MEMORY REQUEST SCHEDULING IN MULTIPROCESSOR MULTI-MEMORY SYSTEMS

by

Yu-Huei Jea

August 1976 TR-61

This research constituted a Master's Thesis presented to the Faculty of the Graduate School of The University of Texas at Austin.

DEPARTMENT OF COMPUTER SCIENCES
THE UNIVERSITY OF TEXAS AT AUSTIN

TABLE OF CONTENTS

1. INTRODUCTION

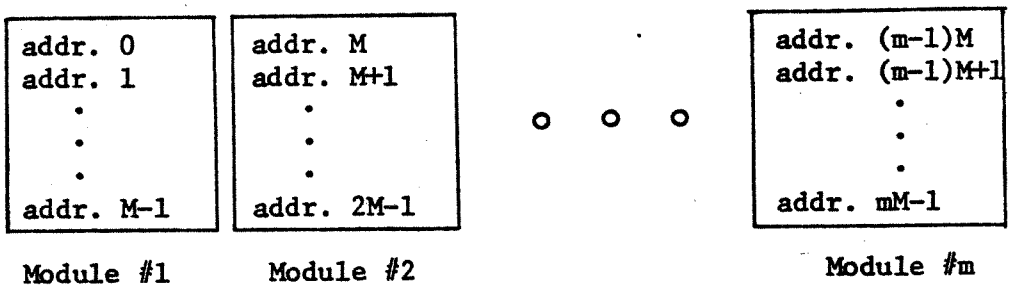
Memory access has always been a limiting factor to the execution speed of high performance computers. The use of multiprocessor architecture (implicit as for independent I/O channel operations or explicit as for multiple processing units) has intensified the need for larger bandwidth in memory system architecture. This paper defines a memory access subsystem which maximizes (or nearly maximizes) the average bandwidth of a multiple-module memory system (or multi-memory system for short).

There have been two principal approaches to obtaining large high performance memory systems at a reasonable cost. The multi-level hierarchy or vertical approach [1,2,3] uses a memory system with components of different speeds. Typically there will be two or three levels with access time increasing with relative size. Data and instructions will be moved to the fast access level on first use and fetched from the fast access level on subsequent uses (unless displaced). The effectiveness of this organization depends upon the tendency of programs to operate in restricted or local portions of their total address space for time periods much longer than memory fetch and instruction execution times. When fast memory is adequate to hold the "working sets" [4] of the active processes, then the effective memory system bandwidth is close to that of the fast component. The multiple-module or "horizontal" approach decomposes the executable memory into a number, m, of independently

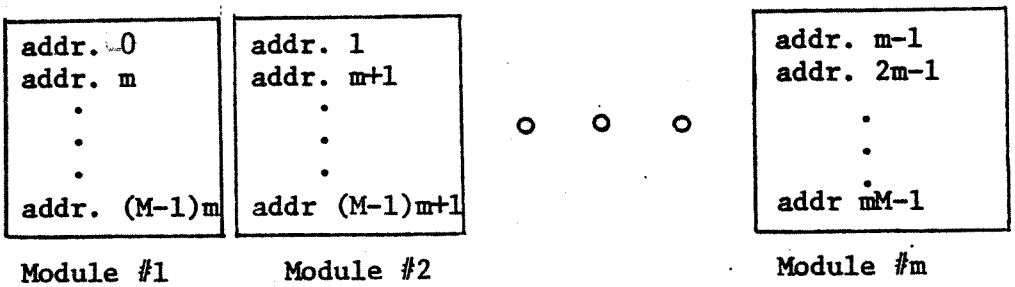
functioning modules. The bandwidth of the memory system is then potentially much greater than the bandwidth of a single element and is upper bounded by m times the bandwidth of a single module. The performance of a multiple-module memory organization will be high when serial correlation of references to each module is low.

There are several reasons for an increased interest in multiple-module memory organizations. LSI fabrication is lowering the cost of independent driving circuitry. The multiple-module organization can be more effective at lowering the referencer conflict problem for multiprocessor organizations. The memory organizations described here are assigned to exploit the possibility of increased complexity in driving circuitry to resolve the referencer conflict problem for multiprocessor systems.

There are two modes of address assignment for multiple-module memory systems. One is to assign the address space consecutively within each module. The second is to assign the address consecutively across all modules. The first is called a multiple independent memory system and the second is called an interleaved memory system. Figures 1a and 1b illustrate the two assignment modes. Let m be the number of memory modules and M the number of words in each module. Then the address space in any one module of a multiple independent memory system is a complete set of residues modulo M while all addresses in one module



(a) Multiple independent memory system



(b) Interleaved memory system

Figure 1. Multi-memory system addressing

of an interleaved memory system are all congruent modulo m.

There are a number of possible approaches to the reduction of the reference conflict problem. Kurtzberg [5] has analyzed the effects of assignment of jobs to the memory modules of a multiprocessor system. Coffman and Burnett [6,7] have analyzed the effects of separate request queues for single-processor interleaved memory systems. These two approaches both attempt to prevent the occurrence of conflict by controlling the contents of the request queue for memory references.

The approach taken here is to optimize the performance of a multiple-module memory system when the request queues for memory references are given. Consider a multiple module memory system with a reference request queue for each independent module. This research determined, described, and tested algorithms for selecting, at each memory cycle, references from these request queues which maximize or nearly maximize total memory system bandwidth. There were found several straight-forward algorithms which yield substantial bandwidth enhancement for random reference workloads.

The measure employed to evaluate memory system performance is <u>average bandwidth</u>, i.e. the average number of modules in operation per memory cycle. The number of memory modules which can be in operation in a given system is bounded either by the total number of memory modules, m, or by p x n where p is the number of processors and n is the ratio of memory cycle time to processor cycle time.

$$n = \frac{\text{memory cycle time}}{\text{processor cycle time}}$$

The two upper bounds are considered separately. The case of p x n bounds is referred to as bandwidth under restriction to indicate that the memory system may separately have a greater bound for bandwidth.

Section 2 gives needed definitions and establishes the notation. Section 3 defines the memory conflict problem which is to be analyzed. Algorithms for the two-processor m-memory module case are described in Section 4 and their performance evaluated in Section 5. Section 6 considers the two processor case under restriction. Section 7 extends the algorithms to numbers of processors greater than 2 and Section 8 evaluates the performance of these algorithms. A possible scheme for physical implementation of the class of algorithms studied here is given in Section 9. Section 10 summarizes the results obtained and suggests related problems requiring further research.

2. DEFINITIONS AND NOTATIONS

Let there by p processors each with processor cycle time T_p and m memory modules each with memory cycle time T_m. A request for memory module access made by some processor is, for simplicity, represented only by the module number. Thus a request r is an element of the set $\{1,2,\ldots,m\}$.

Consider the single processor case first. There is a request queue Q generated by this processor. It is represented by a vector of requests $(r_1,r_2,\ldots,r_\ell,\ldots)$. When no confusion is possible, the parentheses and commas can be ignored and written in more compact form as $r_1r_2\ldots r_\ell\ldots$.

Definition: A request sequence, denoted by Γ, of the given request queue is a finite sequence of requests $r_1r_2\ldots r_\ell$ such that

(a) $i\neq j$ implies $r_i\neq r_j$ for all i,j, $1\leq i, j\leq \ell$

(b) $r_{\ell+1} = r_i$ for some i, $1\leq i\leq \ell$

A request subsequence, denoted by Γ_i, of request sequence Γ is defined to be the sequence $r_1r_2\ldots r_i$ with $0\leq i\leq \ell$. For i=0, it means the null sequence, i.e., $\Gamma_0=\emptyset$. Note that $\Gamma_\ell=\Gamma$.

The length of a request sequence (or subsequence) is the total number of elements in it. It is expressed by the notation $\ell g(\cdot)$. The length of a null sequence is zero.

The ground universe of a request sequence Γ is the collection of all its possible request subsequences. Thus the ground universe

$$G(\Gamma) = \{\Gamma_0, \Gamma_1, \ldots, \Gamma_\ell\}, \ \ell = \ell g(\Gamma)$$

Example. For m=8, request queue might be

$$Q = 2714135\ldots$$

Then its request sequence is

$$\Gamma = 2714$$

and the length of Γ is

$$\ell g(\Gamma) = 4$$

The ground universe of Γ is

$$G(\Gamma) = \{\Gamma_0, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4\}$$
$$= \{\emptyset, 2, 27, 271, 2714\}$$

Let us now extend the preceding definitions to the multiprocessor cases. Let $Q^1, Q^2, \ldots Q^p$ be the request queues for each processor, and their corresponding request sequences be $\Gamma^1, \Gamma^2, \ldots \Gamma^p$.

Definition: The request sequence set Λ is the collection of all request sequences for all processors. Thus

$$\Lambda = \{\Gamma^1, \Gamma^2, \ldots, \Gamma^p\}$$

The request subsequence set of Λ is a set

$$\{\Gamma^1_{i_1}, \Gamma^2_{i_2}, \ldots, \Gamma^p_{i_p}\}$$

where

$$\Gamma^k_{i_k} \in G(\Gamma^k), \ 1 \leq k \leq p.$$

The length of a request sequence (or subsequence) set is defined to be the sum of the individual lengths of each element in the set, i.e.,

$$\ell g(\{\Gamma^1_{i_1}, \Gamma^2_{i_2}, \ldots, \Gamma^p_{i_p}\}) = \sum_{k=1}^{p} \ell g(\Gamma^k_{i_k})$$

There exists memory conflict if more than one processor requests a memory module during a given memory cycle time.

