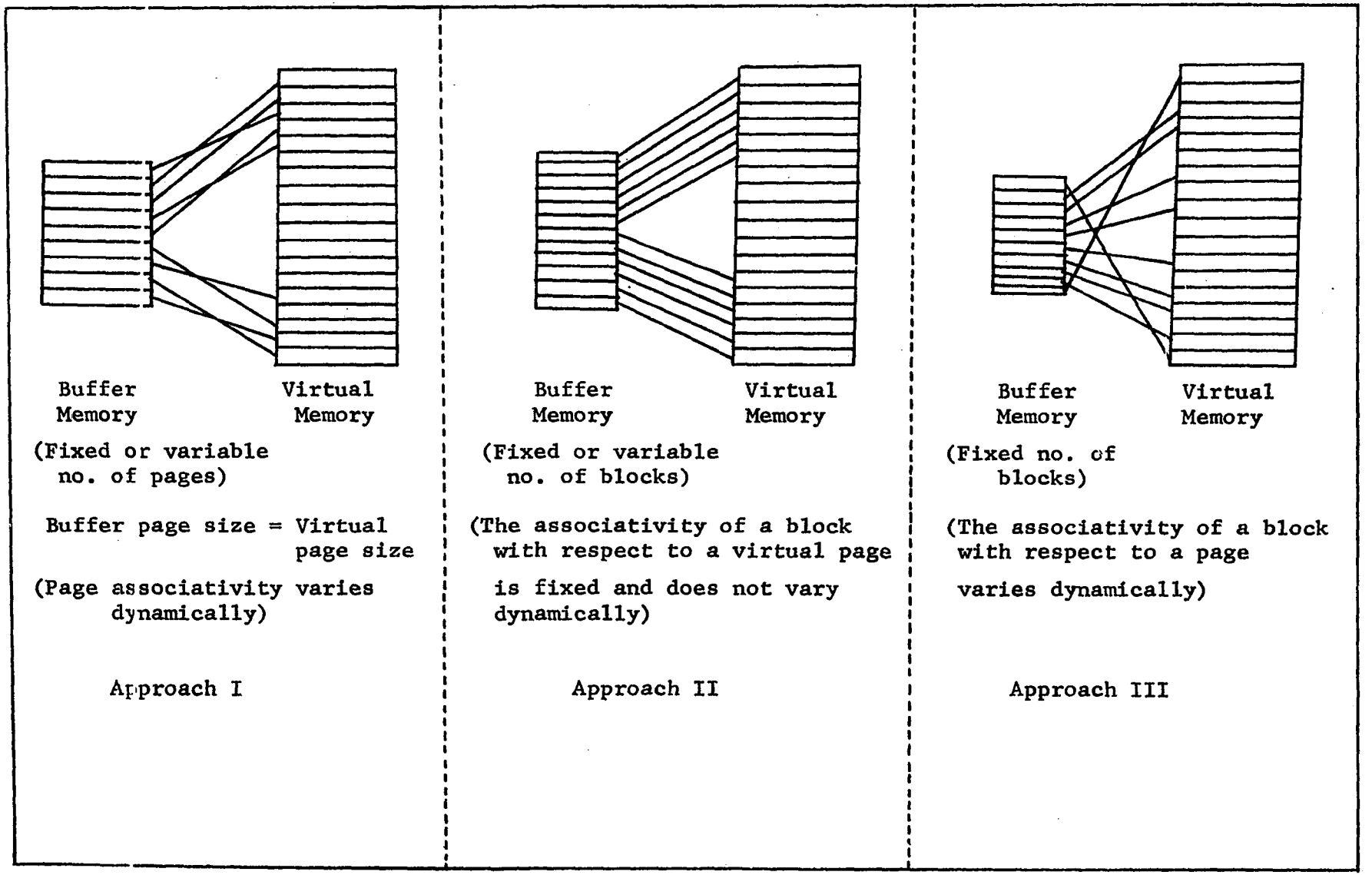

Fig. 7. Three Approaches for Size and Associativity Considerations

seem to be too exciting, because in SYMBOL-IIR like time-sharing computer structures, virtual pages are segregated according to their usage and each terminal needs 3 page lists for the execution of its job. Also, terminals are not allowed to share each others information; and since buffering has to be provided for the system operating in a time-sharing environment, the need for providing sufficient buffer space to obtain a better hit-ratio might result in the need for a buffer of quite large size (larger than the present core of SYMBOL-IIR) thus making it quite expensive.

Fixed or variable blocks size, fixed associativity

Because terminals are not allowed to share each other's information the approach of fixed pages and fixed sizes with variable associativity loses much of its charm. Hence another approach might be that of buffering a fixed or variable number of blocks per page with fixed associativity (Fig. 7) i.e., associate these particular blocks with the particular virtual pages only. Here the idea is to allocate a certain amount of buffer space for each virtual page (unlike the whole page as in the previous approach). The total buffer space now would consist of m * N blocks, where virtual memory size is N pages (each page being of the size of Q blocks) and m is an integer with $m/Q << 1$. Now the associativity of each block is fixed or static with a virtual page and does not vary with time. If m is equal to 1 then it results in the

simplest type of buffer organization, there being no need for the priority update list or chronology (41). Any time a particular block belonging to a certain virtual page is not found in the buffer, then the corresponding block of that virtual page is replaced by the new required block. There is no problem of deciding which block is to be replaced. However, if m > 1 then there has to be a priority list associated with the blocks of each virtual page. In case of a miss, the block having the lowest priority amongst all might be replaced.

Even though this approach seems conceptually simple, buffer size tends to be proportional to virtual memory size. Increasing the size of virtual memory would result in an increase of buffer size (Fig. 8).

Fixed number of blocks, variable associativity Another approach which would tend to reduce the problem of buffer size being proportional to virtual memory size is to design the buffer to be of fixed size, consisting of a fixed number of blocks, where the associativity of a particular block with a virtual page is varied dynamically. Now each virtual page does not have a fixed amount of buffer space for it, but there is a fixed amount of buffer space for a variable number of virtual pages. Hence, based on size and associativity considerations, this approach of small and fixed buffer size (and hence economical) with dynamic block

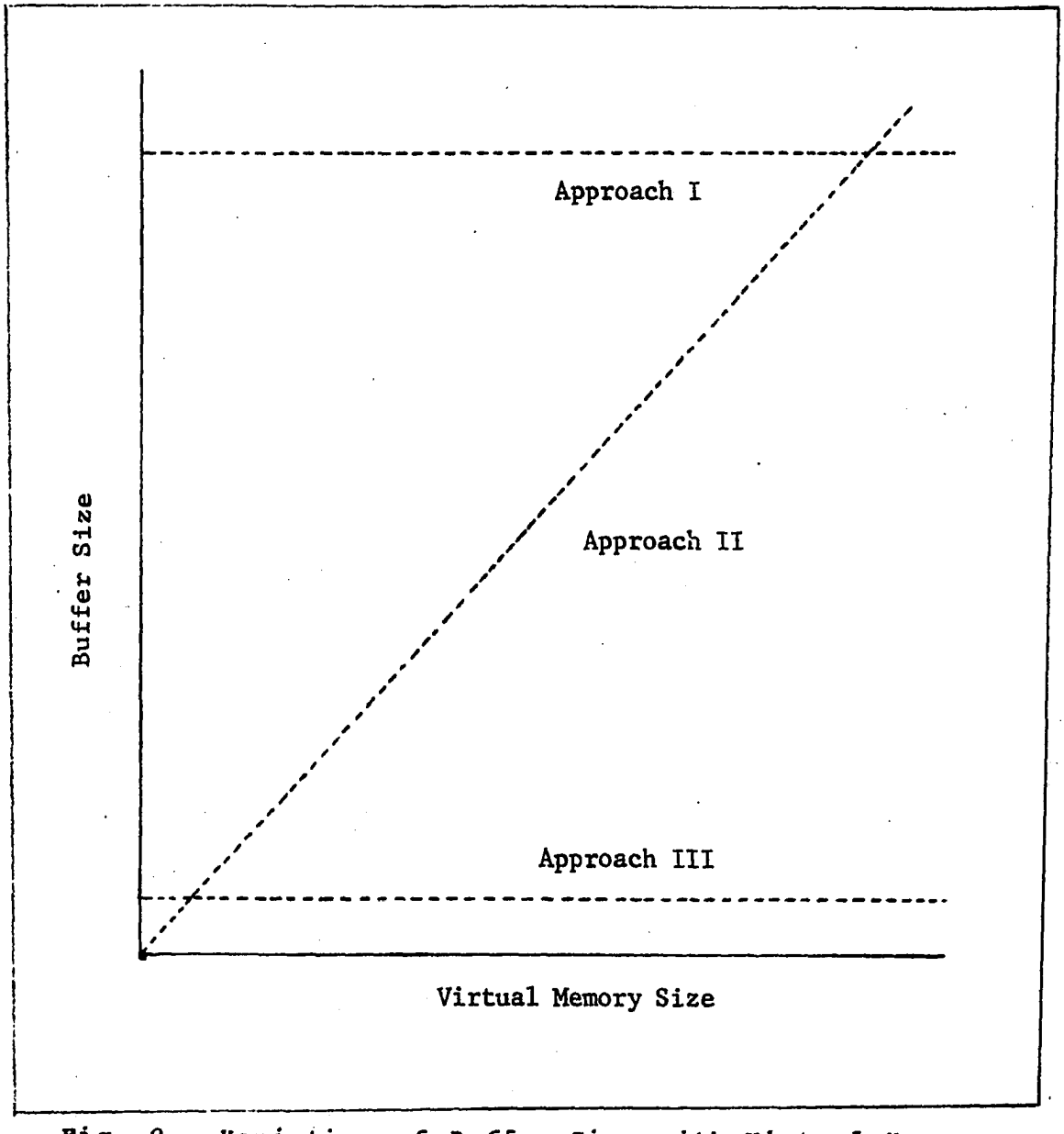

Fig. 8. Variation of Buffer Size with Virtual Memory

associativity seems to be the right approach for SYMBOL-IIR like computing structures (Fig. 8).

Physical and functional considerations

SYMBOL-IIR like computing structures consist of a network of multidedicated or non-homogeneous or special purpose processors. The whole system is organized as a network of master-slave processors, and each processor is indispensable to the system. A terminal needs the service of these dedicated processors, for various amounts of time, for the execution of its job. Hence, after deciding the size and associativity considerations, another parameter which seems to be very important is to decide how the partitioning of buffer space should be done, that is should buffer space be provided physically for different existing physical processors or should buffer space be provided for existing terminals or should the buffer space be just provided for the whole system.

Since a dedicated processor performs a dedicated function (that of either storing input programs, or translation, or execution, or system supervision, etc.), the assignment of buffer space to different processors in a dedicated processors environment indirectly segregates buffer space functionally and buffer space is automatically partitioned into a number of different homogeneous units.

However, if buffer space is allotted physically to either different terminals or for the whole system, it takes a different perspective. Now the buffer space is not segregated into different homogeneous units automatically, and if further partitioning of buffer space is to be done it could be done on a functional basis.

The idea of partitioning buffer space functionally is to treat the whole buffer space either as a homogeneous unit or a heterogeneous unit. The analogous approach of partitioning buffer space functionally in a conventional computer system might be that of segregating buffer space into "instruction buffer space" and "data buffer space". However, for SYMBOL-IIR like computing structures, the approach might be that of of partitioning buffer space into three parts--one part for source string (source buffer space), one for object string (object buffer space), and one for name-table and data (name-table buffer space). The reasoning behind this kind of partitioning is based on the fact that almost all the virtual pages of SYMBOL-IIR like computer structures are used for one of those three purposes.

Another approach of partitioning buffer space functionally is to have one to one correspondence between the buffer space and the main memory. Hence the whole buffer space might be thought of as partitioned into three parts--system and terminal header buffer, data group linking buffer

and data buffer respectively.

The idea of providing buffer is to simply provide frequently accessed data at a faster rate to the processors. Hence, data which are accessed infrequently (and data which could be accessed at a faster rate) need no buffering. Group link words are generally accessed only once for the whole group. Hence the access frequency of group link words is 1/w, where w is the number of words in the group. Hence as w decreases, the frequency of group link word accesses increases, and as w increases, the frequency of accessing group link word decreases. The justification for providing a buffer for data or group linking words is valid only after taking extensive traces for frequency of accesses to group link words. If the provision of a data link buffer does not improve the performance significantly (as far as the hit-ratio is concerned) and if it is relatively expensive then apparently there is not much justification in providing a buffer data link. More about this is discussed in chapter IV. Taking this approach the whole buffered virtual memory system of SYMBOL-IIR like computing structures would look as in (Fig. 9).

Fixed or variable partitioning

Another factor which is very important for efficient buffer management strategy is to decide how the buffer space should be partitioned among processors or the terminals or

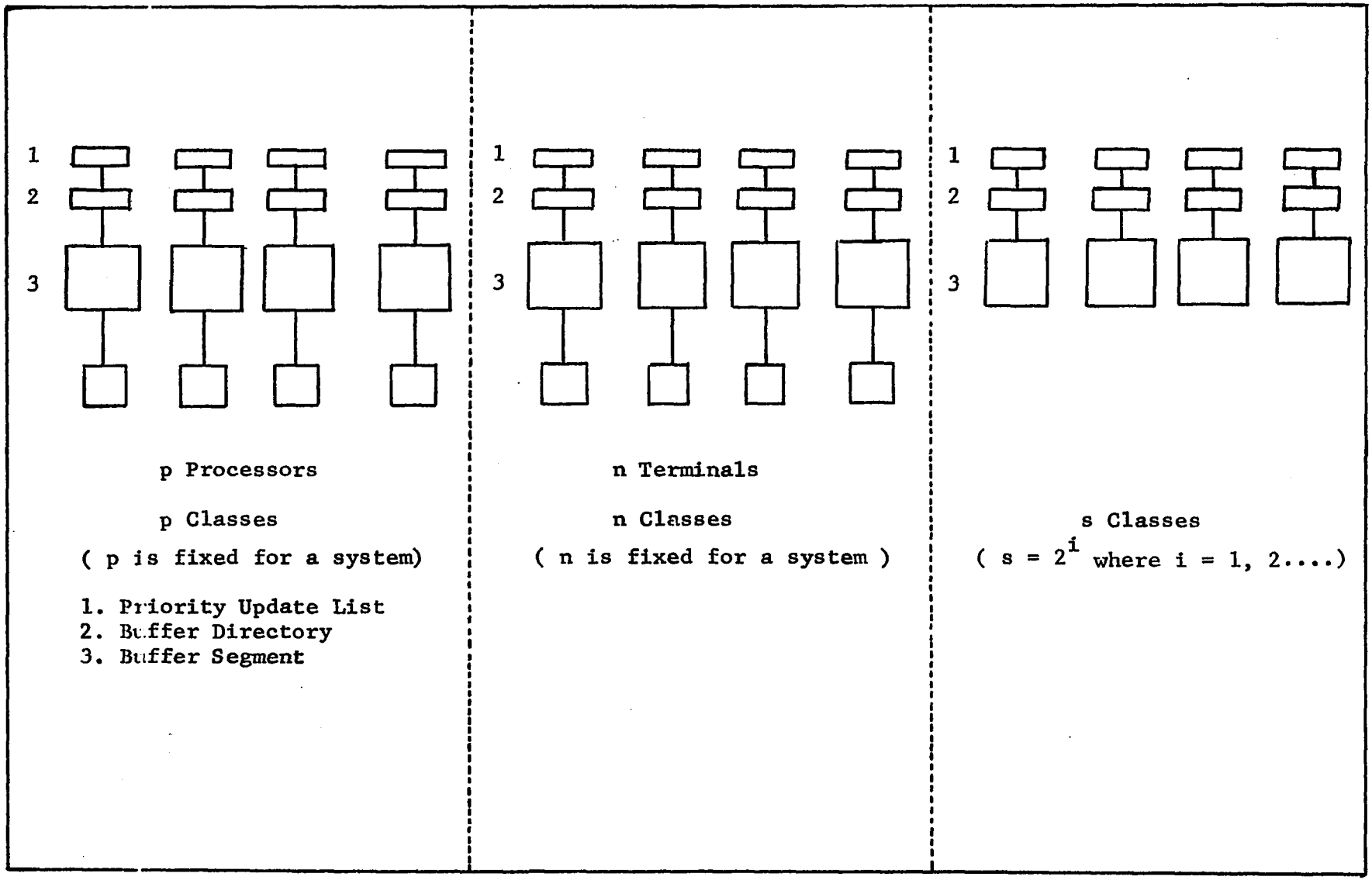

Fig. 9. Buffer Partitioning Strategies

for the whole system.

The overall hit-ratio obtained in a buffered virtual memory system partitioned among different classes is the combination of hit-ratios for each class. Hence, the principle behind partitioning of buffer space among various classes is to obtain maximum hit-ratio for the most frequently referenced class. The problem of deciding whether to partition buffer space equally among all the classes or not depends upon the frequency of reference of each class. As long as all the terminals of the system are given equal priority for system usage, nothing can be said about the load on the system by different terminals, and hence one easy approach of partitioning buffer space for terminals would be to partition the buffer space equally among all the terminals.

Though the same approach of fixed partitioning of buffer space might be used for the dedicated processors, the following factor should be kept in mind.

The idea of achieving multiprocessing by the use of multidedicated processors in a time-sharing environment is the efficient utilization of hardware resources and faster response time (36,37,38). One of the major presumptions behind this concept is that, in a full operational time-sharing environment, all of the processors would be fairly busy. However, as far as the processing of a job by different dedicated processors is concerned the duration of use or

the service time of each dedicated processor is not the same, that is all the processors are not needed for the same amount of time. The amount of service required of a dedicated processor is highly program dependent. Also, the type of addresses referenced by these dedicated processors vary. Some of these processors generate a sequential address reference, and some scatter their address referencees non-sequentially. Hence, the amount of processor service time along with its type of address reference should be another major factor in deciding whether the buffer space should be partitioned equally among all the dedicated processors or not. If the frequency of reference of different dedicated processors is about the same then equal partitioning of buffer space would seem to be the right approach. Otherwise, for obtaining better hit-ratio, the processor with the maximum (minimum) frequency of reference should have the largest (smallest) portion of the buffer space. Hence, the addressing pattern of different dedicated processors, their frequency of usage, and the amount of their service time should be some of the major factors in the decision of equal or unequal and fixed or variable partitioning of buffer space amongst different processors.

Buffer address translation strategies

Another factor which is also of some importance in the design of a buffered virtual memory system is the efficient

design of "address translation strategies".

The most important goals of an efficient address translation scheme are that it

1. Should try to reduce the delay associated with address transformation with minimum cost and

2. Should be as simple as possible.

A virtual memory system tends to give the illusion of an infinite physical storage space to the users even though the physical storage space is limited. Hence, there has to be an address translation strategy for mapping the virtual space into physical space. In a simple paged virtual memory system this is accomplished by the provision of an associative memory, which serves as a dynamic map table and keeps track of the association of a physical page to a virtual page. Every reference to memory is now accompanied by an address transformation, to locate its physical location.

In a buffered virtual memory system, buffering adds another level to the levels of memories. Thus there has to be another map for mapping buffer locations into physical locations. Hence intuitively it seems that, in a buffered virtual memory system, every memory reference would involve two address transformations--one from the virtual space to main memory space and the other one from main memory space to the buffer space. However the control can be so designed that two address transformations are needed only when an

auxiliary to main or main to cache transfer is performed and not when the data is already in the cache. The extra price one has to pay for saving one extra transformation is in the increase in size of the buffer directory. The buffer directory now has to have information regarding the association of buffer location with the main memory as well as the auxiliary memory.

For SYMBOL-IIR like computing structures without the buffered memory, the address translation could be done as shown in Fig. 10.

However with buffering, the address translation is not so simple.

In SYMBOL-IIR like computing structures, with any memory request, we have the following information:

1. The virtual address which involves
   a). The virtual page address
   b). The group address and
   c). The word address.
2. The processor which is requesting
3. The terminal number for which the processor is working and
4. The page list number (the purpose for which the request is being made).

Based on this information, mapping techniques for buffered virtual memory system of SYMBOL-IIR like computing

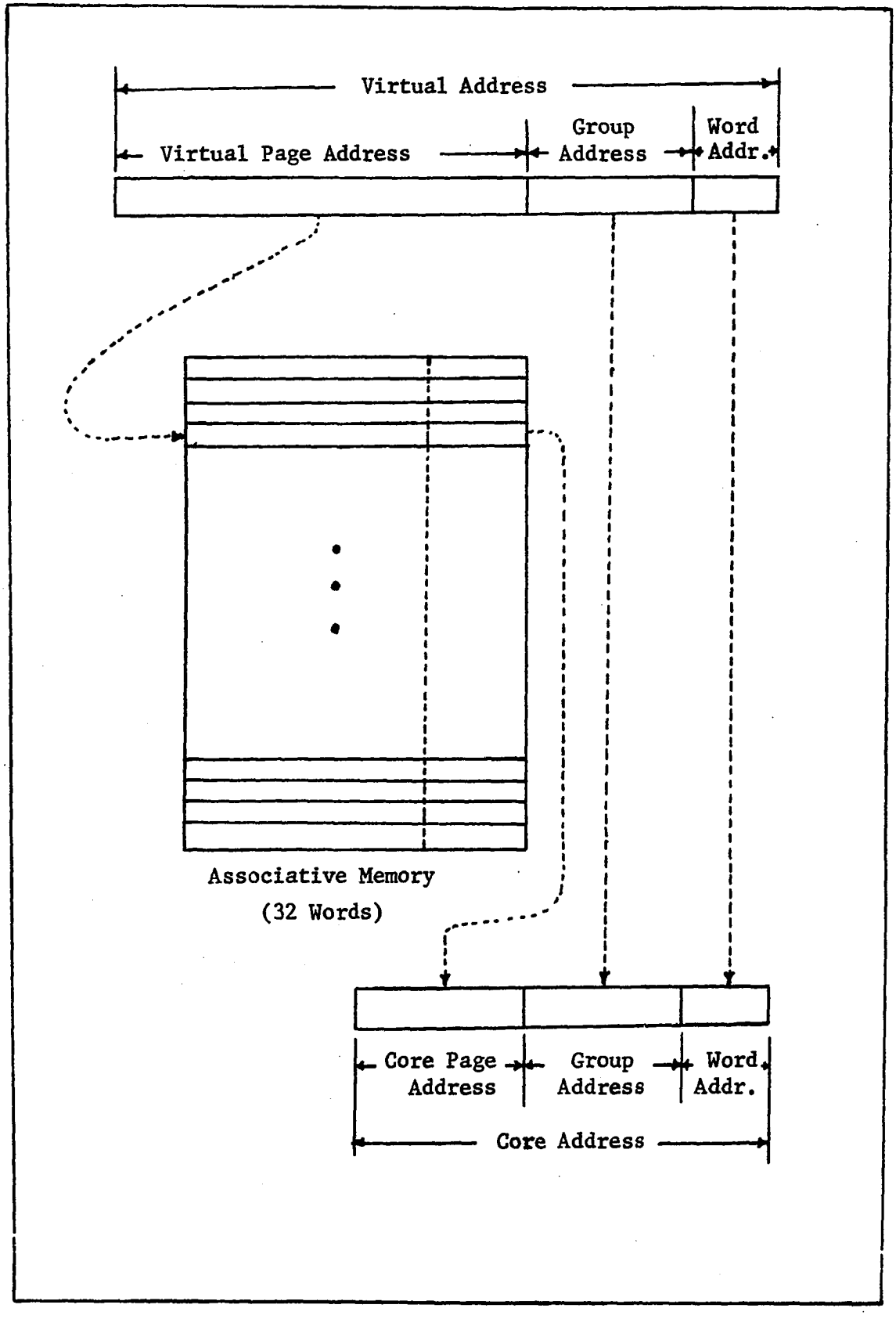

Fig. 10. Address Translation of SYMBOL-II

structures could be divided into three cases depending on how the buffer space is going to be partitioned, that is whether it is going to be divided into various classes on terminal number basis, or processors basis or on the whole system basis. Each of these three cases could be further divided into two schemes based on whether each reference to memory has to involve one or two address transformations.

These three cases are illustrated in Fig. 11 to Fig. 14.

When the whole buffer space is allotted to the system and if it is to be partitioned into several classes, logically adjacent pages are thought to reside in contiguous classes. Hence for this case, the lower order bits of a virtual page represent a class number. But when the buffer space is to be partitioned among various terminals it takes a different perspective.

In SYMBOL-IIR like computing structures, any terminal could be owner of several logically or physically contiguous pages, hence representing the class number by the lower order bits of the virtual page address might result in the pages of one terminal belonging to two different classes. In this case the terminal number itself (rather than the lower order bits of virtual page address) represent the class number, and it is used as the key for searching the buffer directory.

The same approach could be used also for the processors. However partitioning of buffer space into classes based on

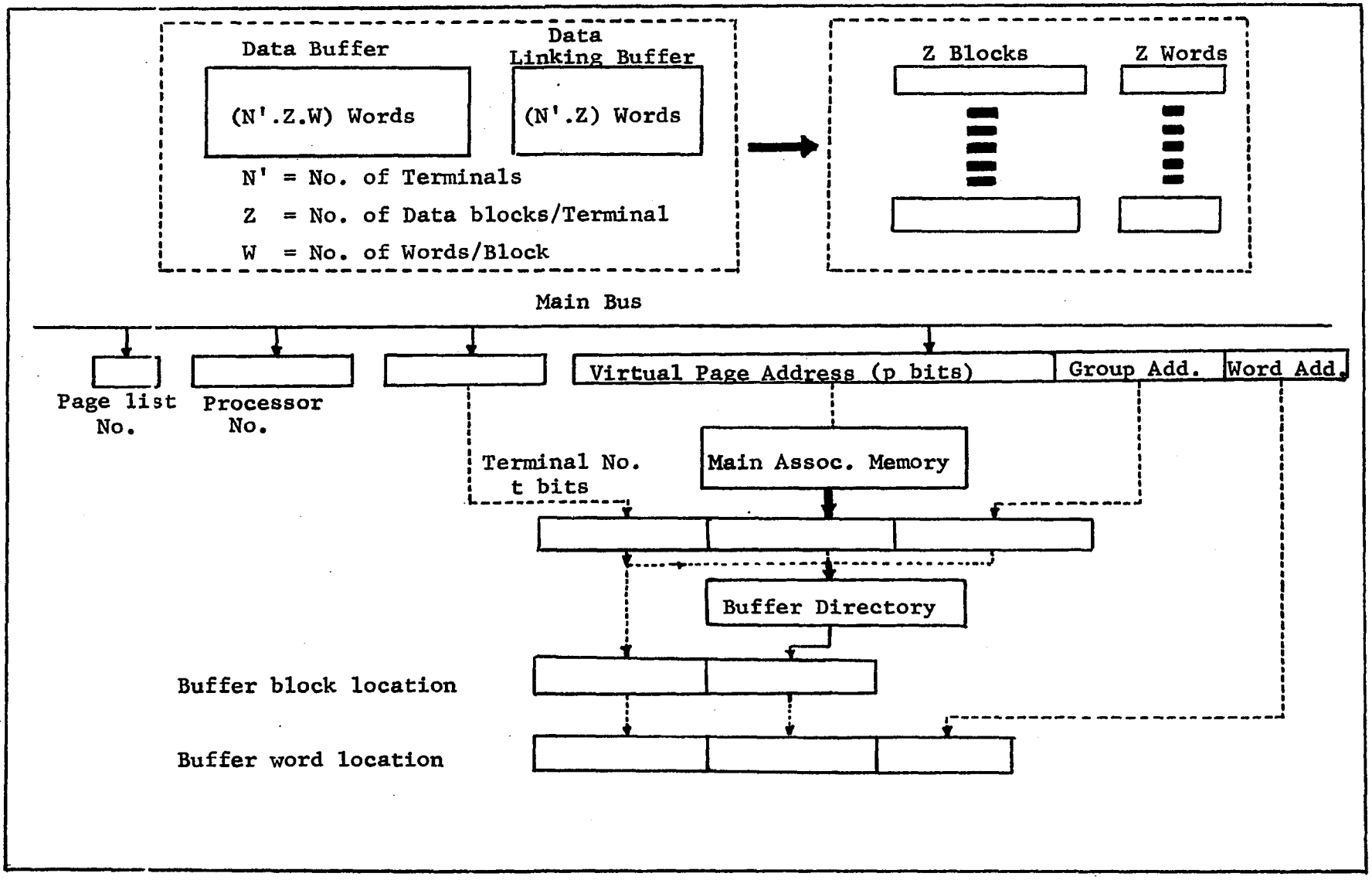

Fig. 11. Address Translation of SYMBOL-IIR like Computing Structures

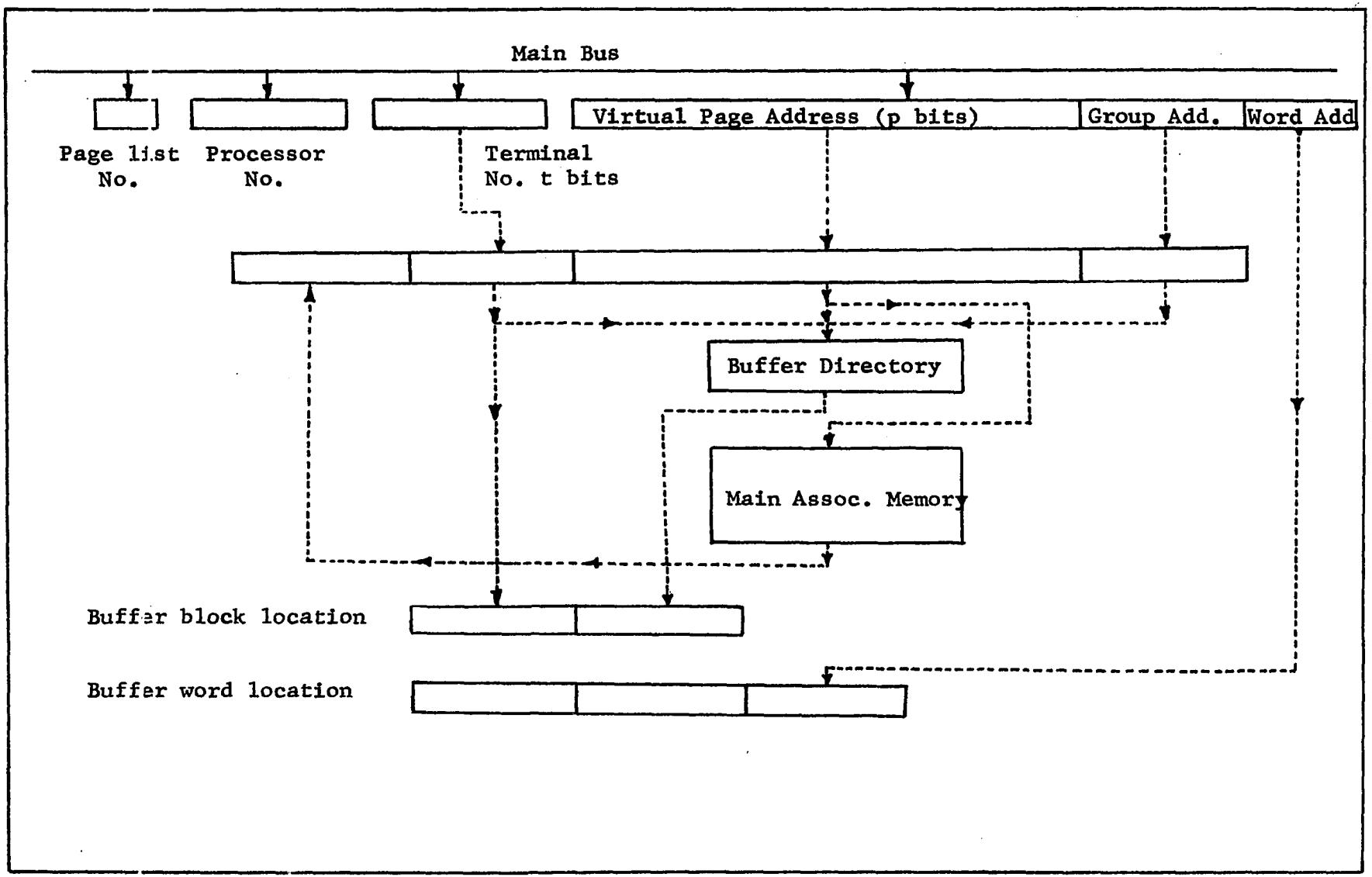

Fig. 12. Address Translation of SYMBOL-IIR like Computing Structures

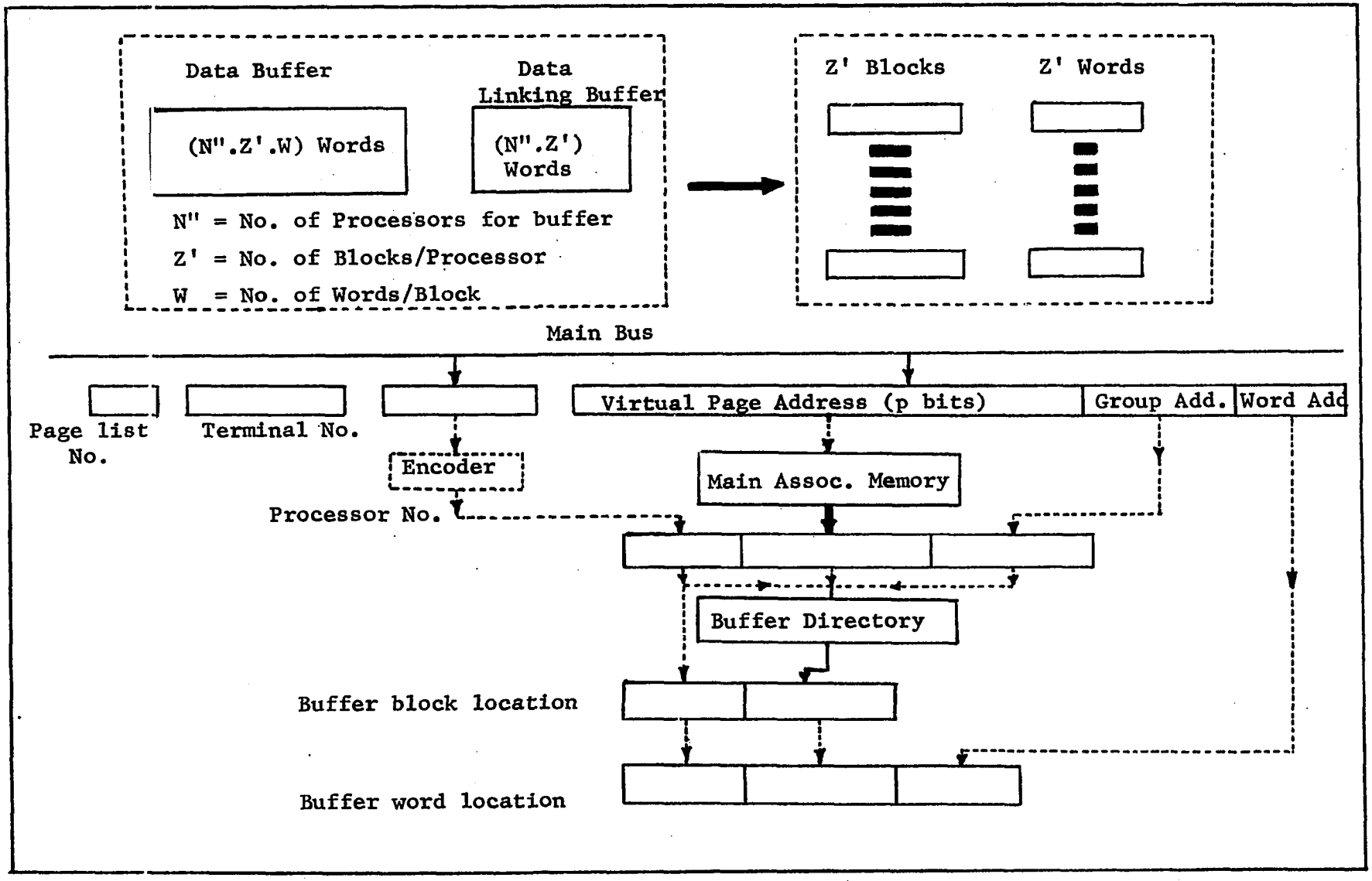

Fig. 13. Address Translation of SYMBOL-IIR like Computing Structures

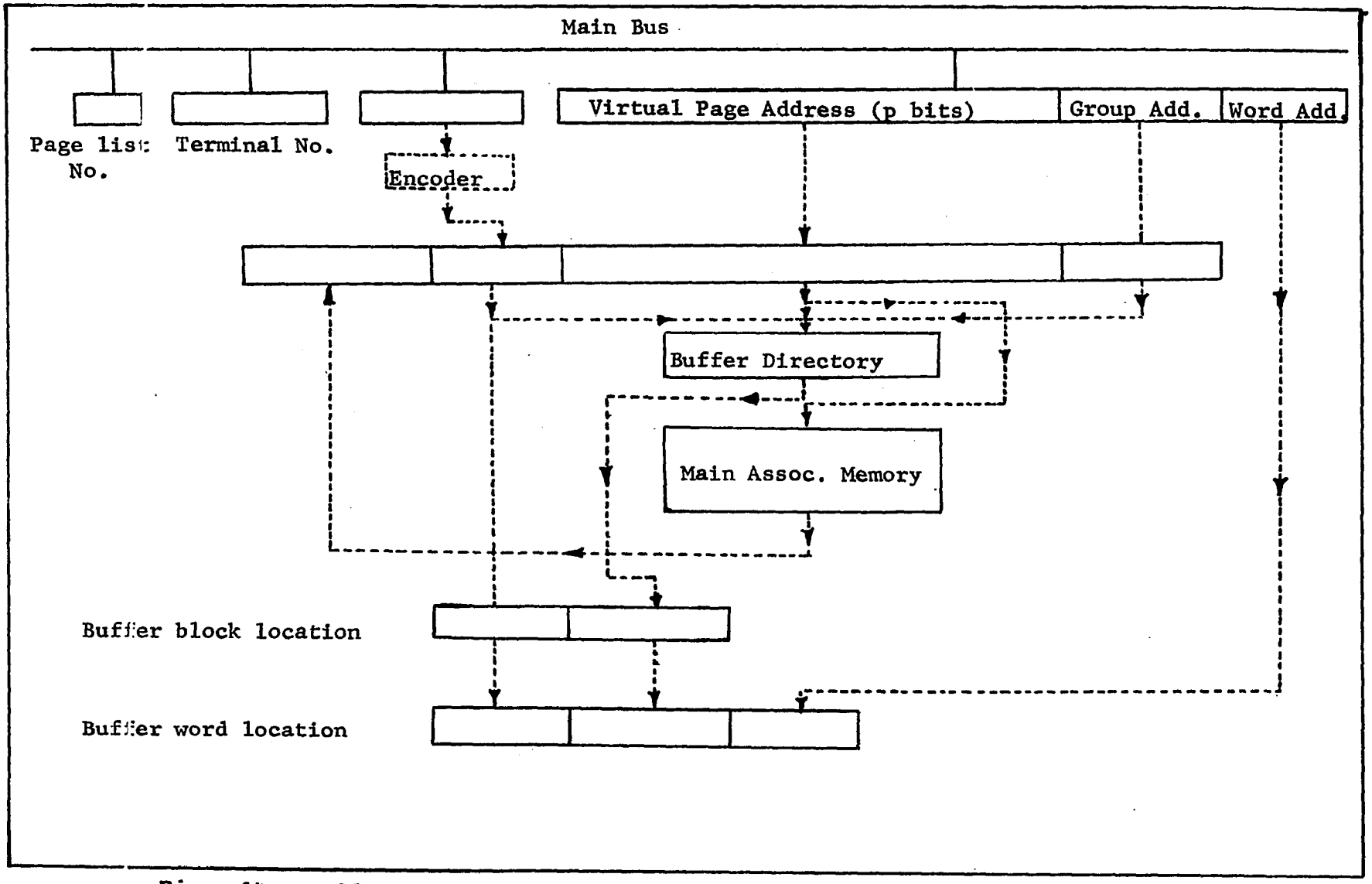

Fig. 14. Address Translation of SYMBOL-IIR like Computing Structures

processors does not truly partition buffer space among various classes. because they share each others information. But as far as the searching of the buffer directory is concerned the processor number along with the virtual address could be used as a key to search the whole directory.

When the whole buffer space is allotted to the system, and if the partitioning of the buffer into various classes is desired, one easy approach might be of allotting equal buffer space for each class.

In the schemes where one address transformation is saved, the buffer directory (instead of the main associative memory) is searched first. And if the required block is not found in the buffer, the virtual page address is used as a key to search main memory to see if the required page is in the main memory or not. Hence now the buffer directory has to have the association of buffer location with both the main memory and the auxiliary memory. This results in the bigger size of the buffer directory. Hence, there has to be an address translation strategy for mapping the virtual space into physical space.

Buffer space replacement considerations

One of the very important parameters which affects the overall effective design of the buffered memory for SYMBOL-IIR like computing structures is the design of the buffer space replacement strategies. Even though the problem of

"thrashing" is not so severe in buffered memory (because of speed difference of about 10 and not 100 or more between main memory and buffer (12,13), the whole idea of improving the performance by the use of a buffer memory would lose its importance if, because of a bad space replacement strategy, a very poor hit-ratio were obtained. Hence, while designing an efficient buffer space replacement strategy the following two factors should be kept in mind:

1. The algorithm should try to maximize the "block residency time" in the buffer, that is the number of distinct blocks encountered between two successive block swaps should not be more than the buffer capacity. This principle also would tend to increase the "activity" of the buffer (42,43) and

2. The algorithm should tend to replace a block from the buffer whose probability of reference in near future is minimum. This also will measure indirectly the "buffer inactivity time" (42,43).

Even though the buffer replacement strategy has not been found to be of much importance in conventional machines, it seems to be important for a computing structure like SYMBOL-IIR.

It is as important as the buffer space allocation strategy, for the same reason: the buffer has to be as small as possible for cost considerations and at the same time it

should tend to achieve very high hit-ratio.

The design of the buffer space replacement strategy consists of designing of following two factors:

1. Designing of "replacement policies", and
2. Designing of "replacement mechanisms".

Replacement policies are the policies for deciding how the replacement is to be handled, that is whether a "local policy" or a "global policy" should be used, and mechanisms are the means of implementing a particular rule or scheme. The rules might be one of the followings:

1. FIFO (first in first out)
2. LIFO (last in first out)
3. FINUFO (first in not used first out)
4. LINUFO (last in not used first out)
5. LRU (least recently used)
6. MRU (most recently used)
7. LFU (least frequently used)
8. MFU (most frequently used)
9. Biased Replacement Rules etc. (6,7).

The local replacement policy is the policy in which a processor/terminal replaces data from its own subset of buffer memory space to make room for newly demanded data. Whereas, a global policy is the one in which a terminal or processor can replace any other's data. One important aspect which has to be kept in mind though is that, for SYMBOL-IIR

like computing structures there is "no-sharing" of information between different terminals. However, different dedicated processors share each other's information. Allotting fixed and unequal amount of buffer space for either terminals or processors tends to divide buffer space into a fixed number of classes. As far as partitioning of buffer space into various classes for terminals is concerned it is all right, because they do not share each other's information. However as far as the partitioning of buffer space for various processors is concerned, it creates a problem. Because processors have to share each other's information, there has to be a provision for allowing the processors to share each other's buffer. Hence, the technique for allocating and replacing buffer space for processors is as follows:

1. When certain demanded information by some processor is not in its own buffer, then all the buffer directory should be searched in order to find the possibility that it might belong to some other processor's buffer space. If it is found in any other processor's buffer directory, then the required data should be fetched from that processor's buffer space.

2. However, if the required information is not found in the whole buffer, then a certain block has to be replaced from the buffer. At this point, a "local" or "global" policy could be used. As far as the processors are concerned, a

local policy seems to be better than the global, because, in using a global policy a processor might replace a block (belonging to some other processor's buffer) which could seem to be of least importance to it at that point, whereas it could be quite important to the processor from whose buffer space the block is being replaced.

Hence, as far as processors are concerned, the principle of "global searching with local replacing" seems to yield better results intuitively.

However, as far as terminals are concerned, the picture looks quite different.

1. When information is needed by a certain terminal, then the directory of that terminal only is searched. If it is not found in that class, then a particular block has to be replaced.

2. Now the principle of either "local" or "global" replacement strategy could be used. Since the terminals do not share each other's information the "local" policy seems to yield better results than the "global" one. Hence, as far as terminals are concerned the principle of "local searching with local replacing" should be used.

As far as partitioning of buffer space into different terminals or processors is concerned the buffer space replacement strategy could be investigated a little further.

The buffer space for a terminal could be pictured as a collection of three homogeneous sub-units. These three sub-units are the buffer for source, object and name-table and data. Hence, the "local" replacement policy, for a terminal, could be further divided into two sub-policies as follows:

1. Sub-local policy or
2. Sub-global policy.

Sub-local policy is the policy of replacing a particular block, belonging to the same sub-class as the demanded block i.e., source, object or name-table and data list, from its own assigned buffer space only. Sub-global policy is the policy of replacing any block belonging to its own assigned buffer space for a new demanded block. So, for the sub-local policy, a demand for a block belonging to the source list could only replace a block belonging to the source list only from the buffer, whereas in the sub-global policy, a demand for a block belonging to say source list could replace a block belonging to other lists.