Definition: The request subsequence (or sequence) set $\{\Gamma^1_{i_1}, \Gamma^2_{i_2}, \ldots, \Gamma^p_{i_p}\}$ is in <u>conflict</u> if there exists a request r such that

$$r \in \Gamma^{k_1}_{i_{k_1}} \quad , \quad k_1 \in \{1,2,\ldots,p\}$$

and

$$r \in \Gamma^{k_2}_{i_{k_2}} \quad , \quad k_2 \in \{1,2,\ldots,p\}, \; k_1 \neq k_2.$$

A <u>feasible solution</u> $S(\Lambda)$ of a request sequence set Λ is a request subsequence set of Λ such that there exists no conflict.

An <u>optimal solution</u> $S_*(\Lambda)$ of a request sequence set Λ is the feasible solution with maximal length.

Example: For p=2 and m=8, the request sequence set might be

$$\Lambda = \{35841, 2714\}.$$

There are a number of feasible solutions, for example

$$S(\Lambda) = \{35, 2\}$$

or $$S(\Lambda) = \{358, 271\}$$

or $$S(\Lambda) = \{358, 2714\}$$

etc.

The optimal solution is

$$S_*(\Lambda) = \{358, 2714\}$$

Note that the optimal solution need not be unique. In the above example, there are two other optimal solutions which are {35841, 27} and {3584, 271}.

The optimal solution maximizes the average bandwidth.

The following mathematical notations are adopted in this paper:

$\lfloor x \rfloor$ The floor of x. For example, $\lfloor 2 \rfloor = 2$, $\lfloor 2.99 \rfloor = 2$.

$\lceil x \rceil$ The ceiling of x. For example, $\lceil 2 \rceil = 2$, $\lceil 2.01 \rceil = 3$.

$\sum_{i=i_1}^{i_2} a_i$ The sum of all a_i for which the integer i satisfies $i_1 \leq i \leq i_2$. If no such integer i exists, the sum is defined to have the value of zero.

$\pi_{i=i_1}^{i_2} a_i$ The product of all a_i for which the integer i satisfies $i_1 \leq i \leq i_2$. If no such integer i exists, the product is defined to have the value unity.

P_ℓ^n The permutation function which is defined to be $n \cdot (n-1) \cdot \ldots \cdot (n-\ell+1)$ for $1 \leq \ell \leq n$ and 1 otherwise.

$\Gamma^i \cdot \Gamma^j$ The concatenation of two request sequences. Thus if $\Gamma^i = r_1^i \ldots r_{\ell_i}^i$ and $\Gamma^j = r_1^j \ldots r_{\ell_j}^j$, then $\Gamma^i \cdot \Gamma^j = r_1^i \ldots r_{\ell_i}^i r_1^j \ldots r_{\ell_j}^i$. Note that $\Gamma \cdot \emptyset = \emptyset \cdot \Gamma = \Gamma$ and $\emptyset \cdot \emptyset = \emptyset$.

3. PROBLEM MODELING

The following four conditions define the problem domain of the subsequent analyses.

a. All processors and memory modules are synchronized.

b. The requests in any request queue are independent, thus there is no relationship between any two elements in one request queue.

c. The processors make the requests independently. Thus there is no relationship between any two requests, one from one request queue and the other from any other request queue.

d. The system operates at full load such that the request queues always contain sufficient requests to determine their request sequences.

The goal is to find a procedure which selects requests to be serviced from each request queue in such a way that the average bandwidth will be maximal. This is in principle a global optimal problem, i.e., overall execution time must be analyzed. Fortunately it can be established that local optimization over every memory cycle generates global optimality.

Optimality Principle: Local optimality implies global optimality.

Proof: Let the contents of all request queues characterize the

state of the system. Two states are said to be equivalent as long as every request in all request queues are independent. Since the system transits from one state into the next state after every memory cycle, the system behavior can be represented by a sequence of transitions

$$S_0 \rightarrow S_1, S_1 \rightarrow S_2, \ldots, S_i \rightarrow S_{i+1}, \ldots$$

From the conditions of problem definition, it is easy to see that $S_0, S_1, S_2, \ldots$ should be equivalent states, that is

$$S_0 = S_1 = S_2 = \ldots$$

Let $B_{S_i \rightarrow S_{i+1}}$ be the bandwidth produced at transition $S_i \rightarrow S_{i+1}$. Also let $B_{S_0 \rightarrow S_i}$ be the sum of all bandwidths at all transitions $S_0 \rightarrow S_1, S_1 \rightarrow S_2, \ldots, S_{i-1} \rightarrow S_i$, i.e., $B_{S_0 \rightarrow S_i} = \sum_{k=0}^{i-1} B_{S_k \rightarrow S_{k+1}}$.

Then $B_{S_0 \rightarrow S_{i+1}} = B_{S_0 \rightarrow S_i} + B_{S_i \rightarrow S_{i+1}}$.

Now we are ready to prove that local optimality yields global optimality. We use mathematical induction.

For i=0, the whole process is just $S_0 \rightarrow S_1$. It is trivial that local optimality implies global optimality.

Next, we make the following induction hypothesis:

For i=n, $B_{S_{n-1} \rightarrow S_n}$ is optimal implies $B_{S_0 \rightarrow S_n}$ is optimal.

Consider i = n + 1 case.

Since $B_{S_0 \rightarrow S_n}$ is maximal (by hypothesis), $B_{S_0 \rightarrow S_{n+1}}$ would not be

optimal unless $B_{S_n \rightarrow S_{n+1}}$ is optimal. Thus the case of $i = n + 1$ is true.

Therefore it is always true that local optimal will imply global optimal under our previous assumptions.

Q.E.D.

It is quite easy to see that the local optimal problem is to find an optimal solution from the request sequence set of every memory cycle. Figure 2 gives a schematic of our model.

The function of the scanner is to determine the request sequence from its input request queue [7]. The outputs of all scanners form the request sequence set Λ, which is the input to the selector. The selector properly selects requests from each request sequence to give an optimal solution as its output. We can, by this means, obtain maximal bandwidth for every memory cycle; this results in maximal average bandwidth for the system.

Although the queue lengths would be unbounded in our model, they are always bounded in the real world. When some queues become full, the memory system will issue some commands to halt the corresponding processors. We also notice that in some computer systems the order which the memory requests are awarded to the processors may be different from the order in which they are generated.

We consider now algorithms for selecting optimal request sequences for two processor m memory module systems.