As far as the partitioning of buffer space for the processors is concerned it might appear to be a collection of different homogeneous units. However, because the processors share each others information, the buffer of certain processors might result in as heterogeneous units. So the similar principle of sub-local and sub-global replacement policy can be applied to the processors also.

On the whole, the partitioning of buffer space in SYMBOL-IIR like computing structures can be pictured as in Fig. 15.

Comments on Buffered System Organization

Before carrying out any simulation analysis of a system, it has to be modelled. And to model a system various important parameters have to be decided. This chapter serves as a basis for models of buffered memory systems of SYMBOL-IIR like computing structures and analyses various important parameters which can influence tremendously the simulation. This provides an insight into the design of buffered memory systems of SYMBOL-IIR like computing structures, and based on these factors the simulation experiment is carried out and is discussed in the next chapter. Also one of the main purposes of this chapter was to study the effect of architectural organization and its philosophies on the design and management of its buffered memory systems.

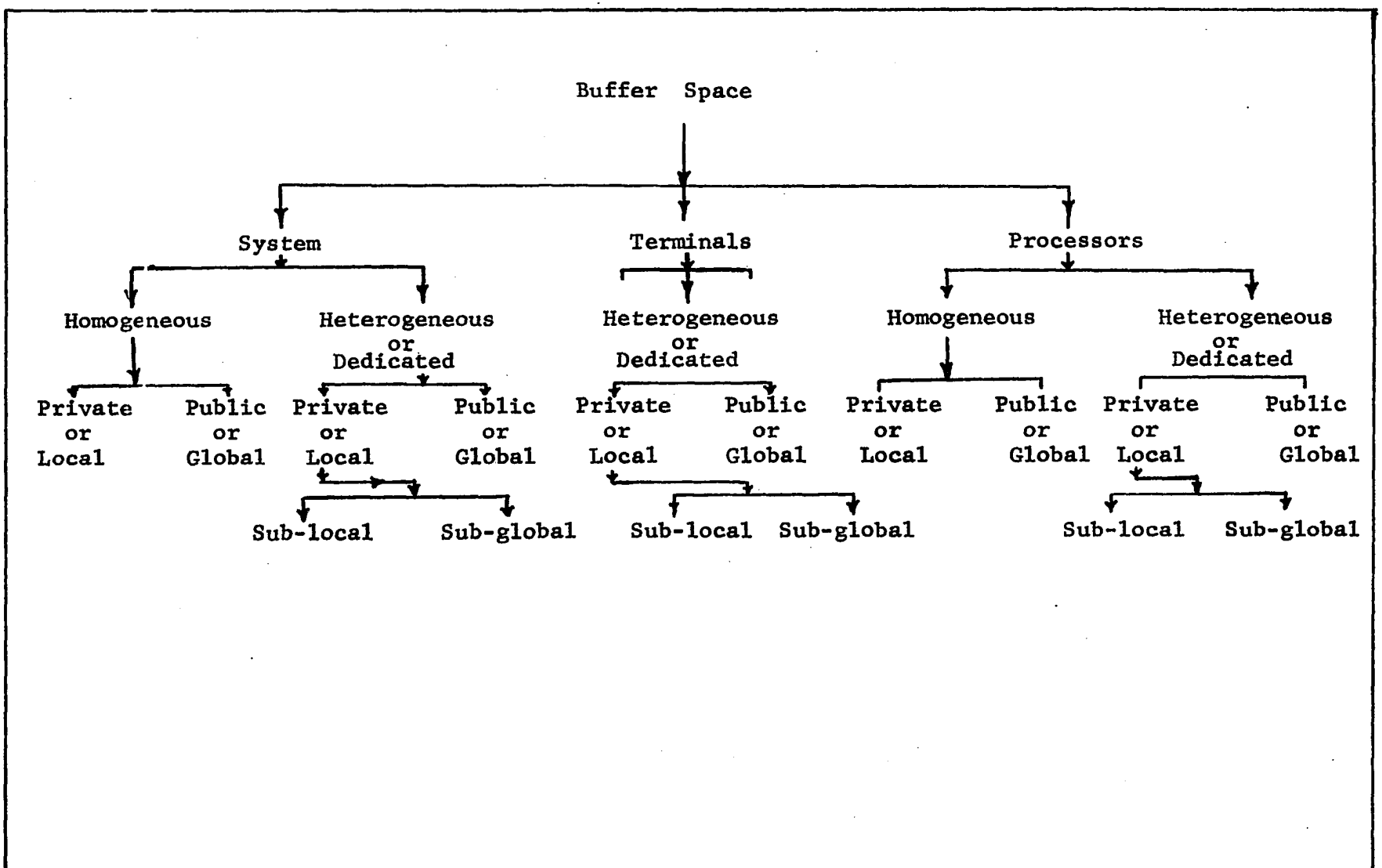

Fig. 15. Buffer Space Allocation for SYMBOL-IIR like Computing Structures

CHAPTER IV. EXPERIMENTAL ANALYSIS AND RESULTS

Introduction

The planning of the experiment for determining the performance of buffered memory systems for SYMBOL-IIR like computing structures essentially consists of three things:

1. The selection of the "controlling" or "experimental" factors.
2. The design of the experiment itself and
3. The determination of the factors to be analyzed and their figures of merit (42,43).

Controlling Factors

Controlling factors are the factors which can be controlled by the experimenter for controlling the process under investigation. These are essentially the factors which control the results of the performance analysis and based on which a feeling of the performance of the system is obtained. For a particular experiment the number of these controlling factors could be quite large. However as the investigation of a process is very much limited by time and economic considerations one must try to select only the factors which seem to be potentially most important for the particular experiment. For analyzing the performance of buffered memory systems for SYMBOL-IIR like computing structures the controlling factors which are chosen for investigation are:

1. The users environments
2. The buffer partitioning algorithms
2. The replacement algorithms and
4. The buffer and block sizes.

As we can see each of these controlling factors can be varied. For these various controlling factors the various alternatives that are considered are as follows:

Users environments

One of the important parameters which determines the comparative suitability of one hierarchical system over another is the nature of users environments. One of the important questions that has to be answered for determining the applicability of a buffered memory system is how does the effectiveness of the cache vary from one program to another, that is from one users environment to another. Hence to evaluate the performance of buffered memory systems, one phase of the task is to test performance in several types of users environments, and the second phase of the task which is perhaps more difficult is to identify the worse conditions where cache design may be a poor choice.

This factor can be divided into three categories:

1. Scientific environments
2. Commercial environments or
3. General purpose environments.

To reduce the amount of computation time and because it was a

feasibility investigation of buffering SYMBOL-IIR like computing structures, the investigation was restricted to only small scientific users environments.

Buffer partitioning algorithms

This factor is divided into three further subdivisions for the purpose of the investigation:

1. Buffer partitioned for processors
2. Buffer partitioned for terminals and
3. Buffer partitioned for system.

Replacement algorithms

1. Least recently used (LRU)
2. Most recently used (MRU)
3. First In first out (FIFO)
4. Last In first out (LIFO)
5. Least frequently used (LFU)
6. Most frequently used (MFU)

Each of these replacement algorithms were further divided into two subfactors--private (local) or public (global), depending on where they were applicable.

Buffer and block sizes

This factor can be varied over a wide range of values, however for our experimental purposes the block size was fixed to be 8 words and the buffer size was varied from 16 to 512 words.

Design of the Experiment

One of the factors strongly affecting the performance of a buffered memory system for SYMBOL-IIR like computing structures is the nature of the trace of memory addresses of both the instructions and the data or operands, referenced by various processors for the various terminals. One of the most common ways for evaluating the performance is by a simulation technique. The main idea behind simulation is to generate sequences of memory requests resembling the real environment and to analyze them for the real environment. There are two approaches for carrying out this simulation:

1. Instrumentation technique and
2. Synthetic technique.

Instrumentation techniques are the techniques for generating sequences of memory requests with the help of instruments when various test programs are really running in the system. The dynamic traces of the addresses are recorded by various instruments. Instrumentation techniques have been very popular because it is possible to record a dynamic address sequence of various processors with sufficient accuracy, that is memory addresses as they are generated are recorded. The major disadvantage of this technique however is that this is very inflexible, and is not machine independent. The need to evaluate a different buffered memory system with a different kind of environment might need a minor or major

modification of the previous instrumentation technique.

Synthetic techniques are the techniques for evaluating a completely different environment from the existing one by a study of statistical characteristics of the traces for the present environment (1).

However, most of the studies that have been taken so far to evaluate the performance of various conventional buffered memory systems have used various instrumentation techniques. The usual approach followed in these studies has been to record memory address traces of some representative test programs and then feed them into a simulator which monitors the loading and unloading, into and out of, cache memory in accordance with the policies and mechanisms of the real environment and computes the probability of either success or failure. Hence the instrumentation technique was adapted for collecting the data dynamically.

One of the main goals of the experiment design should be to collect data dynamically without affecting the dynamics of the system.

As far as the simulation analysis of the buffered system is concerned, the following information was thought to be enough:

1. The processor number
2. The terminal number
3. The page list number (the purpose for which a page

was being used)

4. Virtual page address and

5. Group address

An interface unit was built to record this information dynamically. The interface unit consisted of a Fabri-Tek core and a Kennedy tape unit. These addresses were collected directly from the "local bus" of the Memory Controller and were stored in a Fabri-Tek Core Memory. There was also the provision for storing the data in the tape unit. To analyze this voluminous data there were two alternatives:

1. Collect all the data on a tape and then analyze it with programs in the IBM System, because the SYMBOL-IIR does not have a tape interface capability or

2. Provide some means to transfer collected data from the Fabri-Tek core to the SYMBOL-IIR disc and then analyze it.

The SYMBOL-IIR being available for the research purpose, the second alternative was chosen. Besides being able to analyze the data at times of convenience, this also turned out to be very economical. The programs which were chosen for analysis usually generated about 15K to 24K addresses. But, the Febri-Tek core used for collecting the addresses had the capability of storing only 8K of addresses at a time. Hence, there had to be the provision for transferring the collected data (8K at a time) and resuming the program from

the point where it was interrupted last, conveniently without losing any information. After all the addresses for a particular program were transferred then the simulation analysis program was run.

Factors to be Analyzed and their Figures of Merit

For the design and management of buffered memory systems for SYMBOL-IIR like computing structures, the following factors are considered to be very important:

1. Hit-ratio
2. Processor utilization and its address reference pattern
3. Processor traffic rate and
4. Block and buffer utilization.

Hit-ratio

One of the most important factors for the design of buffered memory systems is the hit-ratio. Hit-ratio is the ratio of number of successes of memory requests to the buffer divided by the total number of memory requests. Miss ratio can be defined as the total number of misses divided by the total number of memory requests. So miss ratio is equal to (1 - hit-ratio).

Hence the figure of merit for hit-ratio is that it should be as close to unity as possible and the miss ratio should be as small as possible.

Processor utilization and its address reference pattern

One of the other important factors to be considered for buffering a multidedicated processors time-sharing computing system is the percentage utilization of each dedicated processor. The basic idea behind the purpose of achieving multiprocessing with the help of multidedicated processors is the belief that under full load conditions, all the processors would be quite busy. However, since these processors are dedicated, the amount of service needed for a particular dedicated processor is highly program dependent. The amount of buffer to be allotted to a particular processor is affected by the amount of that particular processor's activity or utilization.

Besides the amount of processor utilization, the other factor which affects significantly the amount of buffer for a particular processor is the address reference pattern of that particular processor. The dedicated processors of SYMBOL-IIR are designed to do dedicated tasks--and since these dedicated tasks are quite different, the addressing pattern of these processors are quite different too. Some of these processors have highly sequential address reference patterns and some have highly random address patterns.

Processor traffic rate

Another important factor to be considered is the traffic rate for different dedicated processors. The traffic rate

for each dedicated processor is defined as the ratio of the traffic between the main memory and the dedicated buffer to the traffic between the dedicated buffer and the dedicated processor. It is simply the number of words transferred to that dedicated buffer from the main memory divided by the total number of accesses of memory requests by that particular processor. One of the main goals of buffering such a multidedicated processor system is to minimize the traffic between the main memory and the buffer and maximize the traffic between the particular processor and its dedicated buffer. A higher traffic rate would indicate that most of the blocks which are being brought to that particular buffer are not being used. Hence the figure of merit for traffic rate is to be as low as possible.

Block and buffer utilization

The last but not the least important factors are the block and buffer utilization. Block utilization is the average number of words of the particular block active between two successive block swaps, where as the buffer utilization is the average number of blocks of a particular buffer active between two successive block swaps. So, the block utilization is the number of words of a particular block referenced at least once between two successive block swaps and the buffer utilization is the number of distinct blocks of the buffer referenced at least once between two successive block

swaps.

Block utilization illustrates the usefulness of extra words in a block and buffer utilization illustrates the usefulness of extra blocks in the buffer.

Experimental Results

Variation of hit-ratio with replacement algorithms

An analysis was carried out on one set of data to see the variation of hit-ratio with different replacement algorithms. Various replacement algorithms that were considered are LRU, MRU, LFU, MFU, FIFO and LIFO. It is observed from the Fig. 16 that, like conventional computer systems, hit-ratio does not vary drastically with different buffer replacement algorithms and LRU tends to give the best hit-ratio. Because of this it was decided to run all subsequent simulation analysis for LRU algorithms only.

Buffer allotted to the processors

Percentage of activity of different processors As mentioned above, the percentage of activity of a particular processor for the solution of a program is highly program dependent. However, for small scientific programs, the activity of these different processors are almost constant.

Fig. 17 illustrates the percentage of total memory address request of different processors. We see that the Memory Reclamation Processor (MR), the processor which works in the background mode and has the lowest memory priority, has

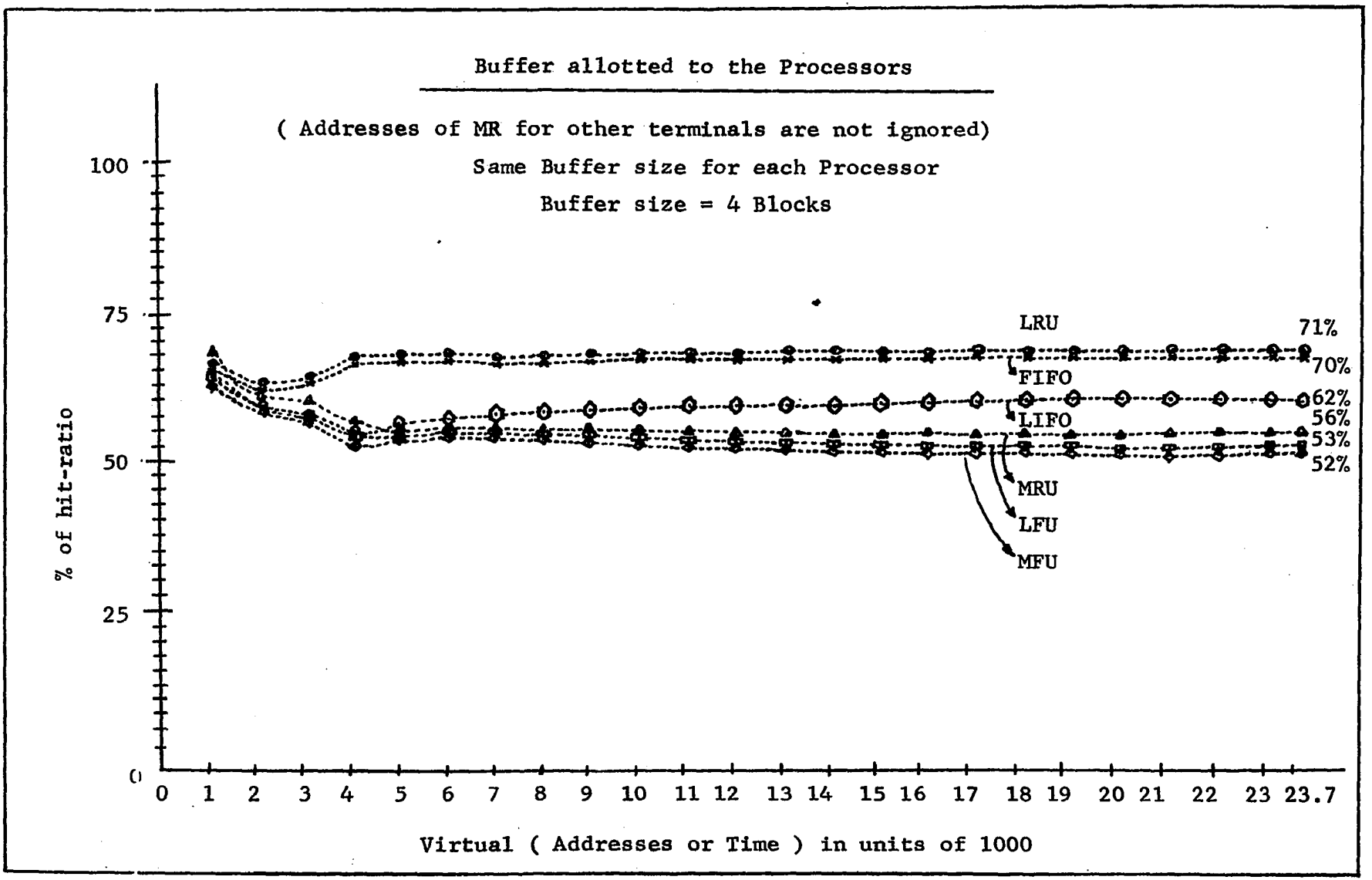

Fig. 16. Variation of Hit-Ratio with Replacement Algorithms

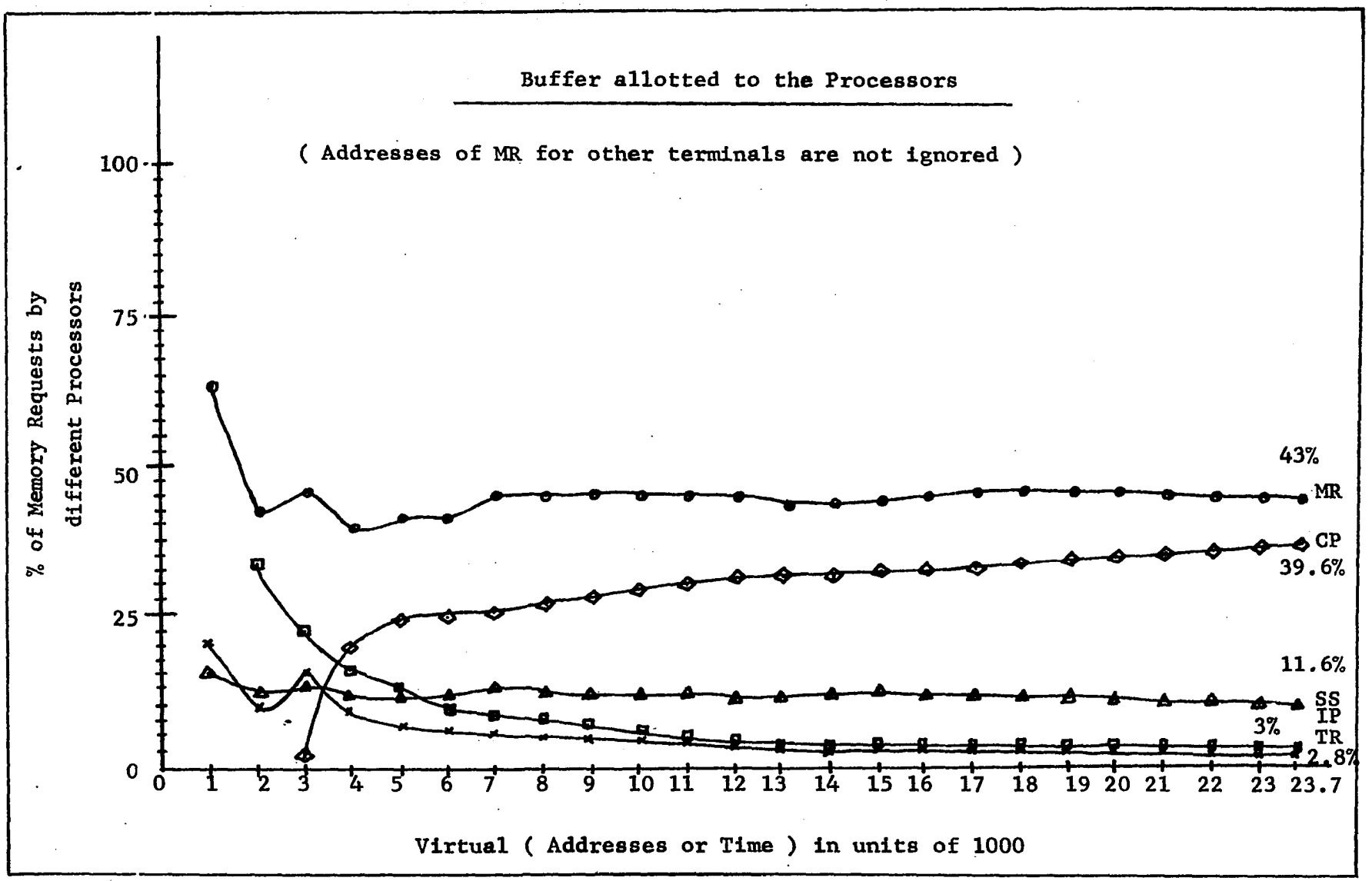

Fig. 17. Variation of Memory Access % of different Processors

the maximum percentage of memory accesses. This is because of certain design features of the system. For memory group reclamation, the MR polls all the terminals 4 times--twice for the page list 2 and once for page list 1 and 3 to check to see if it has any work to do for that particular terminal. This constitutes a substantial percentage of the total memory requests. Also we see that this processor is active almost from the beginning of the program to the end of execution.

The processor having the next highest memory access, at the end of execution, is the Central Processor (CP). At the beginning of execution the Central Processor has a very insignificant portion of total memory requests. However, at the end of program execution, the CP access percentage is quite significant.

The System Supervisor (SS) constitutes about 12% of the total memory accesses. Also, the SS is active from the beginning of the program to the end of the execution.

The Translator (TR) translates the program in a fixed amount of time and after that there is no need for the Translator for that program until the end of execution of that program. Hence after the translation of the program is over, the number of accesses of TR remains equal to what it was when it just finished translating. Hence plotting the percentage of access of memory requests by TR in an accumulative

basis results in a consistent declining curve for the TR. The activity of the TR starts only when the program has been loaded and a run command has been encountered.

The Input/Output Processor (IP) activity is highest during the loading of the program. The activity also increases if there are quite a few input/output statements in the program. Looking at the activity of the IP we see that its activity starts almost from the very beginning and then it starts declining.

The amount of buffer allotted to a particular processor for a reasonable hit-ratio is a function of the amount of the activity of the particular processor. However, besides the amount of the activity of the particular processor, another factor which affects the hit-ratio is the address reference pattern of the particular processor.

To investigate this, hit-ratio was computed for a large variation of buffer sizes for all different processors. Since the principle of global searching with local replacing is used for the processor, the amount of buffer allotted to a particular processor affects the hit-ratio for the other processors.

Each processor contributes to the over all hit-ratio of the system and the degradation in performance of hit-ratio for any particular processor can result in over all poor performance. Hence one of the basic design goals for achieving

a high hit-ratio for multidedicated buffering for a multidedicated processor system is that the individual buffer for each processor is adequate.

Memory Reclaimer (MR) Fig. 18 illustrates the variation of miss ratio of MR with the number of blocks over the total execution of the program. We see that the miss ratio for MR is quite high until the blocks allotted to its buffer are more than 16 and then for 16 blocks the miss ratio dramatically decreases to about 6.3%--yielding a hit-ratio of 93.7%. The reason for needing about 16 blocks for the MR before any increase in performance is observed is its constant polling of 15 other terminals--which may or may not be active at the time of collection of the data. Hence to investigate the number of blocks needed for the MR for the service of the active terminal, the addresses corresponding to other terminals were ignored. Ignoring these address requests we see that just 4 blocks for MR reduces its miss ratio to about 11%--a 3 times improvement in miss ratio (Fig. 20).

Also without stripping the addresses corresponding to other terminals, MR had about 43% of total memory activity (Fig.17). But after stripping these requests the % of MR memory access decreases to about 15% (Fig. 19). Also after stripping these requests, an allocation of 2 blocks to the MR buffer results in better hit-ratio than that of 12 blocks for MR for the unstripped case (Fig. 20).

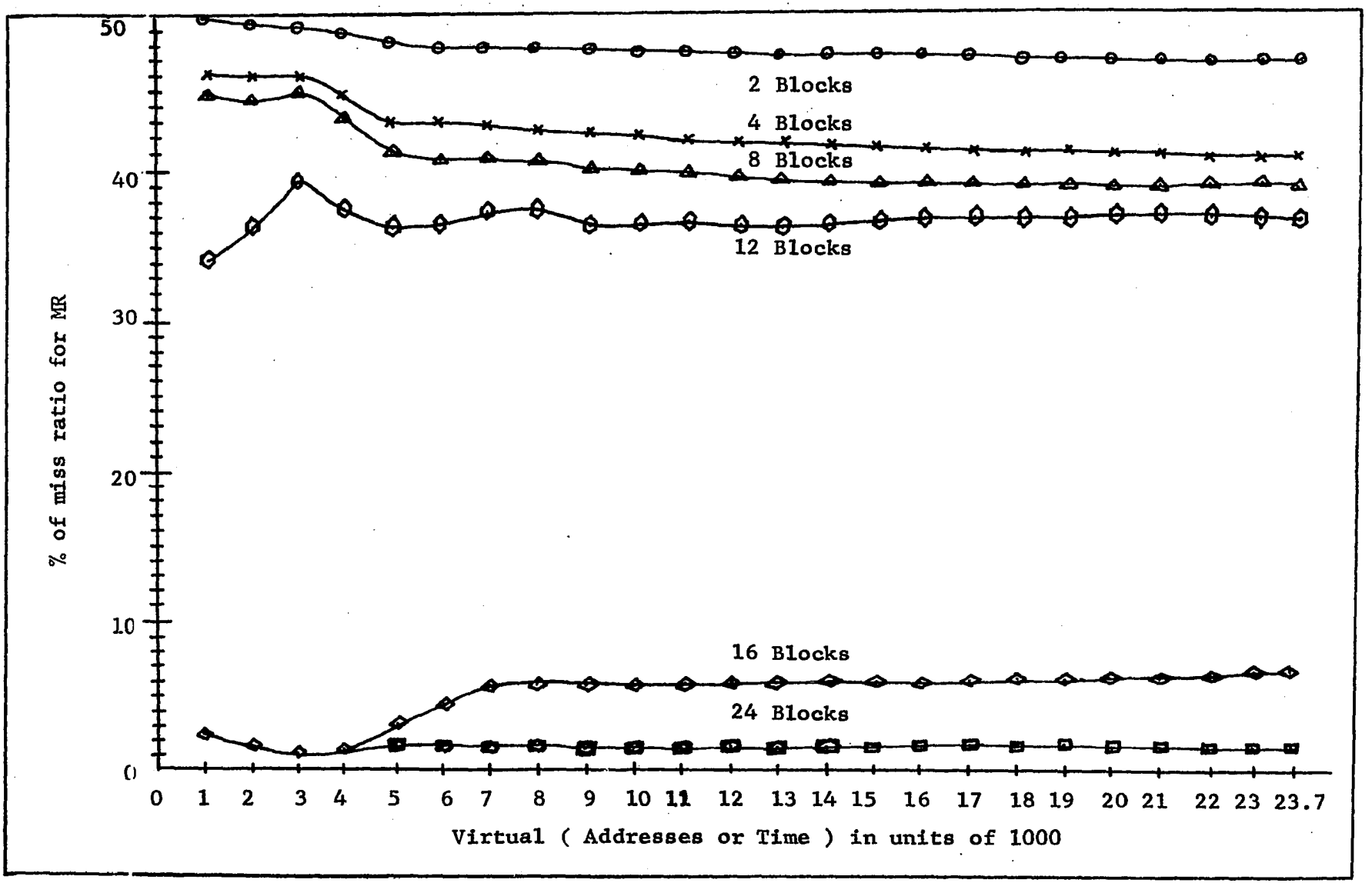

Fig. 18. Variation of miss ratio for MR

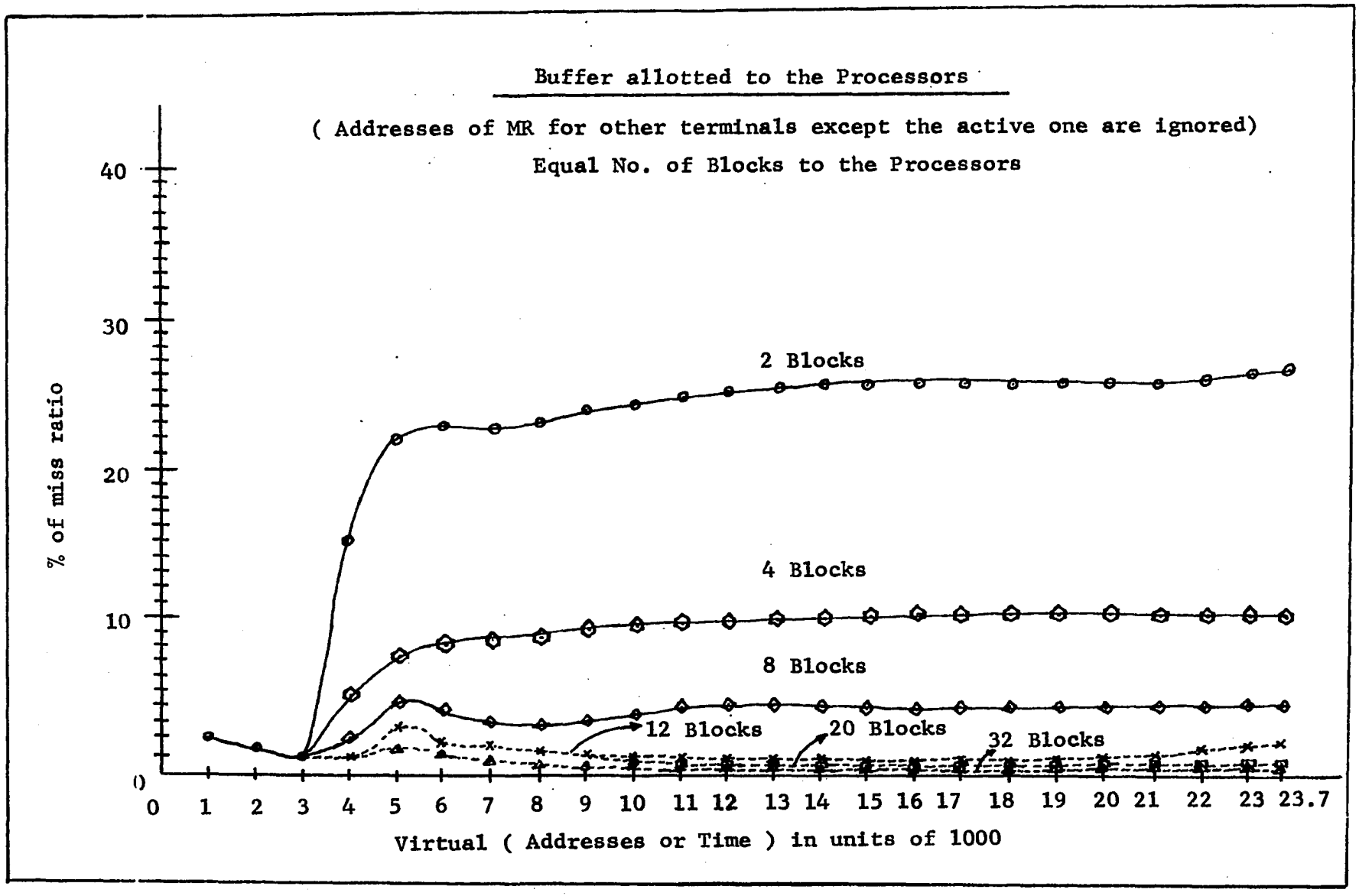

Fig. 19. Variation of miss ratio for MR

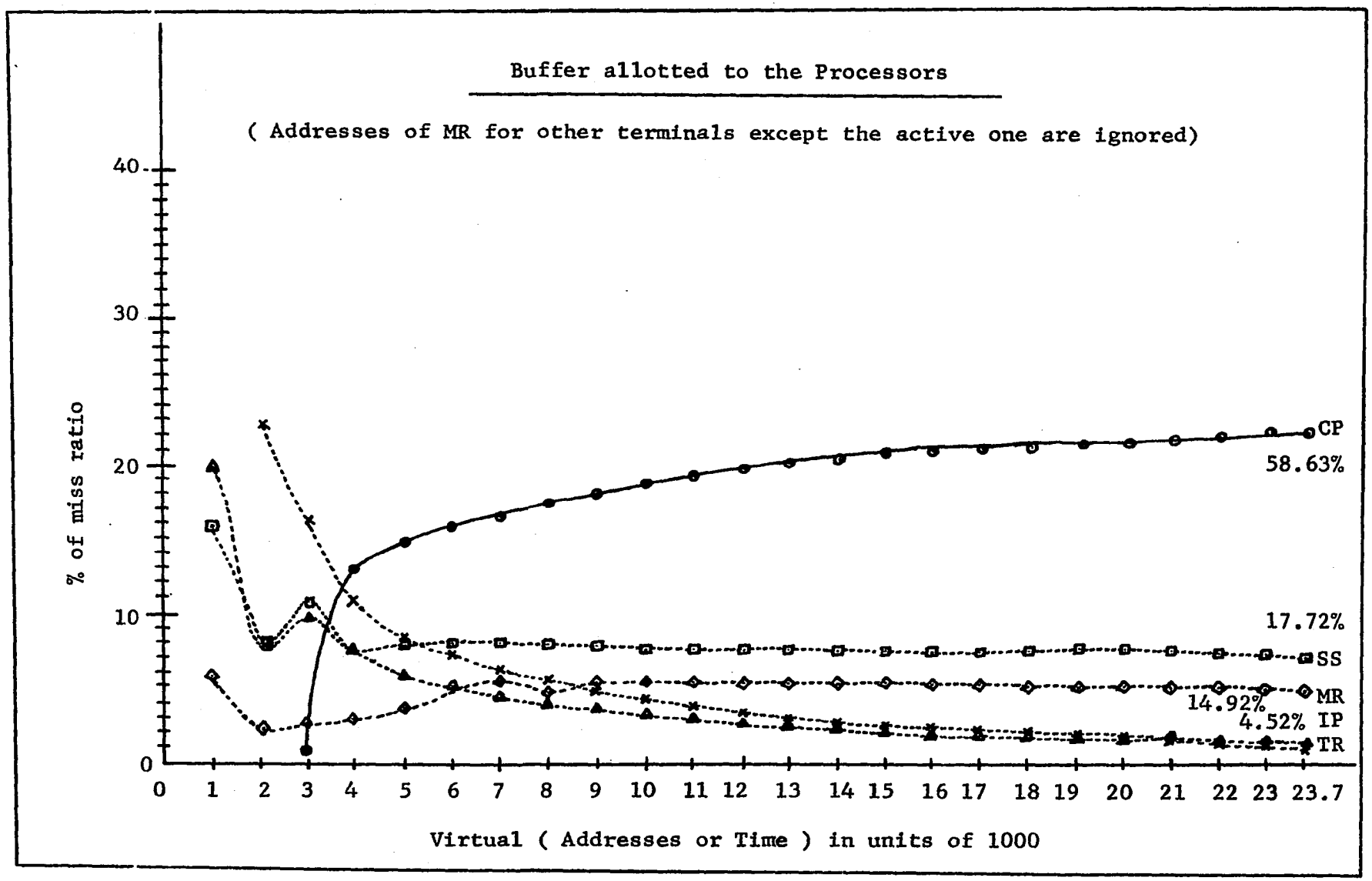

Fig. 20. Variation of miss ratio for different Processors

System Supervisor (SS) Without ignoring the requests by MR for other terminals services, SS had about 12% of total activity (Fig. 17). Now ignoring these requests the total activity of SS increases to 17.72% (Fig. 20).

For SS we see that there is no drastic improvement in performance until the total number of blocks alloted to SS is about 32 blocks (Fig. 21). There is a dramatic improvement in the miss ratio for SS from 24 to 32 blocks. This is because of certain design philosophies of the system. The SS has to go through a 'push' routine for marking the pages belonging to the particular terminal as inactive, when the job is completed. This involves the scanning of an In-core-list (ICL)--a list of pages which are in core at that time. This essentially results in scanning of 28 different page headers. This could result in 28 different blocks. Besides these, SS has to have one block of data corresponding to the terminal header and one block of data for System Queuing.

Even though the % of activity of SS is low compared to that of MR , because of its address reference pattern, a significant decrease in miss ratio for SS does not occur until its buffer has 32 blocks.

Central Processor (CP) Without ignoring the address requests of MR for other terminals, the CP constitutes 39.6% of total memory accesses (Fig. 17). However, ignoring these requests, the CP memory access percentage increases to 58.63%

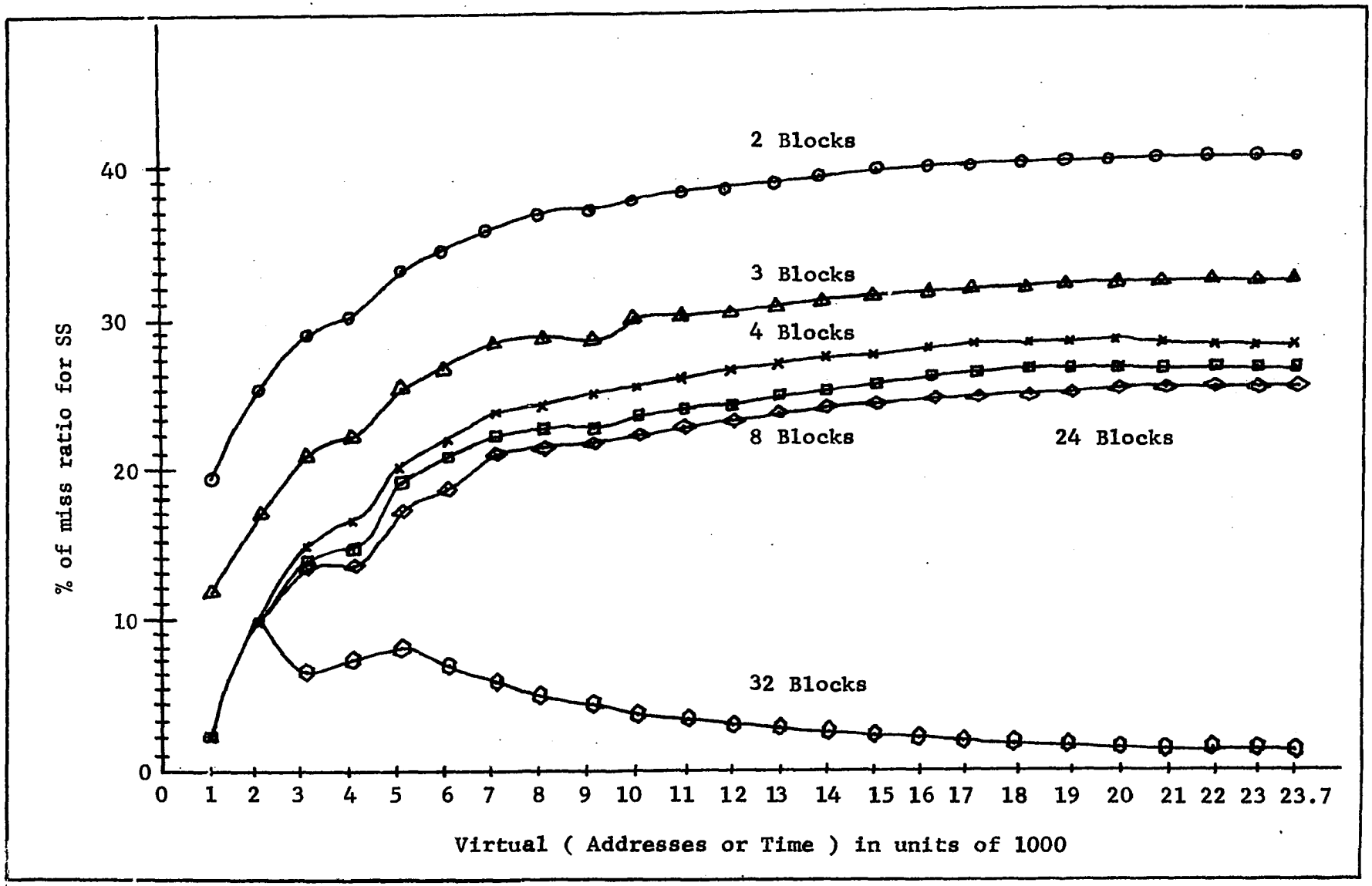

Fig. 21. Variation of miss ratio for SS

(Fig. 20).

As far as the variation of miss ratio with different number of blocks for the CP is concerned, we see three big jumps (Fig. 22).

For less than 3 blocks for the CP buffer the miss ratio is quite high. Increasing the number of blocks to 3 results in a significant increase in the hit-ratio. This is because CP has now roughly one block for object string, one for name table and one for stack or data. Increasing the buffer size to 4 blocks does not result in a very big increase in performance. However increasing the buffer to 8 blocks results in a sudden increase in performance. Afterwards not much increase in performance is obtained until the buffer size becomes 32 blocks--when it results in a hit-ratio of almost unity (Fig. 22). This is because now the CP has almost all the object string, name table and data in its buffer.

The amount of buffer allotted to the Central Processor is a function of the size of the program (because the size of object string and the size of the name table is a function of the size of the program)--and its pattern of reference. The scanning of the object string is almost sequential in nature--where as the scanning of name table and stack are not necessarily sequential.

Eight blocks for the CP buffer seem to yield quite good hit-ratio.

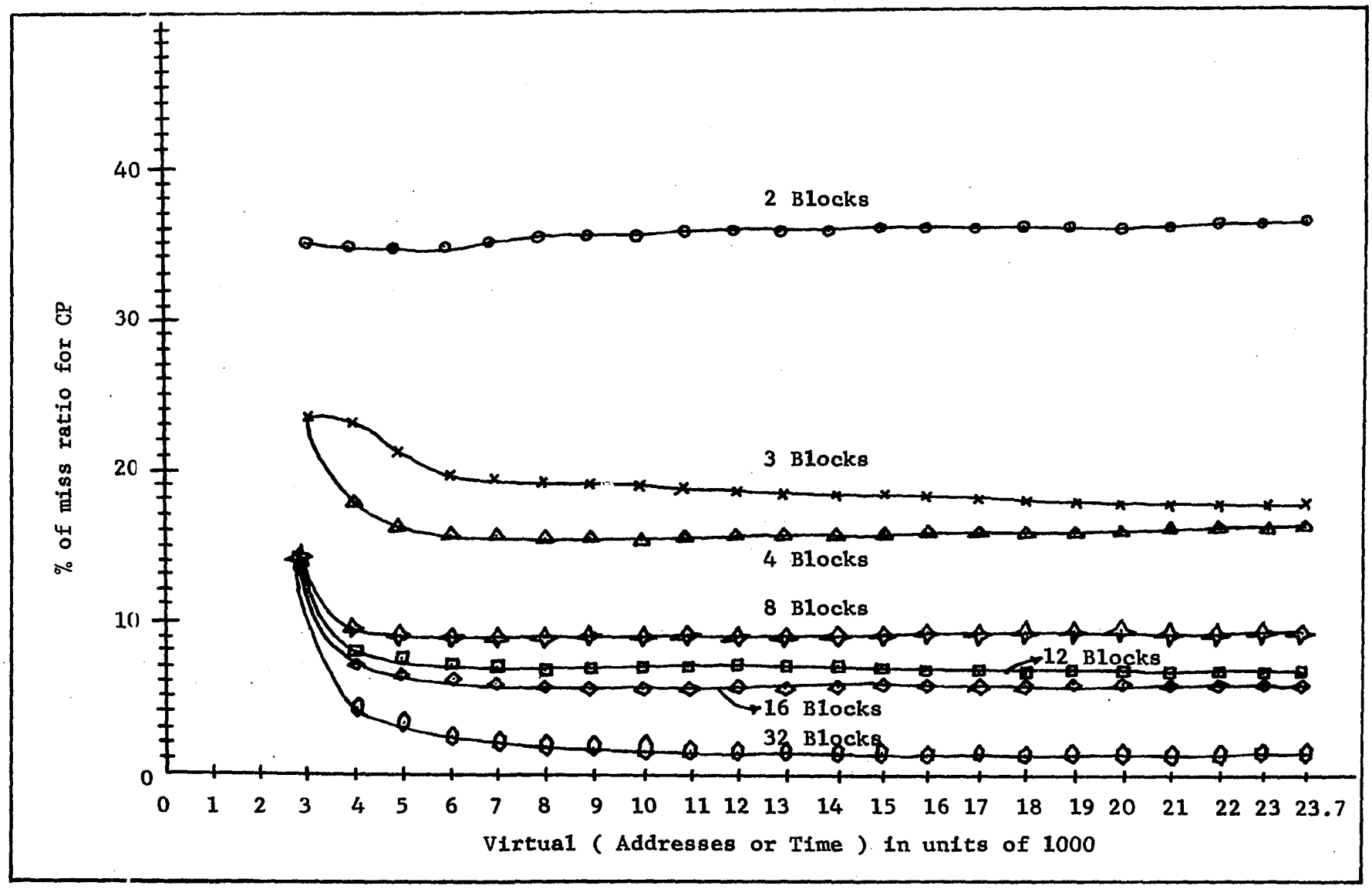

Fig. 22. Variation of miss ratio for CP

Translator (TR) The TR scans the source string sequentially and generates object code almost sequentially. It also needs to scan the the reserved word table and the name table if necessary. The addressing pattern of TR hence jumps between the source string to reserved word table to the name table and the generation of object code. Hence for the TR hit-ratio seems to improve with the increase with the sizes of buffer allotted to it.