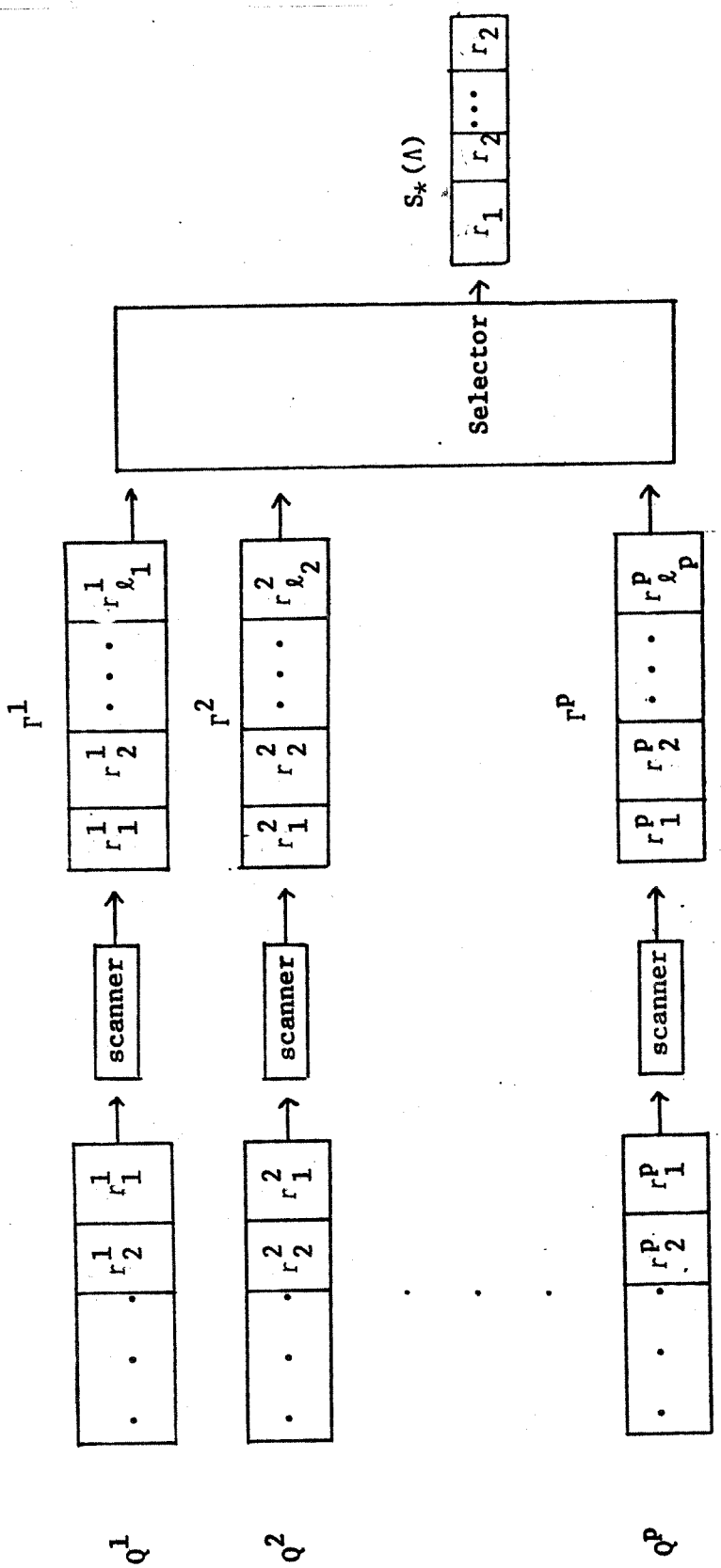


Figure 2. Model Picture

4. OPTIMAL SOLUTION FOR TWO-PROCESSOR SYSTEM

Given a request sequence set of two elements

$$\Lambda = \{\Gamma^1, \Gamma^2\}$$

where

$$\Gamma^1 = r_1^1 r_2^1 \cdots r_{\ell_1}^1$$

$$\Gamma^2 = r_1^2 r_2^2 \cdots r_{\ell_2}^2$$

we want to determine algorithms which will yield at least one of the optimal solutions.

Let us first consider exhaustive search. We construct all request subsequence sets

$$\Lambda_{i_1 i_2} = \{\Gamma_{i_1}^1, \Gamma_{i_2}^2\},\ 0 \le i_1 \le \ell_1,\ 0 \le i_2 \le \ell_2.$$

Then the optimal solution is obtained by selecting the request subsequence set which has longest length for which there exists no conflict. If there is a tie, any arbitrarily chosen subsequence set will be all right.

Exhaustive search is very tedious and impractical because there are a total of $(\ell_1 + 1)\ (\ell_2 + 1)$ request subsequence sets to be considered. Therefore it is used only to show theoretically that the optimal solution of any given request sequence set does exist and can be found. There is no value for practical implementation.

We now consider an iterative approach which just constructs an approximate solution and defines an iterative improvement procedure.

Given a request sequence set of two elements $\Lambda = \{\Gamma^1, \Gamma^2\}$, its trivial approximate solution $S_t(\Lambda)$ is defined by the following algorithm.

Algorithm 1: Algorithm to find trivial approximate solution of $\Lambda = \{\Gamma^1, \Gamma^2\}$ where $\Gamma^1 = r_1^1 r_2^1 \ldots r_{\ell_1}^1$ and $\Gamma^2 = r_1^2 r_2^2 \ldots r_{\ell_2}^2$.

Step 1 [initialize] Set $j = 1$

Step 2 [check for conflict] If $r_j^2 \in \Gamma^1$, go to step 4.

Step 3 [set up next iteration] Set $j = j + 1$. If $j \leq \ell_2$, go to step 2.

Step 4 [answer] Set $S_t(\Lambda) = \{\Gamma_{\ell_1}^1, \Gamma_{j-1}^2\}$

Example: Given $\Lambda = \{35841, 2714\}$

Then $S_t(\Lambda) = \{35841, 27\}$

which happens to be the optimal solution as has been shown in a previous example.

Consider another example.

Example: Given $\Lambda = \{12345, 5678\}$

Then $S_t(\Lambda) = \{12345, \emptyset\}$

It is clear that this result is not optimal since the optimal solution is

$S_*(\Lambda) = \{1234, 5678\}$

How can we start from the trivial approximate solution and promote step-by-step to the optimal solution?

Let us consider the request subsequent set

$$\Lambda_1 = \{\Gamma^1_{\ell_1 - 1}, \Gamma^2_{\ell_2}\},$$

i.e., look one request ahead from the end of first sequence (see Figure 3), and get the corresponding trivial approximate solution in the first iteration. For convenience, we call this iteration as lookahead = 1. The solution after lookahead = 1 should be the longer of $S_t(\Lambda)$ and $S_t(\Lambda_1)$. To be consistent, the trivial approximate solution of original request sequence set Λ, written as Λ_0, will be called the solution after lookahead = 0.

After lookahead = K, the request subsequence sets being considered are

$$\Lambda_i = \{\Gamma^1_{\ell_1 - i}, \Gamma^2_{\ell_2}\} \quad , \quad i = 0, 1, \ldots, K$$

The solution after lookahead = K, $S_K(\Lambda)$, is defined as

$$S_K(\Lambda) = S_t(\Lambda_k), \ 0 \leq k \leq K$$

where k is defined by

$$\ell g(S_t(\Lambda_k)) = \max\{\ell g(S_t(\Lambda_0)), \ell g(S_t(\Lambda_1)), \ldots, \ell g(S_t(\Lambda_K))\}$$

The preceding definition depends upon how we assign the request sequences to Γ^1 and Γ^2. For example, let the two request sequences be 12345 and 5631. If we assign Γ^1=12345 and Γ^2=5631, then

$$S_0(\Lambda) = \{12345, \emptyset\}, \ell g(S_0(\Lambda)) = 5$$

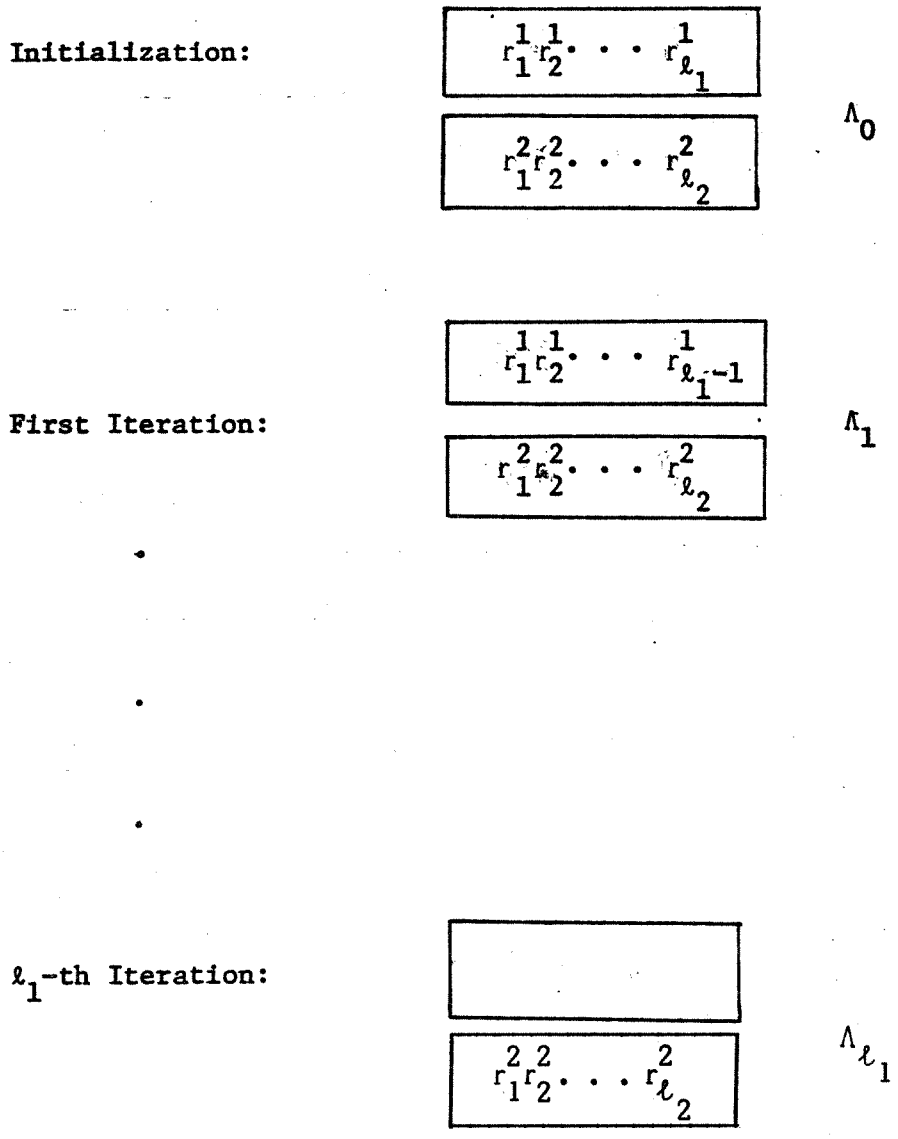


Figure 3. Lookahead Process

and $S_1(\Lambda) = \{1234, 56\}$, $\ell g(S_1(\Lambda)) = 6$

If we assign $\Gamma^1 = 5631$ and $\Gamma^2 = 12345$, then

$$S_0(\Lambda) = \{5631, \emptyset\}, \ \ell g(S_0(\Lambda)) = 4$$

and $S_1(\Lambda) = \{563, 12\}$, $\ell g(S_1(\Lambda)) = 5$

Therefore, in order to unify our presentation, we always name the request sequences in such a way that $\ell_1 \geq \ell_2$.