The amount of activity of the TR is quite small (Fig. 17). However the buffer allotted to the TR is a function of the size of the program. From Fig. 23 we see that TR yields a reasonable hit-ratio when the number of blocks allotted to its buffer is around 16.

Input/Output Processor (IP) The IP scans the program almost sequentially. It needs one block to keep the source address and one block for the data.

We see that for 2 blocks the miss ratio is quite high and as the number of blocks is increased to 4 the miss ratio decreases quite sharply. Increasing the buffer to 8 blocks results in slight increase in hit-ratio (Fig. 24). Hence about 4 blocks are enough for the IP.

Hence we see that as far as the partitioning of the buffer for the processors is concerned the following numbers seem to yield a very good hit-ratio:

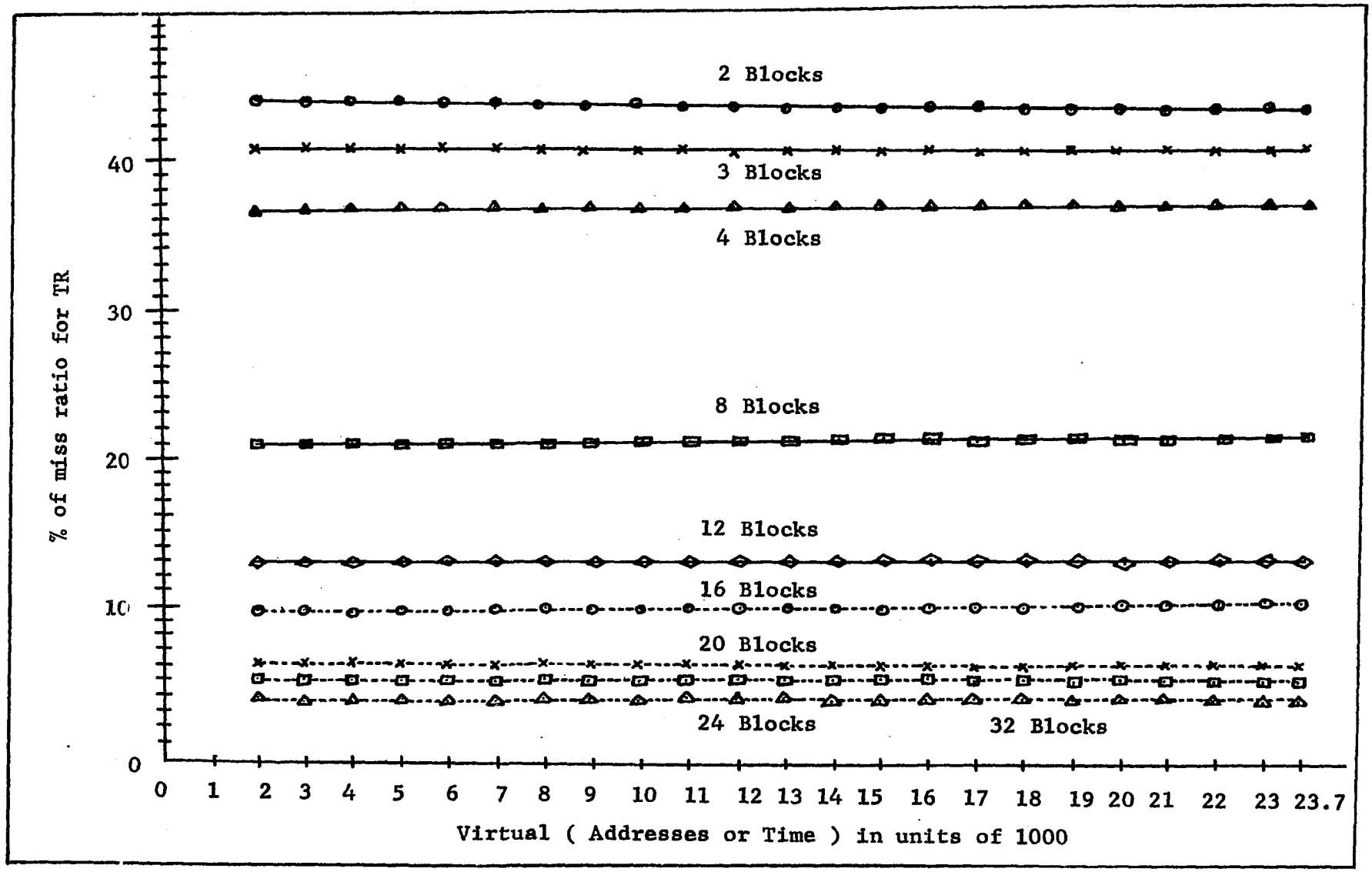

Fig. 23. Variation of miss ratio for TR

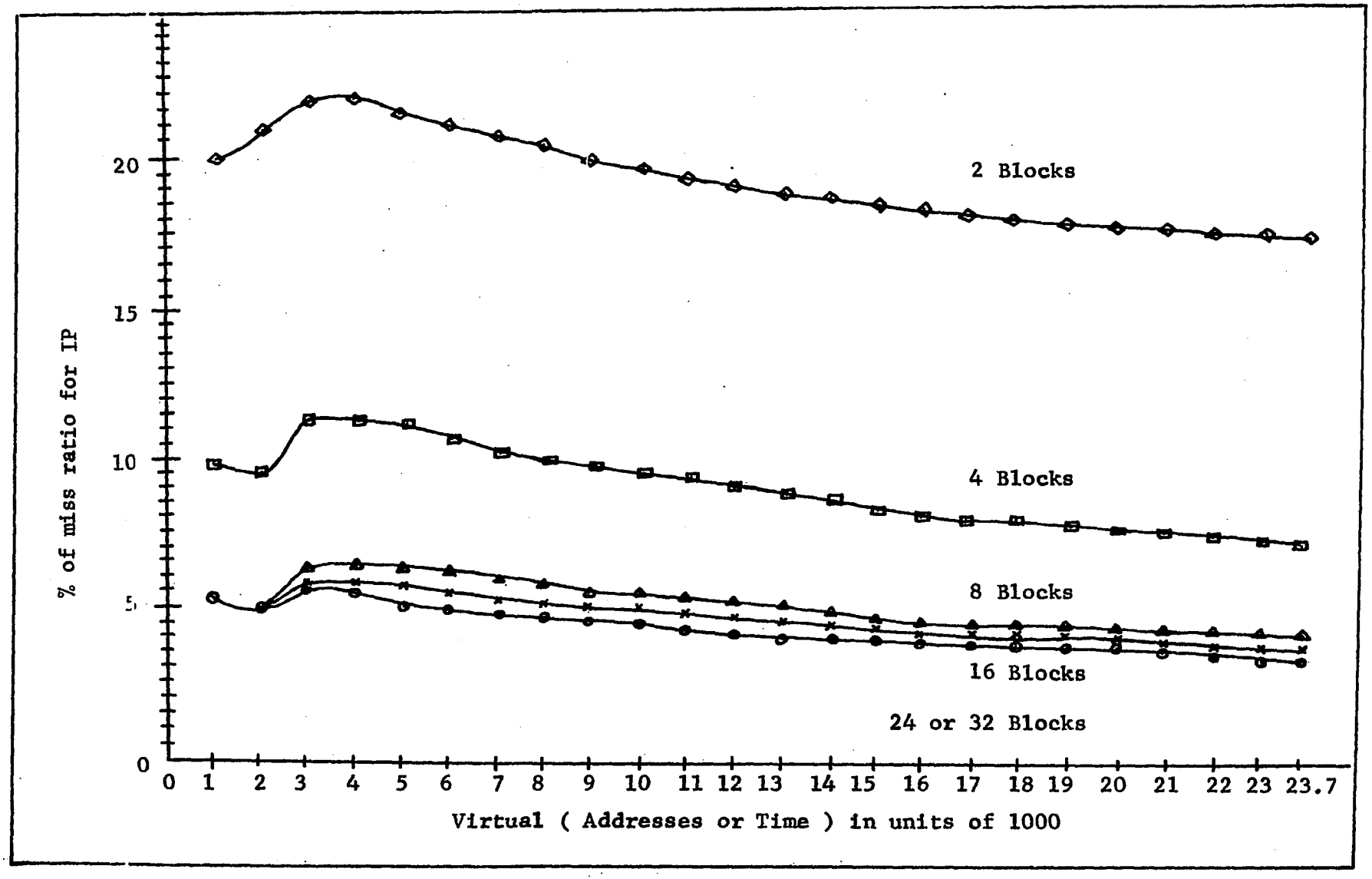

Fig. 24. Variation of miss ratio for IP

16 blocks for MR

16 blocks for TR

8 blocks for CP and

32 blocks for SS

or a total of about 76 blocks results in a dramatic improvement in performance.

Over all miss ratio

Fig. 25 illustrates the variation of over all miss ratio with the virtual time, for different buffer sizes. As we see in the figure, equal buffer size is allotted to each processor except the two cases where the buffer size for MR is made equal to the total buffer sizes for all other processors. This figure illustrates the case where the addresses of MR corresponding to other terminals are not ignored.

From the figure we observe that a total buffer size of only 32 blocks--4 blocks each for SS, IP, CP and TR and 16 blocks for MR yields the best hit-ratio. This seems to be little surprising because considering the optimum buffer size for each processor individually the total optimum buffer size was found to be 76 blocks. Hence considering the over all hit-ratio not much is gained by increasing the buffer size to more than 32 blocks and 32 blocks are adequate.

Fig. 26 illustrates the variation of over all miss ratio with the virtual time for different buffer sizes--when the addresses of MR corresponding to other terminals except the

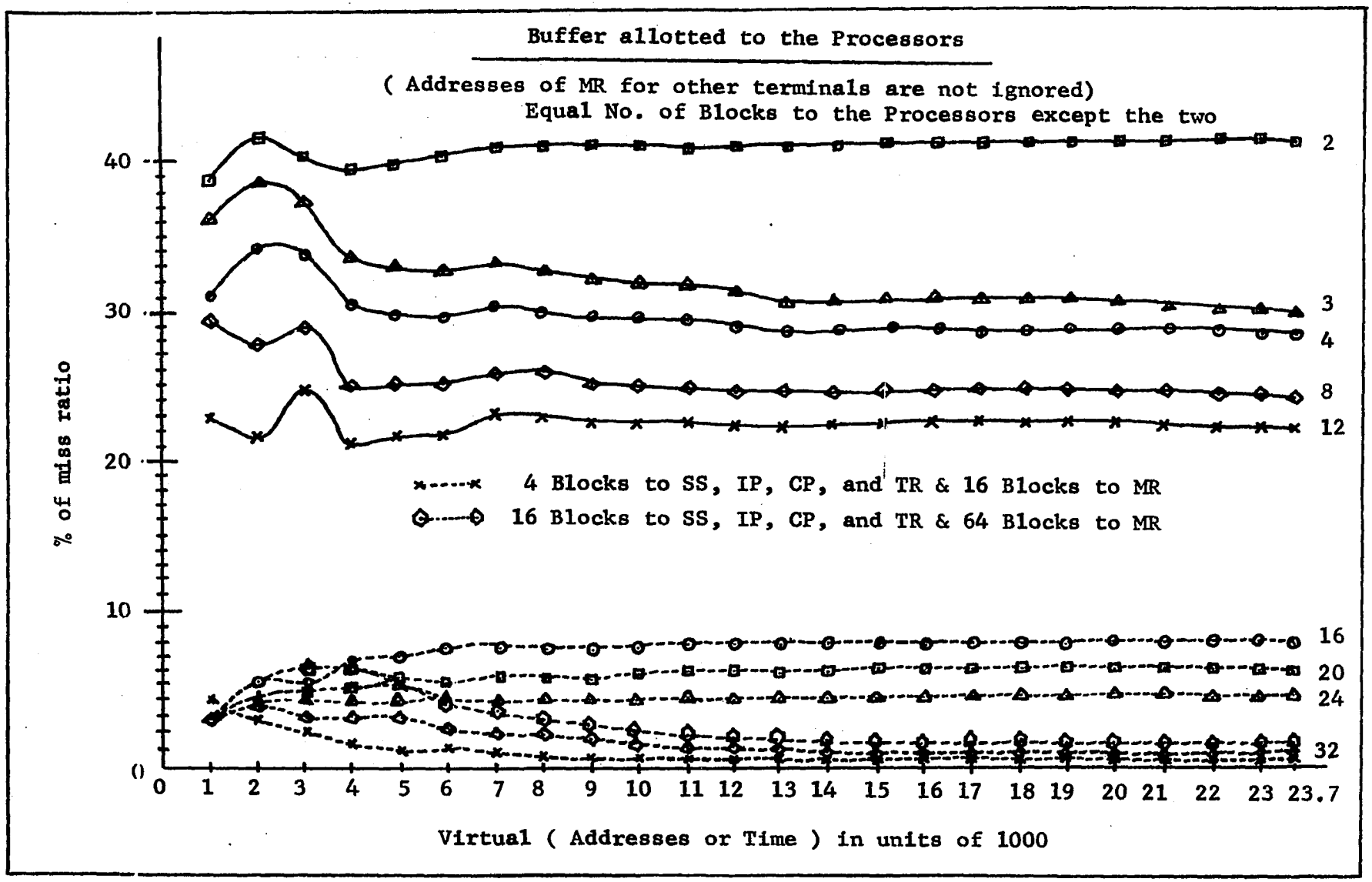

Fig. 25. Variation of Over All miss ratio

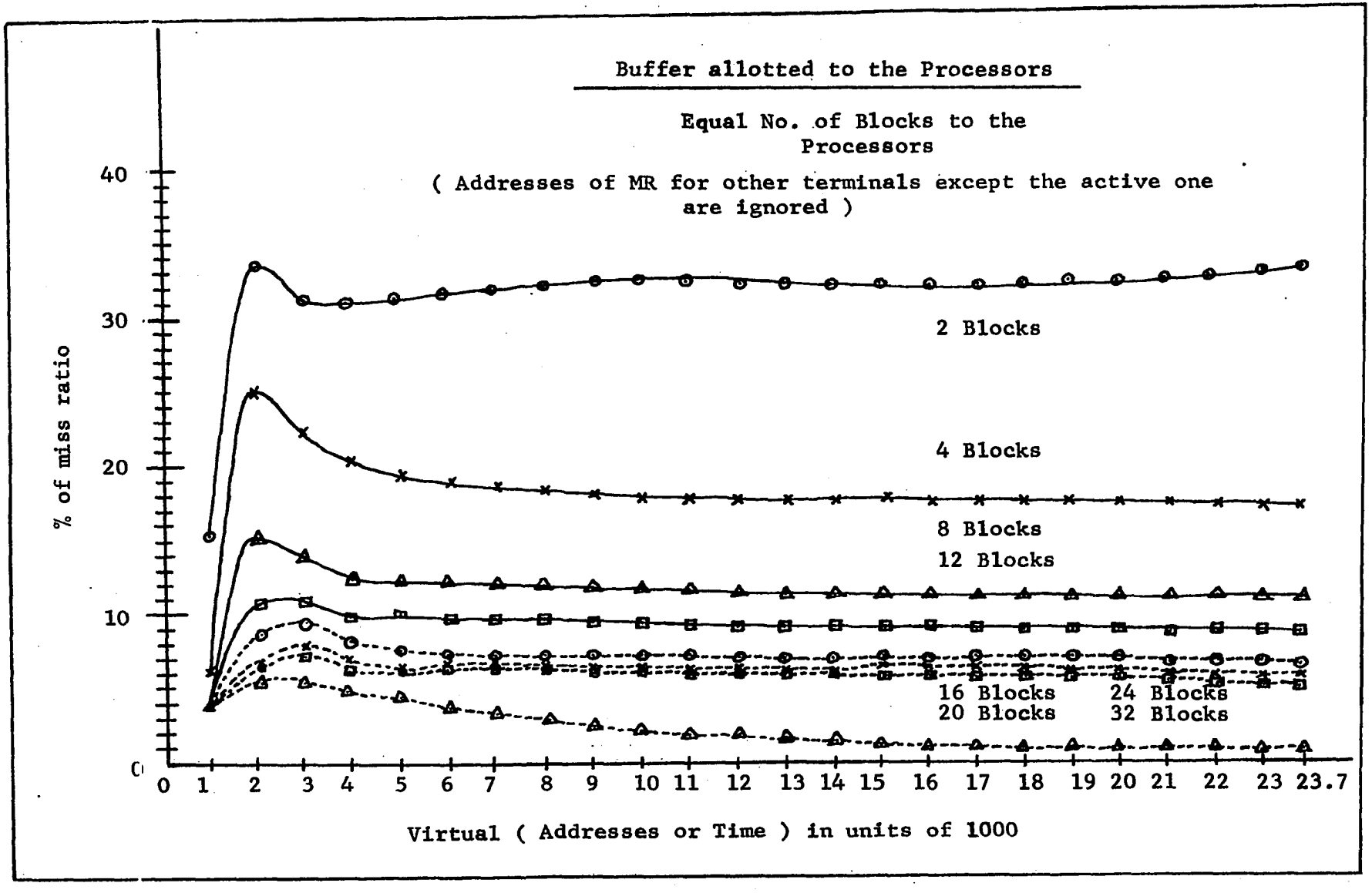

Fig. 26. Variation of Over All miss ratio

active one are ignored. This plot also illustrates the case when Sub-global LRU replacement policy is used.

Comparing Fig. 25 and Fig. 26 we observe that for smaller buffer sizes, the case with ignoring the addresses of MR for other terminals results in better hit-ratio--however as the buffer size is increased the unstripped case tends to give better results.

Fig. 27 illustrates the over all miss ratio percentage variation for various buffer sizes for processors using Sub-local LRU replacement policy. All the processors are allotted equal buffer space. The addresses of MR for other terminals except the active one are now ignored. Comparing Fig. 27 and Fig. 25 we observe that for smaller buffer sizes Sub-local policy yields a dramatic improvement in performance over the Sub-global policy. However, as the buffer size is increased, there is not much difference between the two and both tend to give about the same results. This proves the superiority of Sub-local policy over the Sub-global policy for smaller buffer sizes when the buffer is allotted to the processors.

Buffer allotted for the whole system

One other possibility of buffering SYMBOL-IIR like computing structures is to provide a buffer for the whole system.

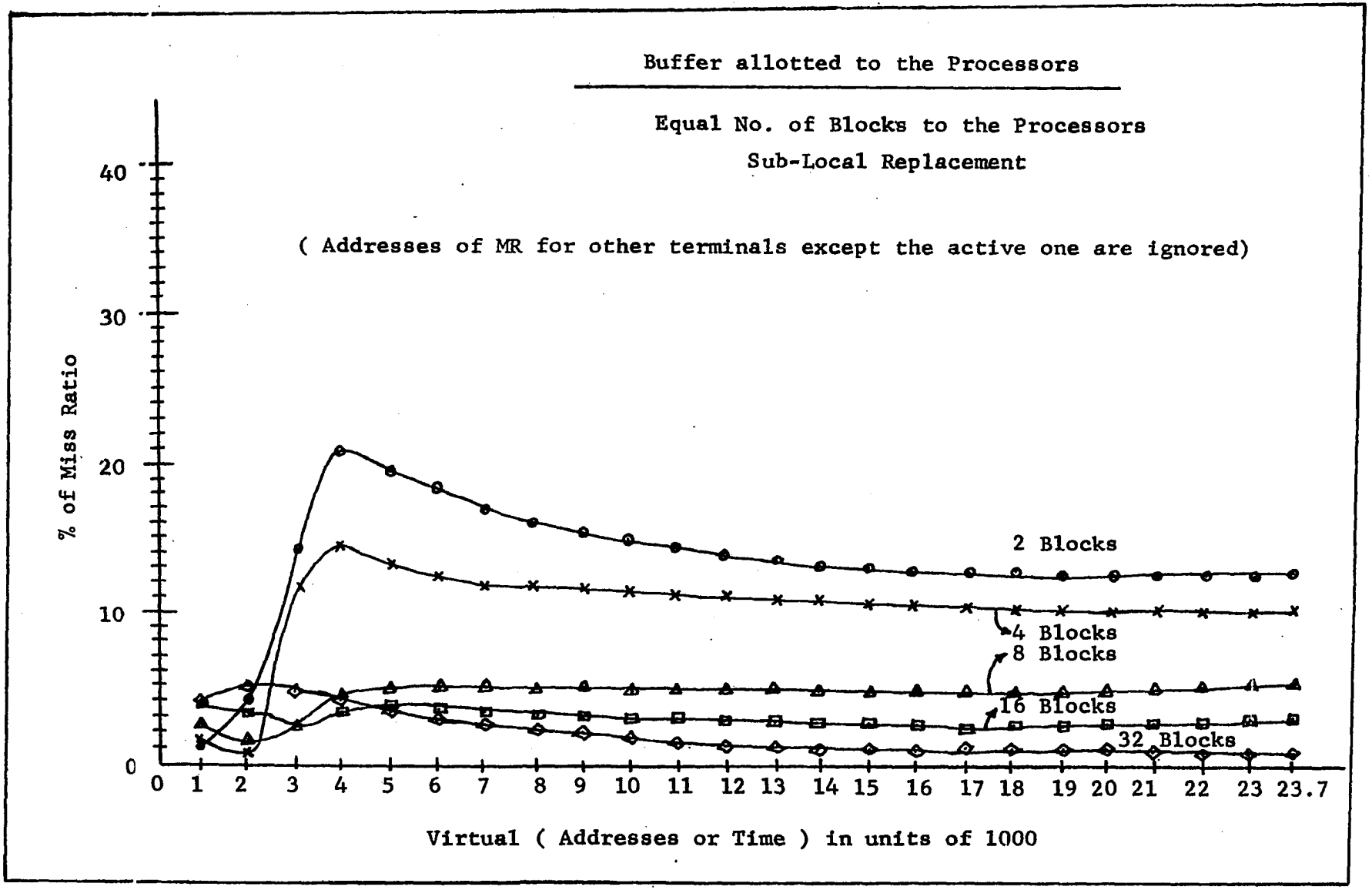

Fig. 27. Variation of Over All miss ratio

Now like a conventional system, the whole buffer can be partitioned into certain number of classes. One technique of partitioning is on the basis of lower order bits of virtual page address. Now each class has its own directory and priority update list--and there is no sharing of information between these classes.

Based on this principle, simulation analysis was carried out for three different cases as follows:

1. 4 classes
2. 8 classes and
3. 16 classes.