The termination of iterations will naturally arrive after lookahead $= \ell_1$, the length of first request sequence.

We now proceed to establish that the solution after lookahead $= \ell_1$ is optimal. We first introduce the following lemma.

Lemma 1: The solution after lookahead = k cannot be worse than the solution after lookahead = k - 1. Thus

$$\ell g(S_{k-1}(\Lambda)) \leq \ell g(S_k(\Lambda))$$

Proof: $\ell g(S_{k-1}(\Lambda)) = \max\{\ell g(S_t(\Lambda_0)), \ldots, \ell g(S_t(\Lambda_{k-1}))\}$

$$\ell g(S_k(\Lambda)) = \max\{\ell g(S_t(\Lambda_0)), \ldots, \ell g(S_t(\Lambda_k))\}$$

Since $\max\{a_0, \ldots, a_{k-1}\} \leq \max\{a_0, \ldots, a_k\}$ for any $a_0, \ldots, a_k$, then $\ell g(S_{k-1}(\Lambda)) \leq \ell g(S_k(\Lambda))$. Q.E.D.

Theorem 1: The solution after lookahead $= \ell_1$ is optimal.

Proof: Assume there exists one feasible solution.

$$S(\Lambda) = \{ r_1^1 r_2^1 \ldots r_{i_1}^1, r_1^2 r_2^2 \ r_{i_2}^2 \}, \ 0 \leq i_1 \leq \ell_1, \ 0 \leq i_2 \leq \ell_2$$

which is better than $S_{\ell_1}(\Lambda)$, i.e.,

$$\ell g(S(\Lambda)) > \ell g(S_{\ell_1}(\Lambda))$$

Let $S_t(\Lambda_{\ell_1 - i_1}) = \{r_1^1 \ldots r_{i_1}^1, r_1^2 \ldots r_j^2\}$, $0 \le j \le \ell_2$

Case 1 $i_2 \le j$

This gives $\ell g(S_t(\Lambda)) \le \ell g(S_t(\Lambda_{\ell_1 - i_1}))$

Since $\ell g(S_t(\Lambda_{\ell_1 - i_1})) \le \ell g(S_{\ell_1 - i_1}(\Lambda))$ (by definition)

and $\ell g(S_{\ell_1 - i_1}(\Lambda)) \le \ell g(S_{\ell_1}(\Lambda))$ (by Lemma 1)

then $\ell g(S(\Lambda)) \le \ell g(S_{\ell_1}(\Lambda))$

which contradicts the assumption.

Case 2 $i_2 > j$

According to Algorithm 1, r_{j+1}^2 should belong to $\Gamma_{i_1}^1$. Thus $\{r_1^1 \ldots r_{i_1}^1, r_1^2 \ldots r_j^2 r_{j+1}^2 \ldots r_{i_2}^2\}$ is not a feasible solution which also contradicts the assumption.

Therefore no such feasible solution exists, i.e., $S_{\ell_1}(\Lambda)$ is an optimal solution. Q.E.D.

We can, by this approach, cut our search space to $\ell_1 + 1$, i.e., consider only the request subsequence sets $\Lambda_0, \Lambda_1, \ldots, \Lambda_{\ell_1}$.

We next further shrink our search space by introducing the following

lookahead principle.

Theorem 2: (Lookahead Principle) Optimal solution of $\Lambda = \{\Gamma^1, \Gamma^2\}$, $\ell_1 \geq \ell_2$ is equal to the solution after lookahead $= \ell_2 - 1$.

Proof: By the construction of Algorithm 1

$$\ell g(S_t(\Lambda_\ell)) \leq \ell g(\Gamma^1_{\ell_1 - \ell}) + \ell g(\Gamma^2_{\ell_2}),\ 0 \leq \ell \leq \ell_1$$

or $$\ell g(S_t(\Lambda_\ell)) \leq (\ell_1 - \ell) + \ell_2$$

When $\ell \geq \ell_2$, it becomes

$$\ell g(S_t(\Lambda_\ell)) \leq \ell_1$$

Also we have

$$\ell g(S_t(\Lambda_0)) \geq \ell g(\Gamma^1) = \ell_1$$

Then $$\max\{\ell g(S_t(\Lambda_0)), \ldots, \ell g(S_t(\Lambda_{\ell_2 - 1})), \ldots \ell g(S_t(\Lambda_{\ell_1}))\}$$

$$= \max\{\ell g(S_t(\Lambda_0)), \ldots, \ell g(S_t(\Lambda_{\ell_2 - 1}))\}$$

That is $$S_*(\Lambda) = S_{\ell_2 - 1}(\Lambda)$$ Q.E.D.

The lookahead algorithm can be summarized as follows:

Algorithm 2: Lookahead method to find optimal solution or approximate solution of $\Lambda = \{\Gamma^1, \Gamma^2\}$ where $\Gamma^1 = r^1_1 r^1_2 \ldots r^1_{\ell_1}$ and $\Gamma^2 = r^2_1 r^2_2 \ldots r^2_{\ell_2}$. Note that K is the value of lookahead. When $K = \min(\ell g(\Gamma^1), \ell g(\Gamma^2)) - 1$, the algorithm gives optimal solution.

Step 1 [initialize] Set i = 0. If $\ell_1 < \ell_2$, rename Γ^1 to Γ^2 and

Γ^2 to Γ^1. (Thus $\ell_1 \geq \ell_2$ is always true in the following steps.)

Step 2 [determine K] Assign the value of K. If $K \geq \ell_2$, set $K = \ell_2 - 1$.

Step 3 [solution after K = 0] Compute $S_t(\Lambda)$. Set $S_K(\Lambda) = S_t(\Lambda)$. If K = 0, return.

Step 4 [iteration begin] Set i = i + 1. Compute $S_t(\Lambda_i)$.

Step 5 [compare] If $\ell g(S_K(\Lambda)) < \ell g(S_t(\Lambda_i))$, set $S_K(\Lambda) = S_t(\Lambda_i)$.

Step 6 [iteration end] If $i < K$, go to step 3, otherwise return.

Finally we introduce another method which requires no search. This method is based on the lookahead principle, which guarantees that the requests $r_1^1 r_2^1 \cdots r_{\ell_1-\ell_2+1}^1$ are always in the optimal solution if $\ell_1 \geq \ell_2$. Therefore we can first assign

$$r_1^1 r_2^1 \cdots r_{\ell_1-\ell_2+1}^1$$

in the solution set. Then we construct the new request sequence set

$$\Lambda^1 = \{\Gamma^{11}, \Gamma^{21}\}$$

where

$$\Gamma^{11} = r_{\ell_1-\ell_2+2}^1 \cdots r_{\ell_1}^1$$

and

$$\Gamma^{21} = \Gamma_{\ell_2}^2 \text{ if } \{\Gamma_{\ell_1-\ell_2+1}^1, \Gamma_{\ell_2}^2\} \text{ exists no conflict.}$$

$$= \Gamma_{\ell}^2 \text{ if } \{\Gamma_{\ell_1-\ell_2+1}^1, \Gamma_{\ell}^2\} \text{ exists no conflict but}$$

$$r^2_{\ell+1} \in \Gamma^1_{\ell_1-\ell_2+1}$$

A similar process can be applied again until the elements in the new request sequence set are both null.

Algorithm 3: The method to find optimal solution of $\Lambda = \{\Gamma^1, \Gamma^2\}$ without search. Assume that $\Lambda^0 = \Lambda$, $\Gamma^{10}=\Gamma^1$, $\Gamma^{20}=\Gamma^2$, $S_*(\Lambda) = \{\Gamma^1_*, \Gamma^2_*\}$. Also we adopt the convention $r_i r_{i+1} \cdots r_j = \emptyset$ if $i>j$.

Step 1 [initialize] Set $j=0$, $S_*(\Lambda) = \{\emptyset, \emptyset\}$.

Step 2 [termination test] Set $\ell=\ell g(\Gamma^{1j})$, $\ell\ell=\ell g(\Gamma^{2j})$.

For convenience, we let $\Lambda^j=\{\Gamma^{1j}, \Gamma^{2j}\}=\{ r^{1j}_1 \cdots r^{1j}_\ell, r^{2j}_1 \cdots r^{2j}_{\ell\ell}\}$.

If $\ell=0$ and/or $\ell\ell=0$, then $S_*(\Lambda)=\{\Gamma^1_* \cdot \Gamma^{1j}, \Gamma^2_* \cdot \Gamma^{2j}\}$ and stop.