The total buffer size was fixed to 64 blocks and each class was allotted the same amount of buffer space. From this analysis it is seen from Fig. 28 that 4 classes with 16 blocks per class yielded the best hit-ratio and 16 classes with 4 blocks per class yielded the worst hit-ratio. From this figure we see that for a fixed buffer size, a small number of classes with a large number of blocks/class yields better hit-ratio than a large number of classes with a small number of blocks/class. However even with 4 classes and 16 blocks per class the miss ratio is quite high--17.5%.

Partioning of buffer space functionally Partitioning of buffer space into various classes (like the conventional machines) for the whole system essentially partitions the whole buffer space into a collection of different

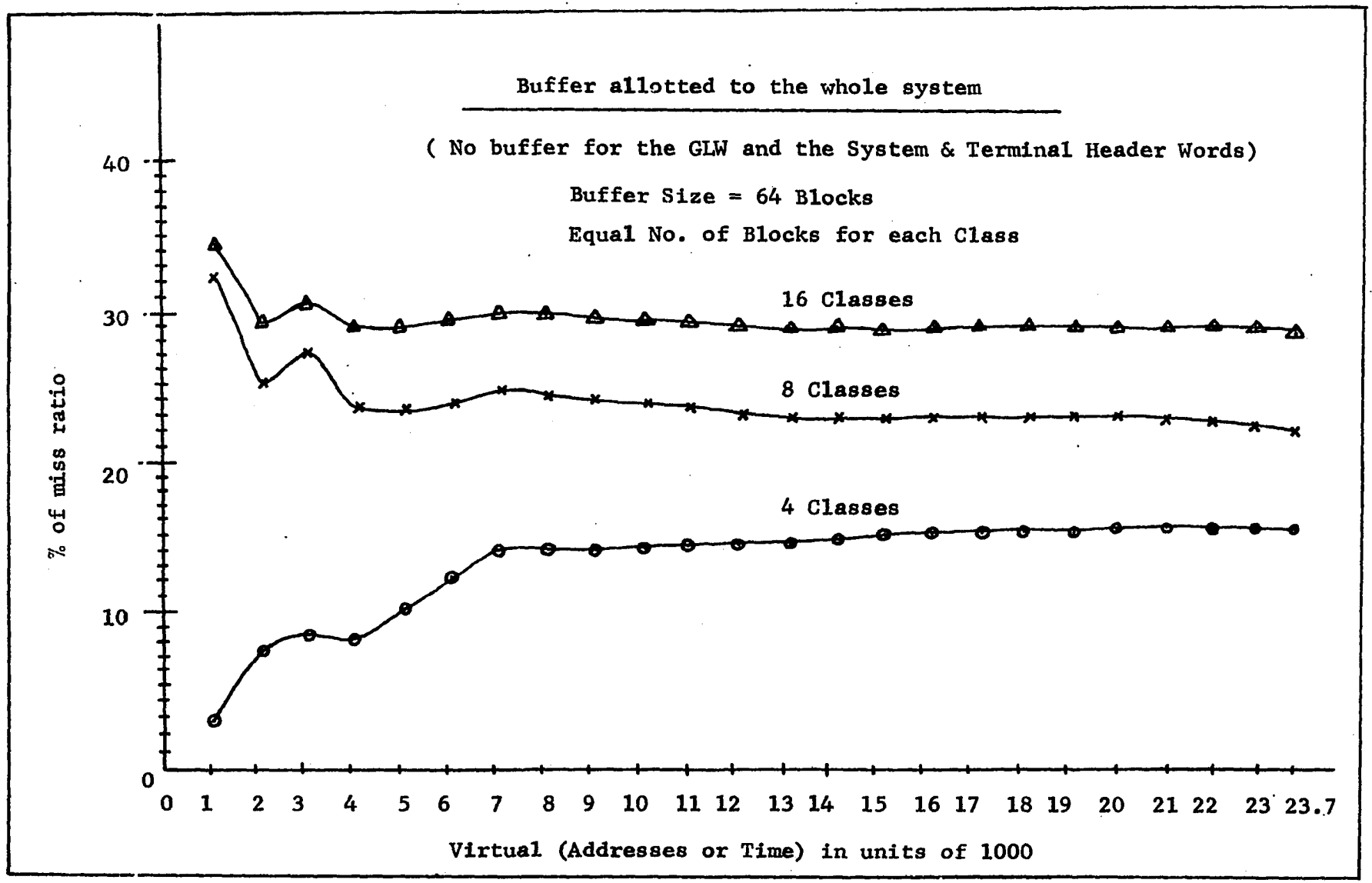

Fig. 28. Variation of Over All miss ratio

heterogeneous units. Another approach might be that of partitioning the whole buffer space functionally--where now the buffer might be looked upon as a collection of different homogeneous units--one for page list 0, one for page list 1--for the source string, one for page list 2--for name table and data and one for page list 3--for the object string. Actually there is no such page list as 0 but, some of the processors use page list 0 when they do not need any space. The maximum percentage of this page list 0 addresses is used by the System Supervisor (SS). Addresses corresponding to page list 0 are allotted a separate buffer space. Based on this a simulation analysis was carried out for different buffer sizes. Fig. 29 illustrates the variation of over all miss ratio percentage with different buffer sizes. From this we see that a buffer of about 64 blocks yields reasonable hit-ratio. However, even with 64 blocks the miss ratio is quite high compared to that of 32 blocks for the buffer allotted to the processors.

Buffer allotted to the terminal

Another way of buffering SYMBOL-IIR like computing structures is to allocate certain buffer space on a strictly job or terminal basis.

We see that on a terminal basis, the hit-ratio increases as the buffer for the terminal increases. With 8 blocks the miss ratio is quite high--about 33% and increasing the buffer

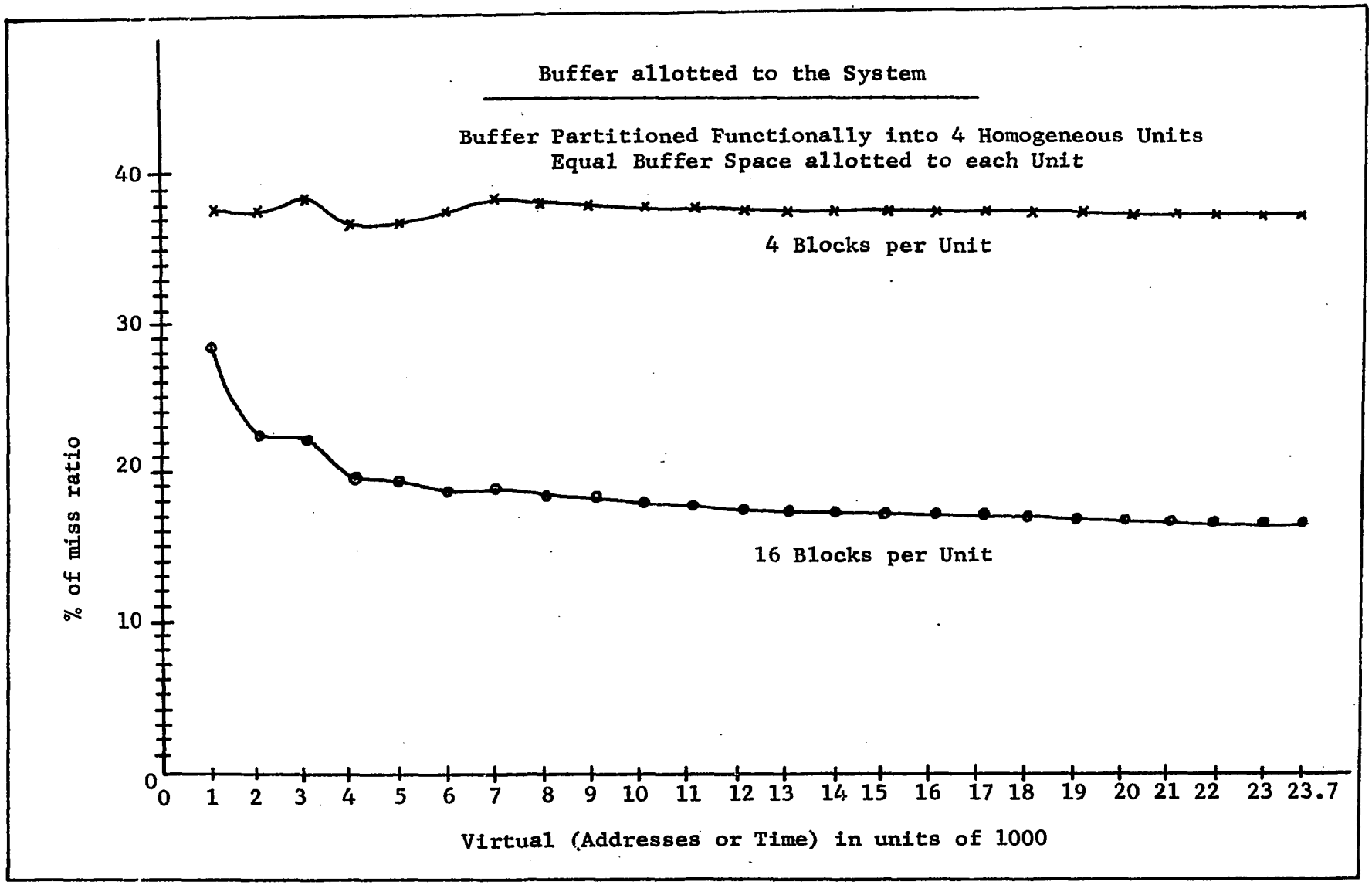

Fig. 29. Variation of Over All miss ratio

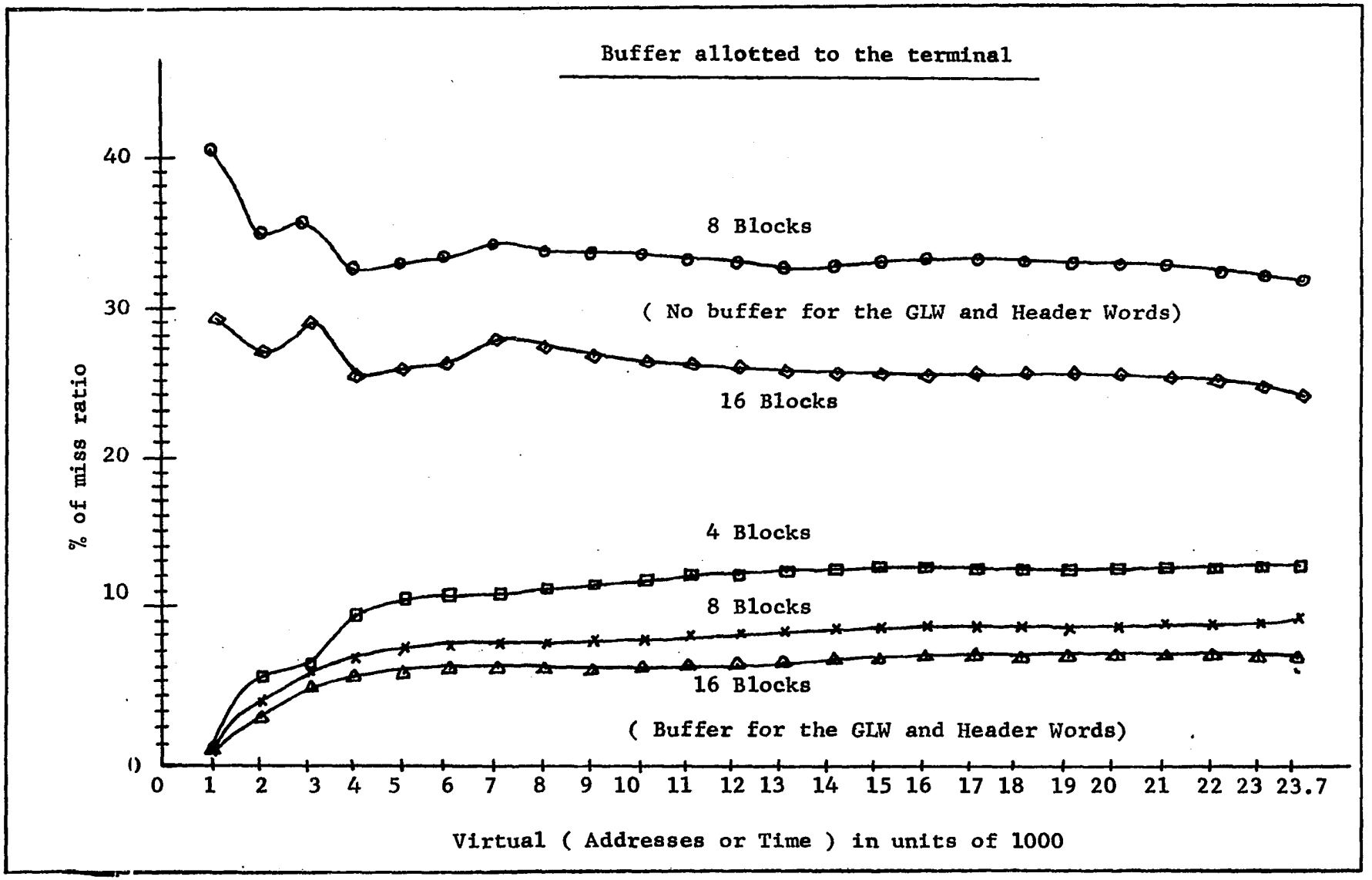

Fig. 30. Variation of Over All miss ratio

to 16 blocks results in a miss ratio of 26%.

However providing separate buffer space for the group link word and the system and terminal header words results in a dramatic improvement in performance with fewer blocks. Allotting separate buffer space for the group link word and system and terminal header words, we see that for 16 blocks for buffer, the miss ratio is about 7.5% (Fig. 30).

Percentage of group link words and header words

In order to justify a separate buffer for group link words and system and terminal header words it is necessary to know the frequency of references for these. If the frequency of reference is quite small, then there is no need for providing separate buffer spaces for these.

From Fig. 31 we see that group link words are accessed roughly 1/8 of the total memory accesses. Surprisingly, the system and terminal header words accesses constitute roughly 50% of the total memory accesses. So both of these combined constitute roughly 2/3 of total memory accesses--which makes one think about providing a buffer for them.

As mentioned before, there is no need to keep the system and terminal header information in the same core. Hence the whole header information can be put in a separate memory and a fast buffer for the group link words (GLW) and the system and terminal header words can be provided.

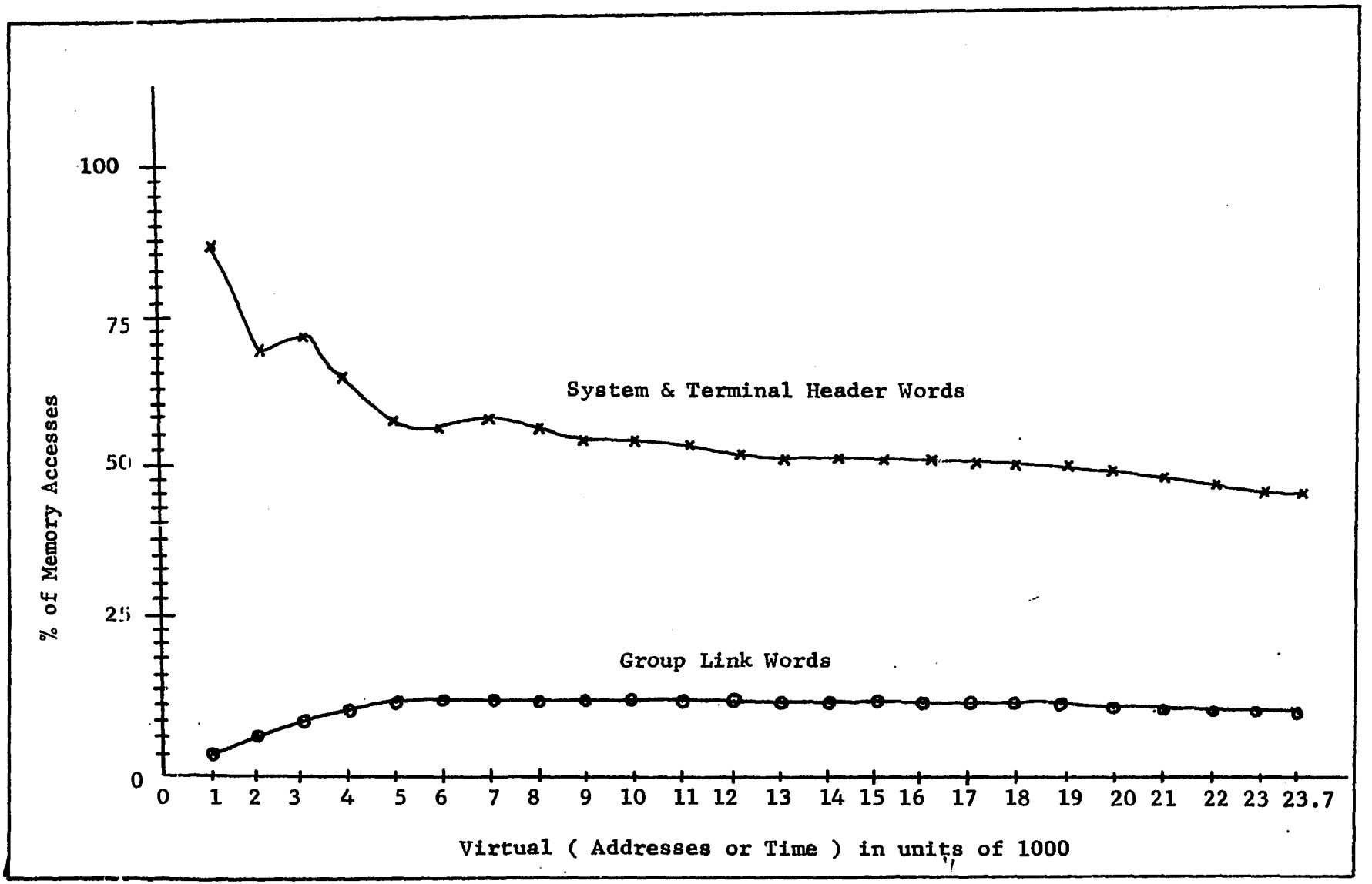

Fig. 31. Percentage of Glw and System and Terminal header words

The buffer size for the group link word for each class will be w words if the total number of blocks for that buffer is w. The size of the buffer for the GLW would be quite small. However, the size of the buffer for the system and terminal header words would vary depending upon the partitioning scheme that is whether the buffer is allotted to the whole system or to the terminal or to the processors. The size of the buffer for the processors case would be quite small--about one extra block for each processor. Allotting one more block for TR and SS would further improve the performance.

Based on this the hit-ratio was computed for various classes for various buffer sizes and the result is shown in Fig. 32.

From Fig. 32 we see that providing a separate buffer for GLW and header words and partitioning the buffer into just 4 classes, a buffer of 8 blocks/class results in a miss ratio of about 6 to 7 %--without providing separate buffer for GLW and header words had resulted in a miss ratio of about 17% for 4 classes with 16 blocks/class (Fig. 29). Hence this results in an improvement of about 2.5 times in the miss ratio.

From Fig. 30 we see that providing a separate buffer for GLW and header words, the miss ratio decreases to about 8% for 16 blocks for the terminal--about 2 times improvement in

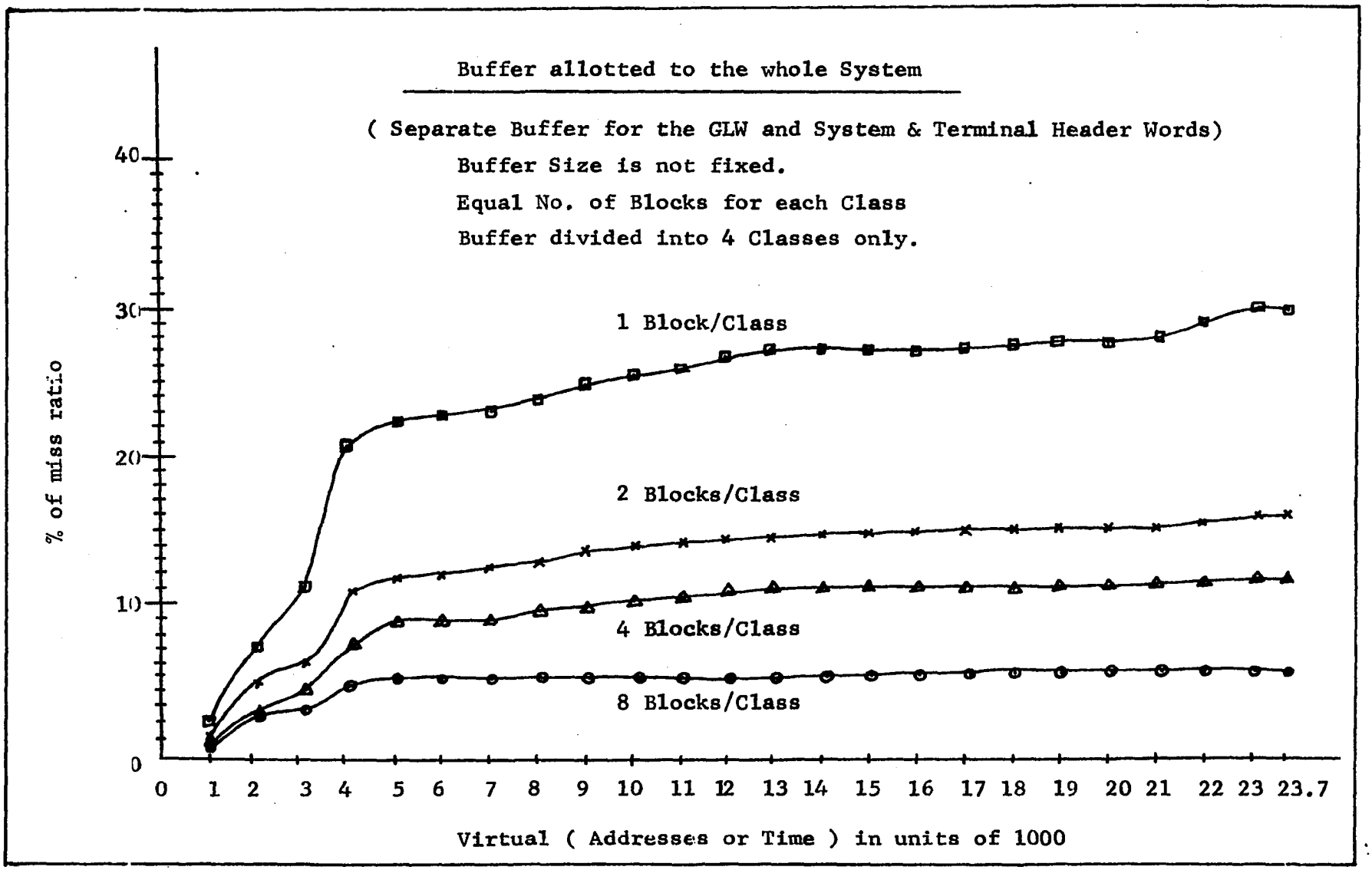

Fig. 32. Variation of Over all miss ratio

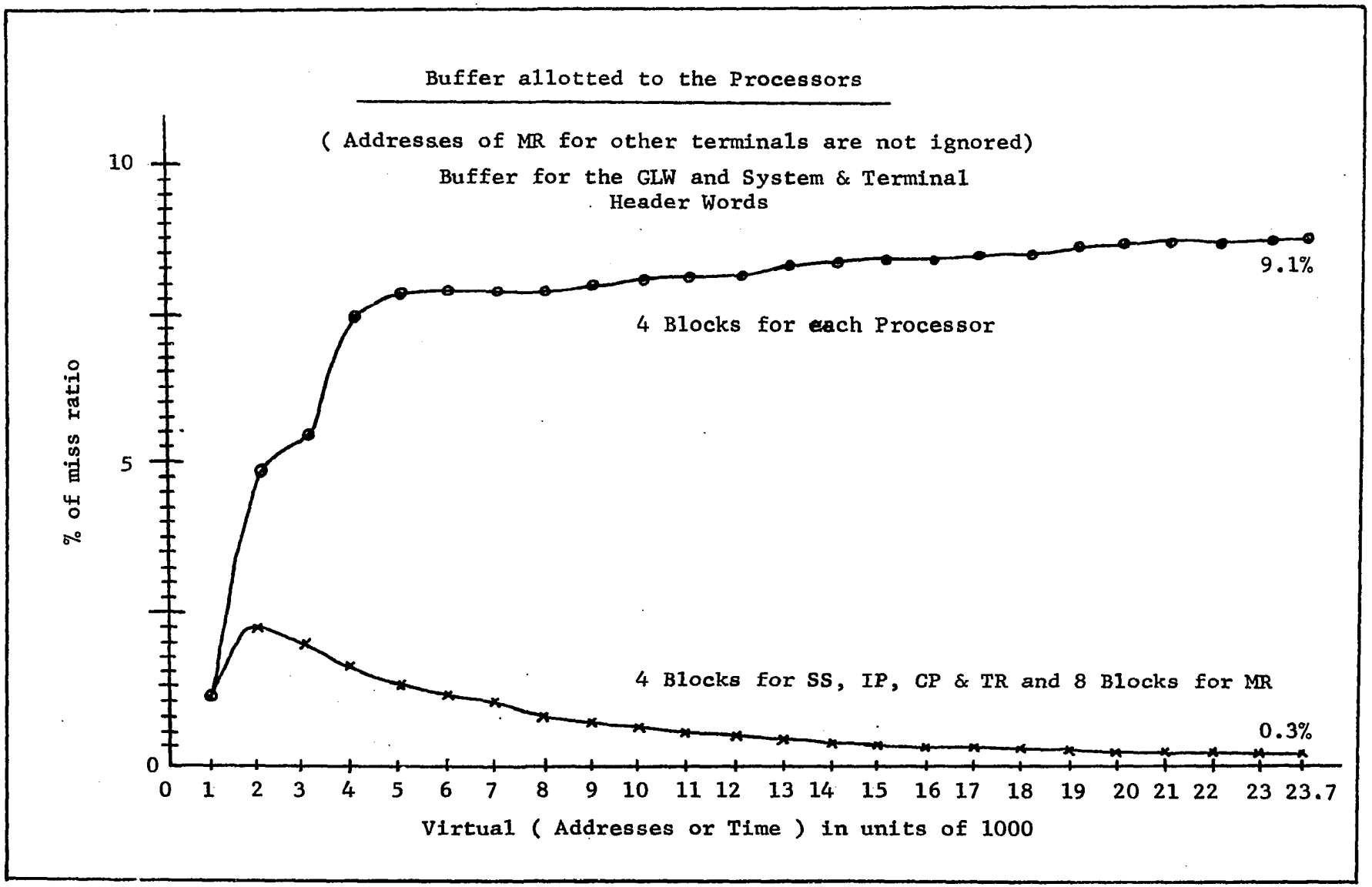

Fig. 33. Variation of Over all miss ratio

the performance.

From Fig. 33 we see that providing a separate buffer for GLW and allocating extra buffer for the system and terminal header words, for different processors, the miss ratio dramatically improves. The improvement can be seen by comparing Fig. 29 and Fig. 33.

Processor traffic rate

Fig. 34 illustrates the variation of average traffic rate for different processors with different number of blocks allotted to their buffers. As mentioned before, the figure of merit for the processor traffic rate is that it should be as low as possible.

For MR, the traffic rate is quite high until its buffer contains 16 blocks--when the traffic rate dramatically reduces to very low value. After 16 blocks, an increase in the number of blocks does not significantly reduce its traffic rate.

For SS, the traffic rate reduces little from 1 to 4 blocks and then remains constant till its buffer has 24 blocks--after which an increase in number of blocks to 32 results in a dramatic improvement in traffic rate.

For TR, the traffic rate consistently declines with the increase of number of blocks. From Fig. 34 we see that about 16 blocks for TR buffer yields a good traffic rate.

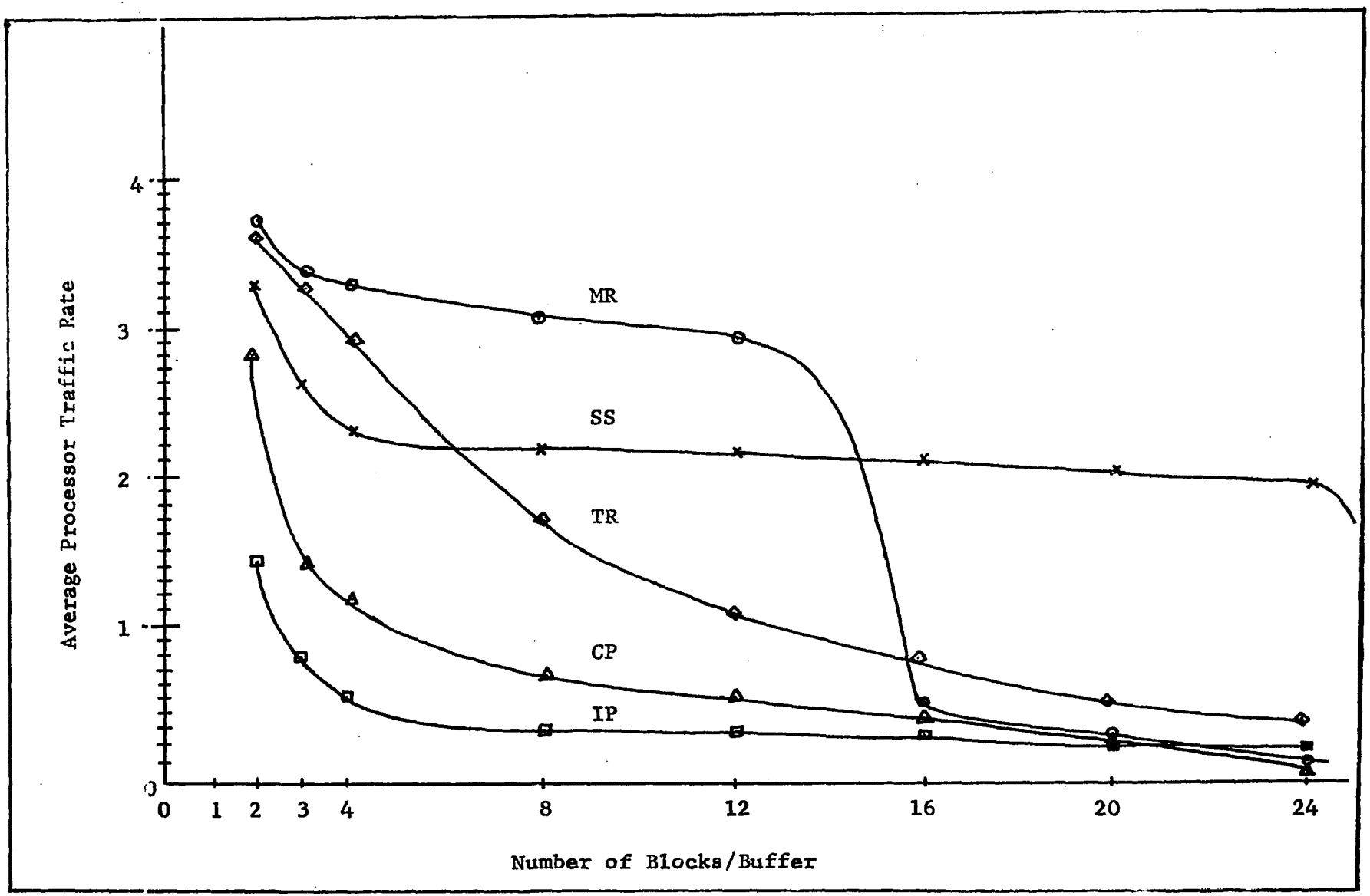

Fig. 34. Traffic Rate of different Processors

For CP, there is a sharp decline in the traffic rate from 2 to 3 blocks--after which the traffic rate decreases slowly until the buffer size is increased to 8 blocks--after which it decreases persistently.

For IP, the traffic rate decreases from 2 to 4 blocks and then it remains almost constant.

Traffic rate as mentioned before gives an idea about the usefulness of the blocks being transferred from the main memory to the buffer. Hence a high hit-ratio should imply a low traffic rate and vice versa. The figures obtained for the traffic rate are compatible with those obtained for the hit-ratio data.

Average block utilization

Average block utilization is the average number of words of a block referenced between two successive block swaps.

Fig. 35 illustrates the average block utilization for different processors with the number of blocks. A good figure of merit for average block utilization is that it should be very high and close to unity.

For MR, the average block utilization remains low until its buffer contains 16 blocks--when it increases and remains constant. As low block utilization illustates the uselessness of extra words in the block, it is observed that about 16 blocks for the MR seem to be the optimum size--before any reasonable size hit-ratio is obtained.

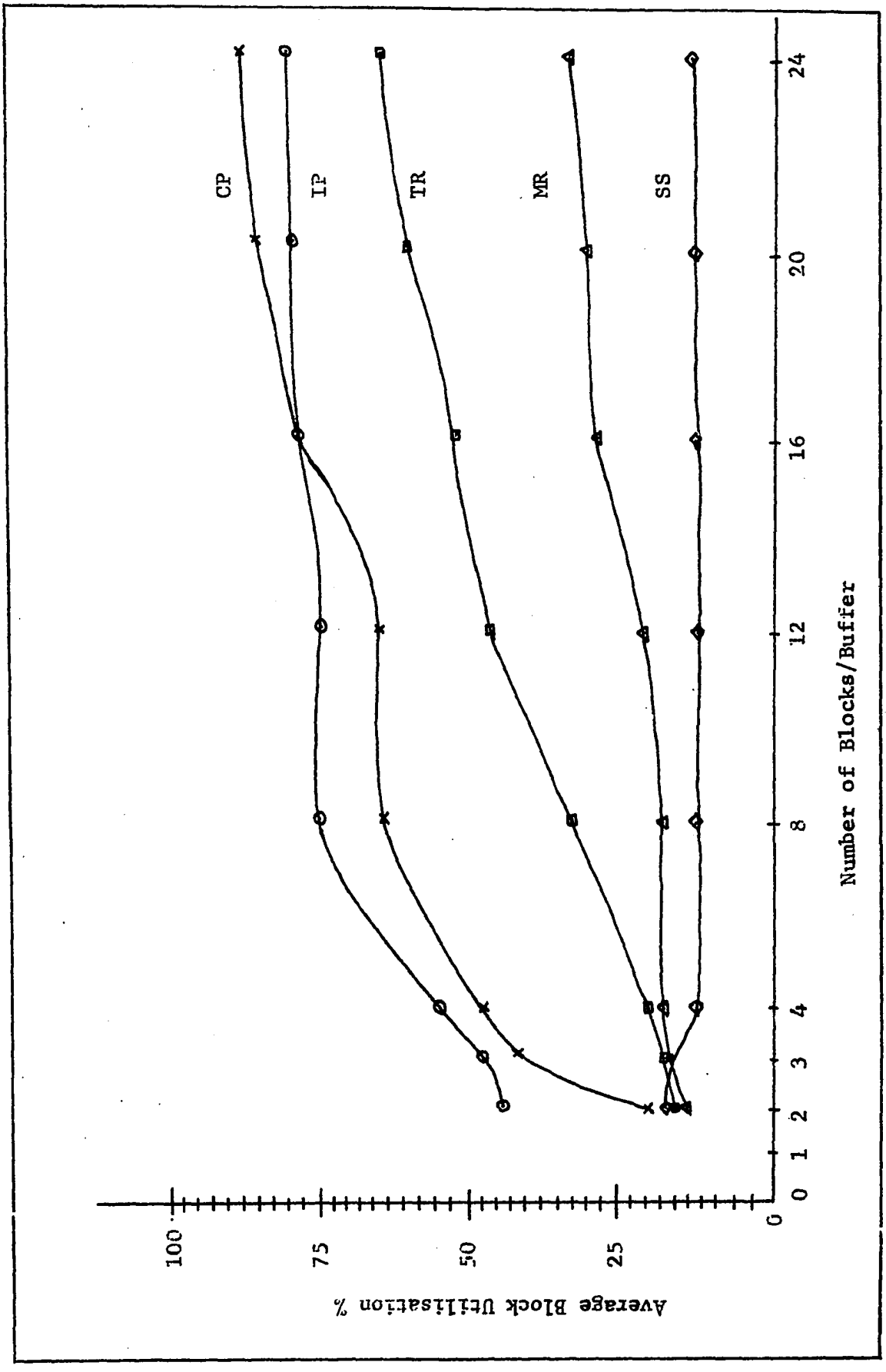

Fig. 35. Average Block Utilization

The average block utilization is lowest for the SS. It is at its maximum value when its buffer has 2 blocks after which it decreases and remains constant till the buffer size is 24 blocks.

For TR, the average block utilization increases persistently with the inc Initially the average block utilization is about 14% with 2 blocks and it increases to about 60% with 24 blocks.

For IP, the average block utilization increases from 2 to 8 blocks--after which it almost remains constant.

Initially for 2 blocks the average block utilization is about 42% and it increases to about 75% for 16 blocks.

For CP, the average block utilization is lowest for 2 blocks. Increasing the buffer size to 3 blocks results in a sharp increase in the average block utilization. It keeps on increasing from 4 to 8 blocks--however the increase is not so sharp. Then again a big increase is encountered from 12 to 16 blocks.

For 2 blocks the average block utilization is about 20%--and for 24 blocks the average block utilization increases to about 85%.

These results are also quite compatible with the hit-ratio data.

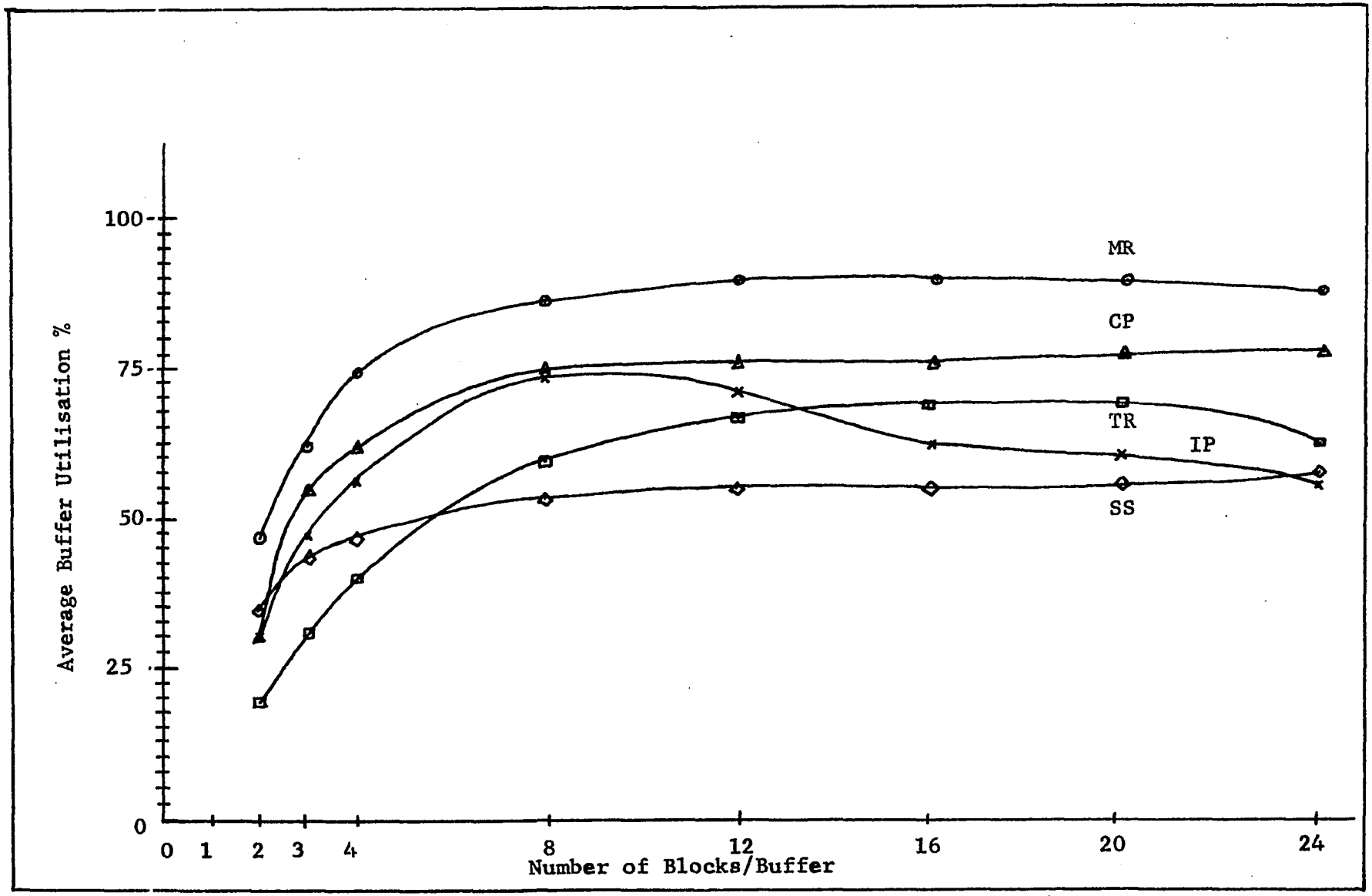

Fig. 36. Average Buffer Utilization

Average buffer utilization

Average buffer utilization is the average number of blocks of the buffer referenced at least once between two successive block swaps. This figure gives an idea about the usefulness of extra blocks in the buffer.

The average buffer utilization for MR, CP, and SS increases from 2 to 8 blocks and then remains almost constant (Fig. 36).

For TR the average buffer utilization is lowest when it has 2 blocks in the buffer--after which it increases until it has about 16 blocks. The average block utilization remains constant till about 20 blocks and then decreases as the number of blocks is increased.

For IP, the maximum buffer utilization is obtained for 8 blocks after which it starts declining.

From average buffer utilization point of view we see that not much is gained from increasing the buffer sizes for MR, CP, SS and TR to more than 12, 12, 12 and 16 blocks respectively.

Conclusion

In this chapter hit-ratio data were obtained for different buffer partitioning strategies for various buffer sizes. From the analysis it is seen that the best way of designing buffered memory system for SYMBOL-IIR like computing structures will be to allocate separate dedicated but sharable

buffer spaces to its different multidedicated processors.

The amount of buffer allotted to various processors is quite small and yet it yields a very good hit-ratio. The amount of a processor's activity and its address reference pattern seem to affect tremendously the decision for optimum buffer sizes for getting reasonable hit-ratio.

Partitioning the buffer spaces on the processor basis results in a multihomogeneous buffer. Even though in computing structures like SYMBOL-IIR the terminals do not share information or memory space, the processors are shared by them and a processor can work for only one terminal at any instant of time. Hence even though the buffers of the processors would be dedicated, for managing this buffered memory system the principle of global searching with local replacing should be used.

The data were collected with programs running one at a time, on a single terminal only. The analysis, which was carried out for the processors, was also based on the assumption that processor was serving one terminal at a time.

Partitioning buffer space into separate homogeneous units for different processors for getting better hit-ratio leads to one of the important system concepts. Instead of all the processors sharing a common memory, now sufficient memory space can be assigned to each processor. However, the memory space assigned to these processors should be adequate

and large enough for obtaining better hit-ratio.

Obtaining a high hit-ratio is part of the solution of the whole problem. As mentioned before, one of the major assumptions of the whole thesis was that the "main memory of SYMBOL-IIR like computing structures is large enough so that speed and performance is not limited by the paging traffic" because, buffering is no solution to a virtual memory system which is limited by the page traffic. So this presumption of the existence of a large main memory with the capability of transferring a block of data at a time to the buffer would necessitate the main memory to be organized as an interleaved system. So besides a high hit-ratio the other two important parameters are--a fast effective cycle time with minimum cost. Buffering should be tremendously cost-performance effective--otherwise it would lose all of its charm and importance. In the next chapter a cost analysis of buffered memory systems for SYMBOL-IIR like computing structures is carried out and it is demonstrated that buffering seems to be the cheapest way of improving the performance.

Besides the architectural organization and the principle of "locality", the program behaviour would also seem to have tremendous impact upon the organization and management of SYMBOL-IIR like computing structures. Since the interface unit is already built, it would be interesting to see the effect of different kinds of programs on the buffering

schemes. The effect of different users environments upon the hit-ratio data is discussed further in the chapter VI.

CHAPTER V. COST PERFORMANCE ANALYSIS

Introduction

"At present day commercial systems cost rather than speed has become a dominant consideration for memory modules" (34).

An experimental buffered memory system for SYMBOL-IIR like computing structures might look as in Fig. 37. This involves the following:

1. The buffer memory itself
2. A buffer controller
3. A directory
4. A priority update list and
5. The switching network.

The switching network is for switching information between the outgoing data bus and various memory modules.

The buffer memory controller, priority update list, directory and switching network add extra cost. To illustrate this extra cost increase, the cost of a buffered memory system using the above type of buffer is computed.

The costs assumed are fairly typical for the state of the art technology.

The main memory is thought of organized as core modules. The core costs are fairly typical for the present original equipment manufacture market.