Step 3 [iteration begin] If $\ell \geq \ell\ell > 0$, then we have

$S_*(\Lambda)=\{\Gamma^1_* \cdot \Gamma^{1j}_{\ell-\ell\ell+1}, \Gamma^2_*\}$

and $\Lambda^{j+1}=\{r^{1j}_{\ell-\ell\ell+2} \cdots r^{1j}_\ell, \Gamma^{2j}_k\}$

where
$$k = \begin{cases} 0 \text{ if } r^{2j}_1 \in \Gamma^{1j}_{\ell-\ell\ell+1} \\ \ell\ell \text{ if } r^{2j}_i \notin \Gamma^{1j}_{\ell-\ell\ell+1} \text{ for all } i,\ 1 \leq i \leq \ell\ell \\ L \text{ if } r^{2j}_i \notin \Gamma^{1j}_{\ell-\ell\ell+1} \text{ for all } i,\ 1 \leq i \leq L \leq \ell\ell \\ \quad \text{and } r^{2j}_{L+1} \in \Gamma^{1j}_{\ell-\ell\ell+1} \end{cases}$$

If $\ell\ell > \ell > 0$, then we have

$S_*(\Lambda)=\{\Gamma^1_*, \Gamma^2_* \cdot \Gamma^{2j}_{\ell\ell-\ell+1}\}$

and $\Lambda^{j+1}=\{\Gamma^{1j}_k, r^{2j}_{\ell\ell-\ell+2} \cdots r^{2j}_{\ell\ell}\}$

where
$$k = \begin{cases} 0 \text{ if } r_i^{1j} \in \Gamma^{2j}_{\ell\ell-\ell+1} \\ \ell \text{ if } r_1^{1j} \notin \Gamma^{2j}_{\ell\ell-\ell+1} \text{ for all } i,\ 1 \le i \le \ell \\ L \text{ if } r_i^{1j} \notin \Gamma^{2j}_{\ell\ell-\ell+1} \text{ for all } i,\ 1 \le i \le L \le \ell \\ \quad \text{and } r^{1j}_{L+1} \in \Gamma^{2j}_{\ell\ell-\ell+1} \end{cases}$$

Step 4 [iteration end] Set $j = j + 1$ and go to step 2.

The preceding algorithm can be illustrated by the following example:

Example: Given $\Lambda = \{35841, 2714\}$

Then at initialization ($j = 0$), we have

$$\Lambda^0 = \{35841, 2714\}$$

$$S_*(\Lambda) = \{35, \emptyset\}.$$

The iterative process gives

$j = 1$: $\Lambda^1 = \{841, 2714\}$

$$S_*(\Lambda) = \{35 \cdot \emptyset, \emptyset \cdot 27\}$$

$$= \{35, 27\}$$

$j = 2$: $\Lambda^2 = \{841, 14\}$

$$S_*(\Lambda) = \{35 \cdot 84, 27 \cdot \emptyset\}$$

$$= \{3584, 27\}$$

$j = 3$: $\Lambda^3 = \{1, 1\}$

$$S_*(\Lambda) = \{3584 \cdot 1, 27 \cdot \emptyset\}$$

$$= \{35841, 27\}$$

$j = 4$: $\Lambda^4 = \{\emptyset, \emptyset\}$ stop.

$S_*(\Lambda) = \{35841, 27\}$

stop.

Finally we want to emphasize that in this research we haven't done the implementation analysis and timing analysis to show which algorithm would be superior to the others. Also we don't know how close these algorithms are to optimal with respect to the execution time and ease of implementation.

5. AVERAGE BANDWIDTH IN TWO-PROCESSOR SYSTEM

We now derive the theoretic values of average bandwidth under different values of lookahead, say lookahead = 0, 1, optimal. These values will be verified by simulation experiments. Recall that m, p represent the number of memory modules and processors respectively.

Theorem 3: The probability of finding a request sequence with length ℓ is (see [8])

$$P_L(\ell) = \frac{P_{\ell-1}^{m-1} \cdot \ell}{m^{\ell}} \quad \text{for } 1 \le \ell \le m$$

$$= 0 \qquad \text{otherwise}$$

By Theorem 3, the average bandwidth in a single processor system is $\sum_{\ell=1}^{m} P_L(\ell) \cdot \ell$.

We next prove the theorem for lookahead = 0.

Theorem 4: The average bandwidth in a two-processor system with lookahead = 0 is

$$\sum_{\ell_1=1}^{m} \sum_{\ell_2=1}^{\ell_1} \left(C \; P_L(\ell_1) P_L(\ell_2) \sum_{\ell=\ell_1}^{L} (P_0(\ell) \cdot \ell) \right)$$

where $C = 1$ if $\ell_1 = \ell_2$ and 2 otherwise.

$L = \min(\ell_1 + \ell_2, m)$

$P_L(\cdot)$ is defined in Theorem 3.

$$P_0(\ell) = \left(\prod_{k=0}^{\ell-\ell_1-1} \frac{m-\ell_1-k}{m-k} \right) \left(\frac{\ell_1}{m-\ell+\ell_1} \right) \quad 1-\lfloor \ell/L \rfloor$$

Proof: There are m x m mutually exclusive events for the outcome of two request sequences, thus

$$E_{\ell_1\ell_2}: \ \ell g(\Gamma^1) = \ell_1 \text{ and } \ell g(\Gamma^2) = \ell_2$$

$$\text{for } 1 \le \ell_1 \le m \text{ and } 1 \le \ell_2 \le m$$

Considering the event $E_{\ell_1\ell_2}$, the application of Algorithm 2 will give us that the length of solution after lookahead = 0 will have the lower bound $\max(\ell_1,\ell_2)$ and the upper bound $\min(\ell_1+\ell_2,m)$. Without loss of generality, we assume $\ell_1 \ge \ell_2$ because we need only rename them if this assumption is not true.

The probability of different lengths ℓ can be found as follows:

For $\ell = \ell_1$, then $r_1^2 = r_1^1$ or $r_1^2 = r_2^1$ or ... or $r_1^2 = r_{\ell_1}^1$

$$\therefore \Pr(\ell=\ell_1) = \frac{\ell_1}{m}$$

For $\ell = \ell_1 + \delta$ where $\ell_1 < \ell < \min(\ell_1+\ell_2, m)$

then $r_1^2 \neq r_1^1$ and $r_1^2 \neq r_2^1$ and ... and $r_1^2 \neq r_{\ell_1}^1$

.

.

.

$r_\delta^2 \neq r_1^1$ and $r_\delta^2 \neq r_2^1$ and ... and $r_\delta^2 \neq \Gamma_{\ell_1}^1$

$$r^2_{\delta+1} = r^1_1 \text{ or } r^2_{\delta+1} = r^1_2 \text{ or } \ldots \text{ or } r^2_{\delta+1} = r^1_{\ell_1}$$

$$\therefore \Pr(\ell=\ell+\delta) = \frac{m-\ell_1}{m} \cdot \frac{m-\ell_1-1}{m-1} \cdots \frac{m-\ell_1-(\delta-1)}{m-(\delta-1)} \cdot \frac{\ell_1}{m-\delta}$$

For $\ell = \min(\ell_1+\ell_2, m)$, there are two cases:

Case 1: $\min(\ell_1+\ell_2, m) = \ell_1+\ell_2$

we have $r^2_1 \neq r^1_1$ and $r^2_1 \neq r^1_2$ and ...and $r^2_1 \neq r^1_{\ell_1}$

.

.

.

$r^2_{\ell_2} \neq r^1_1$ and $r^2_{\ell_2} \neq r^1_2$ and ... and $r^2_{\ell_2} \neq r^1_{\ell_1}$

$$\therefore \Pr(\ell=\ell_1+\ell_2) = \underbrace{\frac{m-\ell_1}{m} \cdot \frac{m-\ell_1-1}{m-1} \cdot \cdots \cdot \frac{m-\ell_1-(\ell_2-1)}{m-(\ell_2-1)}}_{\ell_2 \text{ factors}}$$

Case 2: $\min(\ell_1+\ell_2, m) = m$

we have $r^2_1 \neq r^1_1$ and $r^2_1 \neq r^1_2$ and ... and $r^2_1 \neq r^1_{\ell_1}$

.

.

.

$r^2_{m-\ell_1} \neq r^1_1$ and $r^2_{m-\ell_1} \neq r^1_2$ and ... and $r^2_{m-\ell_1} \neq r^1_{\ell_1}$

$$\therefore \Pr(\ell=m) = \underbrace{\frac{m-\ell_1}{m} \cdot \frac{m-\ell_1-1}{m-1} \cdot \ldots \cdot \frac{1}{\ell_1+1}}_{(m-\ell_1)\text{ factors}}$$

Combining Case 1 and Case 2, we have

$$\Pr(\ell=\min(\ell_1+\ell_2, m)) = \prod_{k=0}^{\min(\ell_1+\ell_2, m)-\ell_1-1} \frac{m-\ell_1-k}{m-k}$$

Therefore the expected length in Event $E_{\ell_1\ell_2}$ is

$$\sum_{\ell=\ell_1}^{L} P_0(\ell)\cdot\ell, \quad L = \min(\ell_1+\ell_2, m)$$

where
$$P_0(\ell) = \begin{cases} \Pr(\ell=\ell_1) \\ \Pr(\ell=\ell_1+\delta) \\ \Pr(\ell=L) \end{cases}$$

or in compact form

$$P_0(\ell) = \left(\sum_{k=0}^{\ell-\ell_1-1} \frac{m-\ell_1-k}{m-k} \right) \left(\frac{\ell_1}{m-\ell+\ell_1} \right)^{1-\lfloor \ell/L \rfloor}$$

Using the above results, we can compute the average bandwidth as

$$\sum_{\ell_1=1}^{m} \sum_{\ell_2=1}^{m} P_L(\ell_1) P_L(\ell_2) \left(\sum_{\ell=\max(\ell_1,\ell_2)}^{L} P_0(\ell)\cdot\ell \right)$$

Since the event $E_{\ell_1\ell_2}$ is equivalent to event $E_{\ell_2\ell_1}$, then the preceding formula is equivalent to:

$$\sum_{\ell_1=1}^{m} \sum_{\ell_2=1}^{\ell_1} \left(C \; P_L(\ell_1) P_L(\ell_2) \sum_{\ell=\ell_1}^{L} (P_0(\ell) \cdot \ell) \right)$$

Q.E.D.

The theorem for lookahead = 1 can be similarly derived. To avoid repetition, we state the theorem without including proof.

Theorem 5: The average bandwidth in two-processor system with lookahead = 1 is:

$$\sum_{\ell_1=1}^{m} \sum_{\ell_2=1}^{\ell_1} \left(C \; P_L(\ell_1) \cdot P_L(\ell_2) \sum_{\ell=\ell_1}^{L} (P_0(\ell)\ell + \Delta_1(\ell)) \right)$$

where $C = 1$ if $\ell_1 = \ell_2$ and 2 otherwise.

$L = \min(\ell_1 + \ell_2, m)$

$P_L(\cdot)$ is defined in Theorem 3.

$P_0(\cdot)$ is defined in Theorem 4.

$$\Delta_1(\ell) = \sum_{\delta=1}^{L_1} P_1(\delta) \cdot \delta$$

in which

$$L_1 = \min\{m-\ell, \ell_1 + \ell_2 - \ell - 1\}$$

$$P_1(\delta) = \left(\prod_{k_1=0}^{\ell-\ell_1-1} \frac{m-\ell_1-k_1}{m-k_1} \right) \left(\frac{1}{m-\ell+\ell_1} \right) \left(\prod_{k_2=1}^{\delta} \frac{m-\ell-k_2+1}{m-\ell+\ell_1-k_2} \right)$$

$$\left(\frac{\ell_1 - 1}{m-\ell+\ell_1-\delta-1} \right)^{1-\lfloor \delta/L_1 \rfloor}$$

It is clear that the average bandwidth for lookahead = 1 is better than that for lookahead = 0.

It would be desirable to have a theorem which gives an explicit expression for the average bandwidth of a two-processor system with optimal request sequence set. We have not been able to obtain this result. We do present an algorithm which yields an optimal solution. Let us examine an example for illustration.

Example: Find the average bandwidth of an optimal request sequence set with m = 4.

There are 16 mutually exclusive events possible for the outcome of two request sequences, thus

$E_{\ell_1\ell_2}$: $\ell_g(\Gamma^1)=\ell_1$ and $\ell g(\Gamma^2)=\ell_2$

$1\leq\ell_1\leq 4,\ 1\leq\ell_2\leq 4$

Consider E_{11}. Let $\Gamma^1 = 1$. Then we have the following two cases:

(1) $\Gamma^2 \epsilon \{1\}$,

(2) $\Gamma^2 \epsilon \{2,3,4\}$

The probability for case (1) to occur is equal to 1/4 and its corresponding bandwidth is 1. Also the probability for case (2) is equal to 3/4 and its corresponding bandwidth is 2.

If Γ^1 happens to be another value, thus $\Gamma^1 = 2$ or $\Gamma^1 = 3$ or $\Gamma^1 = 4$, we get the same results. Therefore for each event, we need only to

consider one case for Γ^1. This saves a great amount of effort.

The resulting average bandwidth for E_{11} is

$$B_{11} = \frac{1}{4} \times 1 + \frac{3}{4} \times 2$$

Consider E_{21}. Let $\Gamma^1 = 12$. We also have two cases:

(1) $\Gamma^2 \in \{1,2\}$

(2) $\Gamma^2 \in \{3,4\}$

The probability for case (1) is $\frac{2}{4}$, for case (2) is also $\frac{2}{4}$. The bandwidth for case (1) is 2, for case (2) is 3. The resulting average bandwidth for E_{21} is

$$B_{21} = \frac{2}{4} \times 2 + \frac{2}{4} \times 3$$

With the same arguments, we find that the resulting average bandwidth for E_{12} is equal to that for E_{21}, i.e.

$$B_{12} = B_{21}$$

Consider E_{22}. We let $\Gamma^1 = 12$. Now we have three cases:

(1) $\Gamma^2 \in \{12,13,14,21\}$

(2) $\Gamma^2 \in \{23,24,31,32,41,42\}$

(3) $\Gamma^2 \in \{34,43\}$

The probabilities for cases (1), (2), and (3) to occur are equal to $\frac{4}{12}$, $\frac{6}{12}$, and $\frac{2}{12}$; the corresponding bandwidth for each case is 2, 3, 4, respectively. Therefore the resulting average bandwidth for E_{22} is

$$B_{22} = \frac{4}{12} \times 2 + \frac{6}{12} \times 3 + \frac{4}{12} \times 4$$

Events E_{31}, E_{13}, E_{32}, E_{23} can be analyzed in the same manner. Next consider E_{33}. Let $\Gamma^1 = 123$. We have only two cases

(1) $\Gamma^2 \in \{123,124,132,134,142,143,213,214,231,241,312,$
$314,321\}$

(2) $\Gamma^2 \in \{234,243,324,341,342,412,413,421,423,431,432\}$

The probabilities and bandwidths for cases (1) and (2) are $\frac{13}{24}$, 3, $\frac{11}{24}$, 4, respectively. The resulting average bandwidth for E_{33} is

$$B_{33} = \frac{13}{24} \times 3 + \frac{11}{24} \times 4$$

Finally consider events $E_{41}, E_{14}, E_{42}, E_{24}, E_{34}, E_{44}$. Since the length of one request sequence is already equal to the number of memory modules m, then it is trival that

$$B_{41} = B_{14} = B_{42} = B_{24} = B_{43} = B_{34} = B_{44} = 4$$

Note that the probability for $E_{\ell_1 \ell_2}$ to occur is equal to

$$P_L(\ell_1) \cdot P_L(\ell_2)$$

where $P_L(\cdot)$ is defined in Theorem 1. Therefore the average bandwidth of the optimal system with m = 4 is computed by

$$B = \sum_{\ell_1=1}^{4} \sum_{\ell_2=1}^{\ell_1} (C \cdot P_L(\ell_1) \cdot P_L(\ell_2) \cdot B_{\ell_1 \ell_2})$$

where $C = 1$ if $\ell_1 = \ell_2$ and 2 otherwise.

In the above approach, we need some systematic way to generate all possible Γ^2 sequentially. This is a permutation problem, i.e. the permutations of m different objects taken ℓ_2 at a time without repetition. There are $P^m_{\ell_2}$ possibilities altogether. The natural approach to general all possible Γ^2 is according to lexicographic order. For example, m=4 and ℓ_2=3, then there are 24 possibilities which are generated in the following order:

123, 124, 132, 134, 142, 143, 213, 214,

231, 234, 241, 243, 312, 314, 321, 324,

341, 342, 412, 413, 421, 423, 431, 432

For convenience, we define the function "next" to give the next one. Thus

next (123) = 124

next (124) = 132

.

.

.

next (431) = 432

next (432) = undefined.

Now we are ready to introduce the following general algorithm.

Algorithm 4. Algorithm to compute average bandwidth of the two-processor system with optimal solution. Γ^1 stores the first request sequence with length ℓ_1; Γ^2 stores the second request sequence with

length ℓ_2. Note that $1 \le \ell_2 \le \ell_1 \le m$ where m is total number of memory modules. For event $E_{\ell_1 \ell_2}$, we let $\Gamma^1 = 12 \ldots \ell_1$, and Γ^2 could be any request sequence which is the permutation of ℓ_2 objects from set $\{1, 2, \ldots, m\}$. Thus there are $P^m_{\ell_2}$ cases in $E_{\ell_1 \ell_2}$. The length of the optimal solution for each case is stored in the array N[1:m].

Step 1 [initialize] Set $\ell_1 = 1$, $\ell_2 = 1$, $B = 0$

Step 2 [iteration begin] Set $\Gamma^1 = 12 \ldots \ell_1$, $\Gamma^2 = 12 \ldots \ell_2$, $t = 1$,

$$t_{max} = P^m_{\ell_2}, \quad N[1:m] = 0$$

Step 3 [inner loop begin] Apply Algorithm 3 to find the optimal solution of $\{\Gamma^1, \Gamma^2\}$. Set $\ell_0 = \ell g(S_*(\{\Gamma^1, \Gamma^2\}))$

Step 4 [inner loop end] Set $N[\ell_0] = N[\ell_0] + 1$, $t = t + 1$.

If $t \le t_{max}$, set $\Gamma^2 = \text{next}(\Gamma^2)$ and then go to step 3.

Step 5 [compute average bandwidth] Set $B = B + C \cdot P_L(\ell_1)$

$\cdot P_L(\ell_2) \left(\sum_{\ell=1}^{m} \frac{N[\ell]}{t_{max}} \cdot \ell \right)$ where $C = 1$ if $\ell_1 = \ell_2$ and 2 otherwise.