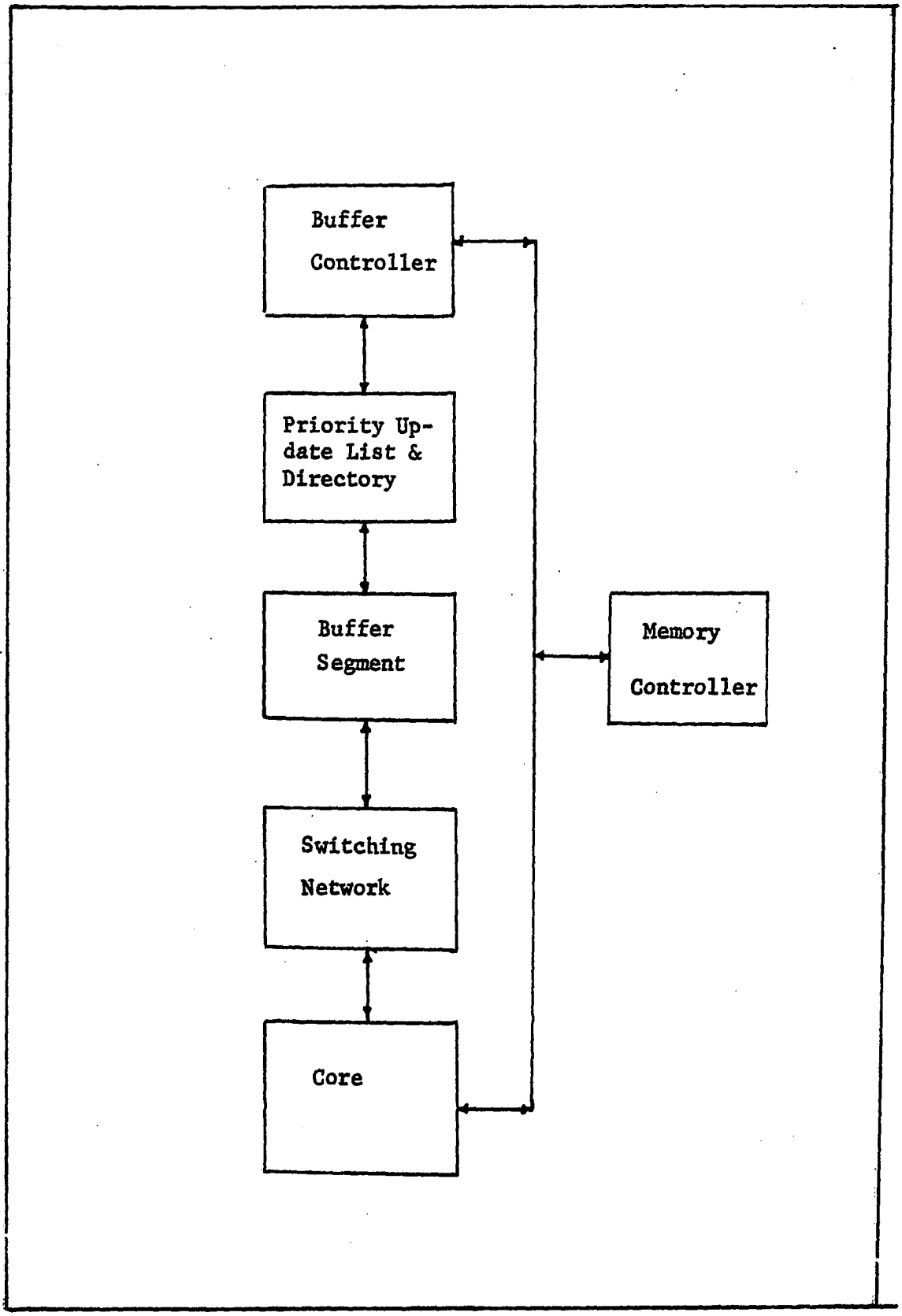

Fig. 37. System Buffer Block Diagram

The buffer memory itself can be thought of as a straight semiconductor memory (organized with relatively long words).

Cost is computed essentially for two types of address translation schemes. Buffer cost is a function of the size of the buffer directory and update list and directory cost is a function of the type of address translation schemes.

The major cost of the buffer memory is the memory itself. Directory, update list and switching network cost are not much. The size of the directory is a function of the address translation scheme. Even though a small variation in the cost of buffer directory will not make much difference to the total cost, the cost analysis is carried out for two different address translation schemes.

Cost Analysis

Let

V = Total number of virtual pages

N = Number of main memory modules

M = Number of pages/module

B = Number of blocks/page

w = Number of words/block

n = Number of blocks in the buffer

f = Number of flag bits/buffer directory word

p = Number of page list bits

Bcost = Buffer cost/bit

Mcost = Main memory cost/bit

Pcost = Priority update list cost/bit

Dcost = Directory cost/bit

Buffer Controller Cost = BC

then

Main Memory Capacity = (N * M * B * w) bits

Buffer Capacity = (n * B * w) bits

As mentioned before the size of the buffer directory would depend upon the type of the addressing scheme.

Address Scheme I:

Using this scheme buffer directory size =

$n * (\log_2 (M*N) + \log_2 N + B + f)$ bits

Address Scheme II:

Using this scheme buffer directory size =

$n * (\log_2 (M*N) + \log_2 N + B + f + \log_2 V)$ bits

Based on this a cost analysis was carried out. Buffer cost for various buffer sizes, for two different block sizes and these two different address schemes were computed and are illustrated in Table I to Table III.

From these tables we see that for buffer size of 64 blocks, and block size of 8 words, the buffer cost varies from about 15% to 2% of the main memory cost for various memory sizes for address scheme I and it varies from about 18% to 2.25% of the main memory costs for address scheme II.

Address scheme II does not cost much more but saves one memory cycle for address transformation.

Table I. Cost Analysis

Main Memory (Core) 0.7¢/bit
Buffer (Semiconductor) 4¢/bit
Buffer Directory 10¢/bit
Priority Update List 10¢/bit

Main Memory		Buffer Size = 64 Blocks							
		Block Size = 8 Words				Block Size = 4 Words			
Capacity	Cost	Fraction of Main		Cost	% of Main	Fraction of Main		Cost	% of Main
32K Words = 256K Bytes	$14,680	1/64	Scheme I	$2484	14.7	1/128	Scheme I	$930	6.35
			Scheme II	$2585	17.6		Scheme II	$1032	7.05
64K Words = 256K Bytes	$29,360	1/128	Scheme I	$2490	8.5	1/256	Scheme I	$937	3.2
			Scheme II	$2591	8.85		Scheme II	$1039	3.54
128K Words = 1,024K Bytes	$58,720	1/256	Scheme I	$2496	4.25	1/512	Scheme I	$944	1.61
			Scheme II	$2600	4.45		Scheme II	$1046	1.78
256K Words = 2,048K Bytes	$117,440	1/512	Scheme I	$2500	2.14	1/1024	Scheme I	$951	0.8
			Scheme II	$2606	2.22		Scheme II	$1053	0.9

Table II. Cost Analysis

Main Memory (Core)	0.7¢/bit
Buffer (Semiconductor)	4¢/bit
Buffer Directory	10¢/bit
Priority Update List	10¢/bit

Main Memory		Buffer Size = 128 Blocks							
		Block Size = 8 Words				Block Size = 4 Words			
Capacity	Cost	Fraction of Main		Cost	% of Main	Fraction of Main		Cost	% of Main
32K Words = 256K Bytes	$14,680	1/32	Scheme I	$3990	27.2	1/64	Scheme I	$1793	12.2
			Scheme II	$4195	28.6		Scheme II	$1997	13.5
64K Words = 512K Bytes	$29,360	1/64	Scheme I	$4005	13.7	1/128	Scheme I	$1806	6.15
			Scheme II	$4210	14.3		Scheme II	$2010	6.85
128K Words = 1,024K Bytes	$58,720	1/128	Scheme I	$4020	6.55	1/256	Scheme I	$1819	3.1
			Scheme II	$4225	7.25		Scheme II	$2023	3.46
256K Words = 2,048K Bytes	$117,440	1/256	Scheme I	$4035	3.45	1/512	Scheme I	$1832	1.5
			Scheme II	$4240	3.53		Scheme II	$2036	1.75

Table III. Cost Analysis

Main Memory (Core) 0.7¢/bit
Buffer (Semiconductor) 4¢/bit
Directory 10¢/bit
Priority Update List 10¢/bit

Main Memory		Buffer Size = 256 Blocks							
		Block Size = 8 Words				Block Size = 4 Words			
Capacity	Cost	Fraction of Main		Cost	% of Main	Fraction of Main		Cost	% of Main
32K Words = 256K Bytes	$14,680	1/16	Scheme I	$7036	48	1/32	Scheme I	$3541	24.2
			Scheme II	$7446	51		Scheme II	$3855	26.2
64K Words = 512K Bytes	$29,360	1/32	Scheme I	$7060	24	1/64	Scheme I	$3567	12.2
			Scheme II	$7470	25.5		Scheme II	$3881	13.0
128K Words = 1024K Bytes	$58,720	1/64	Scheme I	$7085	12.5	1/128	Scheme I	$3593	6.1
			Scheme II	$7495	12.75		Scheme II	$3907	6.6
256K Words = 2,048K Bytes	$117,440	1/128	Scheme I	$7120	6.1	1/256	Scheme I	$3619	3.1
			Scheme II	$7520	6.4		Scheme II	$3933	3.3

Keeping the main memory size fixed and increasing (decreasing) the buffer size increases (decreases) the % of buffer cost (as compared to the main memory cost) linearly.

As was observed in chapter IV, 64 blocks for the whole buffer for SYMBOL-IIR like computing structures would be more than enough for yielding a high hit-ratio. From the cost analysis we see that for a main memory of 256K words or 512K bytes (8 times the capacity of the present SYMBOL-IIR main memory) this buffer of 64 blocks would constitute only about 9% of the main memory cost and still would give hit-ratio very close to unity.

This illustrates the tremendous cost effectiveness of buffering SYMBOL-IIR like computing structures.

CHAPTER VI. CONCLUSION AND DISCUSSIONS

With the advancement of the LSI technology and the persistent decline in the cost of hardware (with a consistent increase in the cost of software) more and more emphasis and thought have been (and will be) given to reexamining the traditional hardware/software boundary. This reexamination has led (and will lead) computer system designers to use as much hardware as possible to reduce the whole system programming cost and to obtain better performance. SYMBOL-IIR computing system is one of the results of this reexamination.

This thesis has explored the applicability of buffering such multidedicated processors, time-sharing systems. It is seen that the architectural organization of the whole system has tremendous impact on the organization and management of its buffered memory systems. Three alternative ways of providing buffer--a buffer for the whole system, buffers for the terminals or buffer for each dedicated processor are investigated and it is demonstrated that allocating a small, dedicated but sharable buffer to its different dedicated processors would result in a significant increase in performance with a very insignificant increase in the cost.

Fig. 25 and Fig. 33 are the most important results of the whole investigation. From Fig. 25 we see that, a buffer of only 2K bytes yields a hit-ratio of 97%. In this case, small dedicated but sharable buffer space is provided for

each dedicated processor. Also from Fig. 25 we see that there is an optimum buffer size of 32 blocks for SYMBOL-IIR like computing structures. Allocating larger buffer sizes to different dedicated processors and thus exceeding the total capacity of 32 blocks does not result in any increase of performance--rather the hit-ratio tends to decrease.

From Fig. 33 we see that providing small additional buffer space for the data linking words and the system and terminal header words results in an incredible hit-ratio--99.7%. The buffer can be visualized now as consisting of a data buffer, a data linking buffer and sytsem and terminal header buffer. Achievement of such high hit-ratio with such a small buffer illustrates the effectiveness of dedicated and sharable buffer for a highly unconventional computing structures like SYMBOL-IIR.

Also it is seen that, for SYMBOL-IIR like computing structures, the behaviour (i.e., the address reference pattern) of processors is tremendously affected by the storage organization and management of its virtual memory system and it is seen from Fig. 17 and Fig. 21 that the address referencing pattern is more important than the total amount of activity of the processor. This observation leads to one of the very important conclusion that the over all hit-ratio would not be affected very much by the variation of the users environments. Depending upon the size and type of programs,

the total amount of activity of different processors might vary, but as the basic behaviour of different dedicated processors would essentially remain the same, there would be only small perturbation on the over all hit-ratio. Because of this important observation, even though the experimental investigation was limited only to small scientific users environments, we can boldly predict that the results obtained in this investigation are extendable to other different environments. However for conventional computer systems, the hit-ratio data is affected very strongly by the behaviour of programs and users environments (2,14,17,18).

Also it is shown that for SYMBOL-IIR like computing structures, besides the principle of "locality" its architectural organization and over all storage management has significant impact upon the organization and management of its buffered memory--and hence the results of this investigation could be extendable to future computer systems consisting of multi homogeneous or heterogeneous processors--as long as those systems would have the similar storage management principles of SYMBOL-IIR system.

This concept of small, multidedicated buffer for multidedicated processors for SYMBOL-IIR like computing structures leads also to one of the very important system concepts that--instead of all the dedicated processors sharing the same common memory, if small dedicated fast memories

are provided for these processors and if the auxiliary memory is quite fast enough to transfer data at reasonable fast rate, then the whole main memory could be completely eliminated.

Now according to many computer experts, P and N channel MOS will dominate the computer main frame memories, taking nearly 60% of total bits by the late 1970s. Core which had 100 percent of main frame memory in 1960 had declined to about 79% in 1972, about 65% in 1973 and is projected to have less than 10% by 1980.

Even if MOS memory replaces core as the main frame memory, for larger systems the hybrid approach--or the approach of buffering--a slow large MOS memory supported by a fast, small bipolar memory would seem to be the best solution.

At least for another decade, the approach of buffering is going to stay and if future systems tend to achieve multiprocessing by the use of multidedicated processors--then separate dedicated buffers for these dedicated processsors would be the best and cheapest way of improving the performance.

But it should be mentioned that buffering is no solution to a virtual memory system which is limited by the page traffic between its main memory and the auxiliary memory. Hence when we are talking about buffering a virtual memory system like SYMBOL-IIR we are implying that the system has the

provision of a large main memory and the system performance is not limited by the page traffic but the speed of the main memory. In systems like that buffering is the cheapest and most effective way for improving the performance at very insignificant increase in cost.

ACKNOWLEDGMENTS

I am deeply indebted to my major professor Dr. Arthur V. Pohm for introducing me to the subject of buffered memory systems and for his constant advice, guidance and encouragement during the preparation of this dissertation.

I would also like to express my deep gratitude to Dr. Roy J. Zingg for giving me an opportunity to work on the SYMBOL-IIR project and for his valuable suggestions and encouragements during the pursuit of this investigation.

I have also benefitted greatly from conversations with numerous colleagues involved in the SYMBOL-IIR project. In particular special mention must be made of Perry Hutchison who besides investing his time in reading this thesis and giving some pertinent comments was immensely helpful in running the experimental simulation work. I am thankful to other members of the SYMBOL-IIR project for their kind cooperation for letting me use the system for a horrendous amount of time.

I am also very grateful to Ralph Luckeroth and Gary Andrews for their big help in constructing the interface unit.

I would like to express my appreciation to the Engineering Research Institute of Iowa State University and

National Science Foundation for supporting my research work.

Last, but not the least, I am extremely grateful to my parents and all the other members of the family, back in India, for their love, encouragement, patience and understanding.

BIBLIOGRAPHY

1. Arora, S. R., and Wu, F. L. "Statistical Quantification of Instruction and Operand Traces." Statistical Computer Performance Evaluation. Edited by Walter Freiberger. New York: Academic Press, Inc., 1972.

2. Baer, J. L., and Sager, G. R. "Measurement and Improvement of Program Behaviour under Paging Systems." Statistical Computer Performance Evaluation. Edited by Walter Freiberger. New York: Academic Press, Inc., 1972.

3. Bell, C. G., and Casasent, D. "Implementation of a Buffer Memory in Mini-Computers." Computer Design, (November, 1971), 83-89.

4. Bell, J., Casasent, D., and Bell, C. G. "An Investigation of MiniComputers Cache Scheme." RCA Report R-72-190.

5. Bersamian, H., and Decegama, A. L. "System Design Considerations of Cache Memories." Con-Comp Conference, (1972), 107-110.

6. Belady, L. A. "A Study of Replacement Algorithms for Virtual Storage Computers." IBM Systems Journal, 5, No. 2 (1966), 78-101.

7. Belady, L. A. "Biased Replacement Algorithms for Multiprogramming." Report NC697. Yorktown Heights, New York: IBM, T. J. Watson Research Center, March, 1967.

8. Bloom, L., Cohen, M., and Porter, S. "Consideration in the design of a Computer with high logic to memory speed ratio." Proc. Gigacycle Computing Systems, AIEE Special Publication S-136, (Jan 29 - Feb 2, 1962), 53-63.

9. Cohen, C. "Japan is packing everything into large Computer Project." Electronics, 44, No. 11 (May 24, 1971), 42-49.

10. Conti, C. J. "Concepts of Buffer Storage." IEEE Computer Group News, 2, No. 5 (March, 1969), 6-13.

11. Denning, P. J. "The Working Set Model of Program Behaviour." Comm. ACM, 11, No. 5 (May, 1968), 323-333.

12. Denning, P. J. "Resource Allocation in MultiProcess Computer Systems." Tech. Report. MAC-TR-50, MIT Project

MAC, Cambridge, Mass., 1968.

13. Denning, P. J. "Thrashing : its Causes and Prevention." Proc. AFIPS Fall Joint Comput. Conf., 33 (1968), 915-922.

14. Fine, G. H., Jackson, C. W., and McIssac, P.V. "Dynamic Program Behaviour under Paging." National Conf. ACM Proc., ACM Publications P-66, 21 (1966), 223-228.

15. Gibson, D. H. "Considerations in Block Oriented Systems Design." Proc. AFIPS Spring Joint Comput. Conf., 30 (1967), 75-80.

16. Gibson, D. H., and Shevel, W. L. "'Cache' turns up a Treasure." Electronics, 42, No. 11 (October 13,1969), 105-107.

17. Hatfield, D. J. "Program Restructuring for Virtual Memory." IBM Systems Journal, 10, No. 3 (1971), 168-192.

18. Hatfield, D. J. "Some Experiments on the Relationship Between Page Size and Program Access Pattern." IBM Journal of Res. and Dev., 16, No. 1 (January,1972), 58-66.

19. Kaplan, K. R., and Winder, R. O. "Cache Based Computer Systems." Computer, 10, No. 3 (March,1973), 30-36.

20. Kilburn, T., Edwards, D. B. G., Lanigan, M. G., and Sumner, F. H. "One level Storage System." IRE Trans. on Computer, EC-11, April, 1962, 223-238.

21. Lee, F. F. "Look Aside Memory Implementation." Memorandum, MAC-M-99, August,1963, 1-3.

22. Lee, F. F. "Look Aside Memory Simulation." Memorandum, MAC-M-131, January,1964,1-4.

23. Lee, F. F. "Study of Look Aside Memory." IEEE Trans. on Computers, 18, No. 11 (November,1969), 1062-1064.

24. Liptay, J. S. "Structural Aspects of the System/360 Model 85: II The Cache." IBM Systems Journal, 7, No. 1 (1968), 15-21.

25. Madnick, S. E. "Storage Hierarchy Systems." Tech. Report. MAC-TR-117, MIT Project MAC, Cambridge, Mass., 1973.

26. Mattson, R. L. "Evaluation of Multi Level Memories." IEEE Trans. on Magnetics, 7, No. 4 (December,1971), 814-819.

27. Mattson, R. L., and Traiger, I. L. "Storage Hierarchy Design." IEEE Trans. on Magnetics, 7, No. 3 (September, 1971), 145-148.

28. Mattson, R. L., Gecsci, J., Slutz, D. R., and Traiger, I. L. "Evaluation Techniques for Storage Hierarchies." IBM Systems Journal, 9, No. 2 (1970), 78-117.

29. Meade, R. M. "On Memory System Design." Proc. AFIPS Fall Joint Comput. Conf., 37 (1970), 33-43.

30. Meade, R. M. "Design Approaches for Cache Memory Control." Computer Design, (January,1971), 87-93.

31. Meade, R. M. "How a Cache Memory Enhances a Computers Performance." Electronics, 45, No. 2 (January,1972), 58-63.

32. Nisenoff, N. "Hardware for Information Processing Systems: today and in the Future." Proc. IEEE, 54, No. 12 (December,1966), 1820-1835.

33. Pohm, A. V. "Competitive High Speed Memory Technologies." IEEE Trans. on Magnetics, 8, No. 4 (December,1972), 888-893.

34. Pohm, A. V., Agrawal, O. P., Cheng, C. W., and Shimp, A. "An Efficient Flexible Buffered Memory System." IEEE Trans. on Magnetics, 9, No. 3 (September,1973), 173-179.

35. Pohm, A. V., Agrawal, O. P., and Cheng, C. W. "FabritTek Buffered Memory Study." Interim Report. Project 963-s. Iowa State Univ., August, 1972.

36. Rice, R., and Smith, W. R. "Symbol - A Major Departure From Classic Software Dominated Computing Systems." Proc. AFIPS Spring Joint Comput. Conf., 38 (1971), 575-587.

37. Richards, H. Jr, and Zingg, R. J. "The Logical Structure of the Memory Resource in the SYMBOL2R Computer." Special Report. NSF-OCA-GJ33097-CL7307. Cyclone Computer Laboratory. Iowa State University, November, 1973.

38. Smith, W. R., Rice, R., Chesley, G. D., Laliotis, T. A.,

Lundstrom, S. F., Calhoun, M. A., Gerald, L. D., and Cook, T. G. "SYMBOL-A Large Experimental System Exploring Major Hardware Replacement of Software." Proc. AFIPS Spring Joint Comput. Conf., 38 (1971), 601-616.

39. Takahashi, S., Nishino, H., Yoshihiro, K., and Fuchi, K. "System Design of the ETL Mk-6 computer." Proc. IFIPS Congress, (1962), 690-693.

40. Traiger, I. L., and Mattson, R. L. "The Evaluation and Selection of Technologies for Computer Storage Systems." Proc. of AIP Conf. on Magnetism and Magnetic Materials, 17, No. 5 (1972), 1-12.

41. Traiger, I. L., and Mattson, R. L. "One Pass Techniques for the Evaluation of Memory Hierarchies." IBM Research Report, RJ-892, 1971.

42. Tsao, R. F., Comeau, L. W., and Marolin, B. H. "A Multi Factor Paging Experiment: I The Experiment and Conclusions." Statistical Computer Performance Evaluation. Edited by Walter Freiberger. New York: Academic Press, Inc., 1972.

43. Tsao, R. F., and Margolin, B. H. "A Multi Factor Paging Experiment: II Statistical Methodology." Statistical Computer Performance Evaluation. Edited by Walter Freiberger. New York: Academic Press, Inc., 1972.

44. Wilkes, M. V. "Slave Memories and Dynamic Storage Allocation." IEEE Trans. on Electronic Computers, 14 (1965), 270-271.

45. Zingg, R. J., and Richards, H. Jr. "SYMBOL: A System Tailored to the Structure of Data." Special Report. ISU-CCL-7302. Cyclone Computer Laboratory. Iowa State University, January, 1973.