Step 6 Set $\ell_2 = \ell_2 + 1$. If $\ell_2 \le \ell_1$, go to step 2.

Step 7 [iteration end] Set $\ell_1 = \ell_1 + 1$, $\ell_2 = 1$. If $\ell_1 \le m$, go to step 2, otherwise stop.

For each m, we notice from the above algorithm that the total number of cases which must be considered to determine an optimal solution is

$$\sum_{\ell_1=1}^{m} \sum_{\ell_2=1}^{\ell_1} P_{\ell_2}^{m} .$$

This figure grows very quickly. For example, for m = 4, it gives 124, but for m = 16, it becomes 219192, which is more than four orders of magnitude larger. Therefore we enumerate some rules which are helpful in reducing the total number of cases which must be considered in the application of the algorithm.

Let $\Gamma^1 = r_1^1 r_2^1 \ldots r_{\ell_1}^1$, $\Gamma^2 = r_1^2 r_2^2 \ldots r_{\ell_2}^2$, $\ell_1 \geq \ell_2$

Rule 1 If $r_1^2 = r_1^1$, then the probability of $B_{\ell_1 \ell_2} = \ell_1$ is

$$Pr(B_{\ell_1 \ell_2} = \ell_1) = 1$$

Rule 2 If $r_1^2 = r_2^1$, then $Pr(B_{\ell_1 \ell_2} = \ell_1) = 1$ for $\ell_1 > \ell_2$

and $\begin{cases} Pr(B_{\ell_1 \ell_2} = \ell_1) = \dfrac{\ell_1 - 1}{m-1} \\ Pr(B_{\ell_1 \ell_2} = \ell_1 + 1) = \dfrac{m-\ell_1}{m-1} \end{cases}$ for $\ell_1 = \ell_2$

Rule 3 If $1 = \ell_2 \leq \ell_1 < m$, then

$$\begin{cases} Pr(B_{\ell_1 \ell_2} = \ell_1) = \dfrac{\ell_1}{m} \\ Pr(B_{\ell_1 \ell_2} = \ell_1 + 1) = \dfrac{m - \ell_1}{m} \end{cases}$$

Rule 4 If $2 = \ell_2 \leq \ell_1 \leq m$, then

$$\begin{cases} \Pr(B_{\ell_1\ell_2} = \ell_1) = \dfrac{\ell_1 - 1}{m-1} \\ \Pr(B_{\ell_1\ell_2} = \ell_1 + 1) = \dfrac{(m-\ell_1)(\ell_1+1)}{m(m-1)} \\ \Pr(B_{\ell_1\ell_2} = \ell_1 + 2) = \dfrac{(m-\ell_1)(m-\ell_1-1)}{m(m-1)} \end{cases}$$

Rule 5 If $\ell_1 = m$, then

$$\Pr(B_{\ell_1\ell_2} = m) = 1.$$

The proofs of these rules are straightforward, so they are omitted. Rules for $\ell_2 = 3$, 4, etc. or $r_1^2 = r_3^1$, r_4^1, etc., can be established but are more complicated and offer less gain.

Table 1 gives some representative computational results from Theorems 3, 4, 5 and Algorithm 4. Optimal solutions are not presented for m = 16, 32, 64 because they take too much computer time in order to compute them. Table 2 shows average bandwidths obtained by simulations.

We use the same random number generator with different seeds for each processor. At every memory cycle, the request sequences for two processors are determined and the corresponding optimal solution is computed. The average bandwidth is obtained by averaging the lengths of all optimal solutions over the entire experiment period. It is found that 1000 memory cycles produce stable results. To be comparable, same random number generator with the same seed for some processor is

used at different value of m, total number of memory modules. The results in Tables 1 and 2 agree very well.

It is clear from Table 2 that the average bandwidth of two-processor system is better than that of single processor system. Thus increasing the processor number will improve average bandwidth of multi-memory system. Also, as it should be, the result of lookahead = 1 is better than that of lookahead = 0; the result of optimal solution is best. It is significant however, that their differences are small. Therefore we may use lookahead = 0 in two-processor system to get the reward of simplified hardware for the selector in Figure 1.

Table 1 Theoretic Results of Average Bandwidth

		p = 2		
m	p = 1	lookahead = 0	lookahead = 1	Optimal
2	1.5	1.875	1.875	1.875
4	2.219	3.068	3.115	3.125
6	2.775	3.981	4.046	4.075
8	3.245	4.751	4.827	4.872
16	4.704	7.134	7.230	
32	6.774	10.510	10.619	
64	9.706	15.287	15.406	

Table 2 Simulated Results of Average Bandwidth

		p = 2		
m	p = 1	lookahead = 0	lookahead = 1	Optimal
2	1.505	1.872	1.872	1.872
4	2.225	3.105	3.154	3.162
6	2.785	4.007	4.076	4.106
8	3.280	4.767	4.868	4.901
16	4.701	7.184	7.270	7.370
32	6.736	10.473	10.596	10.812
64	9.450	15.137	15.274	15.547

6. THE TWO-PROCESSOR SOLUTIONS UNDER RESTRICTION

Throughout the above discussion, we implicitly assume that each processor can handle up to m words per memory cycle. However, it is not always true since this upperbound is decided not only by m, but also by n, the number of processor cycles per memory cycle. In other words, n is computed by

$$n = \frac{T_m}{T_p}$$

where T_m is memory cycle time

and T_p is processor cycle time.

For example, T_m in CDC 7600 is 270 ns while its T_p is 27 ns, then n = 270/27 = 10. But there can be as many as 32 memory modules in this computer system. Therefore the results in above sections are true only if $m \leq n$. For the case $m > n$, the feasible solution must be restricted in such a manner that the length of each element in solution set should not be greater than n. We call it the feasible solution under restriction and use notation $S^r(\Lambda)$ to represent it. For general multiprocessor system, the result is

$$S^r(\Lambda) = \{\Gamma^1_{i_1}, \Gamma^2_{i_2}, \ldots, \Gamma^P_{i_P}\}$$

where $\Gamma^k_{i_k}$ is the request subsequence of Γ^k and

$$\ell g(\Gamma^k_{i_k}) \leq n \text{ for } 1 \leq k \leq p$$

The optimal solution under restriction $S_*^r(\Lambda)$ is the feasible solution which has the greatest length under restriction.

Next we apply the same restriction to the definition of request sequence set. The request sequence set under restriction is

$$\Lambda^r = \{\Gamma^{1r}, \Gamma^{2r}, \ldots, \Gamma^{pr}\}$$

where
$$\Gamma^{kr} = \Gamma^k \text{ if } \ell g(\Gamma^k) \leq n$$
$$= \Gamma_n^k \text{ if } \ell g(\Gamma^k) > n$$

for $1 \leq k \leq p$.

If the optimal solution of the request sequence set under restriction $S_*(\Lambda^r)$ is as good as the optimal solution of Λ under restriction $S_*^r(\Lambda)$ then it shows the simplest way to handle restricted case. Fortunately this circumstance does exist.

Theorem 6 $\ell g(S_*(\Lambda^r)) = \ell g(S_*^r(\Lambda))$

Proof: Assume $\ell g(S_*(\Lambda^r)) \neq \ell g(S_*^r(\Lambda))$

then there are two cases to be considered.

Case 1: $\ell g(S_*(\Lambda^r)) > \ell g(S_*^r(\Lambda))$

This says that there is another feasible solution under restriction $S_*(\Lambda^r)$ whose length is greater than the length of $S_*^r(\Lambda)$, the feasible solution with greatest length by definition. This gives a contradiction.

Case 2: $\ell g(S_*(\Lambda^r)) < \ell g(S_*^r(\Lambda))$

By definition of $S_*^r(\Lambda)$, it must be a request subsequence set of Λ^r. Then $S_*^r(\Lambda)$ should be a feasible solution of Λ^r. But $S_*(\Lambda^r)$ is an

optimal solution of Λ^r, its length should not be less than the length of any feasible solution. So a contradiction exists, too.

By the results of Case 1 and Case 2, we conclude that $S_*(\Lambda^r)$ is as good as $S_*^r(\Lambda)$.

Q.E.D

Note that the above theorem holds for general multiprocessor system ($p \geq 2$). It enables us to find an optimal solution under restriction. We need only first to construct Λ^r from the request queues of all processors, then find its optimal solution. An approximate solution under restriction may be obtained in a similar way.

Now we want to compute the average bandwidths under restriction for two-processor system. It is necessary to modify the results in Section 5.

Theorem 4r. The average bandwidth under restriction in two-processor system with lookahead = 0 is

$$\sum_{\ell_1=1}^{n} \sum_{\ell_2=1}^{\ell_1} C \, P_{L_r}(\ell_1) P_{L_r}(\ell_2) \Big(\sum_{\ell=\ell_1}^{L} P_0(\ell) \cdot \ell \Big)$$

where $C = 1$ if $\ell_1 = \ell_2$ and 2 otherwise.

$$L = \min(\ell_1 + \ell_2, m)$$

$$P_{L_r}(\ell) = P_L(\ell) \text{ for } \ell < n$$

$$= \sum_{\ell'=n}^{m} P_L(\ell') \text{ for } \ell = n$$

$P_0(\cdot)$ is defined in Theorem 4.

Theorem 5r. The average bandwidth under restriction in two-processor system with lookahead = 1 is

$$\sum_{\ell_1=1}^{n} \sum_{\ell_2=1}^{\ell_1} \left(C \; P_{L_r}(\ell_1) P_{L_r}(\ell_2) \sum_{\ell=\ell_1}^{L} (P_0(\ell)\cdot\ell + \Delta_1(\ell)) \right)$$

where $C = 1$ if $\ell_1 = \ell_2$ and 2 otherwise.

$L = \min(\ell_1 + \ell_2, m)$

$P_{L_r}(\cdot)$ is defined in Theorem 4r.

$P_0(\cdot)$ is defined in Theorem 4.

$\Delta_1(\cdot)$ is defined in Theorem 5.

Algorithm 4r. Algorithm to compute average bandwidth under restriction of two-processor system with optimal solution. The notations used are the same as that in Algorithm 4.

Step 1 [initialize] Set $\ell_1 = 1$, $\ell_2 = 1$, $B = 0$. Read n.

Step 2 [iteration begin] Set $\Gamma^1 = 12\ldots\ell_1$, $\Gamma^2 = 12\ldots\ell_2$, $t = 1$

$t_{max} = P^m_{\ell_2}$, $N[1:m] = 0$

Step 3 [inner loop begin] Construct Λ^r. Apply Algorithm 3 to find the optimal solution of Λ^r. Set $\ell_0 = \ell g(S_*(\Lambda^r))$.

Step 4 [inner loop end] Set $N[\ell_0] = N[\ell_0] + 1$, $t = t + 1$. If

$t \leq t_{max}$, set Γ^2 = next (Γ^2) and then go to step 3.

Step 5 [compute average bandwidth] Set $B = B + C \cdot P_L(\ell_1) \cdot P_L(\ell_2) \cdot \left(\sum_{\ell=1}^{m} \frac{N[\ell]}{t_{max}} \cdot \ell \right)$ where $C = 1$ if $\ell_1 = \ell_2$ and 2 otherwise.

Step 6 Set $\ell_2 = \ell_2 + 1$. If $\ell_2 \leq \ell_1$, go to step 2.

Step 7 [iteration end] Set $\ell_1 = \ell_1 + 1$, $\ell_2 = 1$. If $\ell_1 \leq m$, go to step 2, otherwise stop.

7. SOLUTIONS IN MULTIPROCESSOR SYSTEM

In multiprocessor systems with $p > 2$, there are more than two request sequences in request sequence set. Thus

$$\Lambda = \{\Gamma^1, \Gamma^2, \ldots, \Gamma^p\}$$

where Γ^k, $1 \leq k \leq p$, is the request sequence of processor k. We first try to obtain algorithms which can give its optimal solution $S_*(\Lambda)$.

Let us consider exhaustive search. Thus we first construct all request subsequence sets

$$\Lambda_{i_1 \cdots i_p} = \{\Gamma^1_{i_1}, \ldots \Gamma^p_{i_p}\},\ 0 \leq i_1 \leq \ell_1, \ldots,\ 0 \leq i_p \leq \ell_p$$

Then the optimal solution is obtained by selecting the request subsequence set which has greatest length and in which exists no conflict. If there is a tie, any request sequence set arbitrarily chosen will be all right.

There are a total of $\prod_{k=1}^{p} (\ell_k+1)$ request subsequence sets needed to be considered. We attempt now to generalize the algorithms determined for the two-processor system to obtain an optimal solution for three-processor system, and then from three to four, and so on.

First of all, let us consider the case p=3. Without loss of generality, assume that $\ell_3=\min\{\ell_1, \ell_2, \ell_3\}$. Using Γ^3 as reference, it can be decomposed into ℓ_3+1 mutually exclusive events:

E_0: No request in Γ^3 is included in the solution

E_1: r^3_1 is included in the solution.

.

.

.

E_{ℓ_3}: $r^3_1 \ldots r^3_{\ell_3}$ is included in the solution.

For any event E_{i_3}, $0 \leq i_3 \leq \ell_3$, we can construct a new request sequence set

$$\Lambda' = \{\Gamma^1_{i_1}, \Gamma^2_{i_2}\},\ 1 \leq i_1 \leq \ell_1,\ 1 \leq i_2 \leq \ell_2$$

in such a way that

(1) $\Gamma^1_{i_1}$ contains no request in $\Gamma^3_{i_3}$, and $r^1_{i_1+1} \in \Gamma^3_{i_3}$ if $i_1 < \ell_1$.

(2) $\Gamma^2_{i_2}$ contains no request in $\Gamma^3_{i_3}$, and $r^2_{i_2+1} \in \Gamma^3_{i_3}$ if $i_2 < \ell_2$.

Now Algorithm 3 may be applied to compute the optimal solution of Λ', which in turn can give the solution for this event. Then the optimal solution of Λ will be that solution for one of the events which gives the greatest length. Although the optimal solution can still be obtained if Γ^1 or Γ^2 is chosen as reference, the search space would increase which is not desirable.

The preceding approach can be summarized into the following algorithm.

Algorithm 5. Find the optimal solution of $\Lambda = \{\Gamma^1, \Gamma^2, \Gamma^3\}$

where $\Gamma^1 = r_1^1 \ldots r_{\ell_1}^1$, $\Gamma^2 = r_1^2 \ldots r_{\ell_2}^2$, and $\Gamma^3 = r_1^3 \ldots r_{\ell_3}^3$.

Step 1 [initialize] Check to see whether $\ell_3 = \min\{\ell_1, \ell_2, \ell_3\}$.

If not, just rename them. Set $i_3 = 0$.

Step 2 [iteration begin] Construct Λ' as defined above and find $S_*(\Lambda')$ using Algorithm 3. Let $S_*(\Lambda') = \{\Gamma_{i_1}^1, \Gamma_{i_2}^2\}$.

Step 3 [solution for E_{i_3}] Set $S_{i_3}(\Lambda) = \{\Gamma_{i_1}^1, \Gamma_{i_2}^2, \Gamma_{i_3}^3\}$.

Step 4 [iteration end] Set $i_3 = i_3 + 1$. If $i_3 \leq \ell_3$, go to step 2.

Step 5 [answer] Set $S_*(\Lambda)$ = one with greatest length among $S_{i_3}(\Lambda)$, $0 \leq i_3 \leq \ell_3$.

Example: Given $\Lambda = \{123, 27, 18\}$

There are three mutual exclusive events.

E_0: $\Gamma_0^3 = \emptyset$

$\Lambda' = \{123, 27\}$

$S_*(\Lambda') = \{123, \emptyset\}$

$S_0(\Lambda) = \{123, \emptyset, \emptyset\}$

E_1: $\Gamma_1^3 = 1$

$\Lambda' = \{\emptyset, 27\}$

$S_*(\Lambda') = \{\emptyset, 27\}$

$S_1(\Lambda) = \{\emptyset, 27, 1\}$

E_2: $\Gamma_2^3 = 18$

$\Lambda' = \{\emptyset, 27\}$

$S_*(\Lambda') = \{\emptyset, 27\}$

$S_2(\Lambda) = \{\emptyset, 27, 18\}$

Therefore the optimal solution of Λ is $S_2(\Lambda)$, i.e.

$S_*(\Lambda) = \{\emptyset, 27, 18\}$

By applying the preceding algorithm, the search space has been greatly reduced. For given m, the average number of events to be considered in the above algorithm is

$$\sum_{\ell_1=1}^{m} \sum_{\ell_2=1}^{m} \sum_{\ell_3=1}^{m} P_L(\ell_1)P_L(\ell_2)P_L(\ell_3)(\min(\ell_1,\ell_2,\ell_3)+1)$$

while that in exhaustive search is

$$\sum_{\ell_1=1}^{m} \sum_{\ell_2=1}^{m} \sum_{\ell_3=1}^{m} P_L(\ell_1)P_L(\ell_2)P_L(\ell_3)(\ell_1+1)(\ell_2+1)(\ell_3+1)$$

For comparison, the number of events considered are shown in Table 3 for several values of m.

Table 3 Comparison of Algorithm 5 with Exhaustive Search for p=3

m	2	4	6	8	16	42	64
Algorithm 5	2.13	2.48	2.77	3.03	3.85	5.03	6.71
Exhaustive search	15.63	33.35	53.78	76.50	185.61	469.83	1227.03

Next we want to apply Algorithm 5 to find the optimal solution for p=4. In the same manner, we assume $\ell_4 = \min\{\ell_1, \ell_2, \ell_3, \ell_4\}$. Then Γ^4 is used as reference and we get ℓ_4+1 mutual exhaustive events:

E_0: No request in Γ^4 is included in the solution.

E_1: r_1^4 is included in the solution.

.

.

.

E_{ℓ_4}: $r_1^4 \ldots r_{\ell_4}^4$ is included in the solution.

For any event E_{i_4}, $0 \le i_4 \le \ell_4$, we can construct a new request sequence set

$$\Lambda' = \{\Gamma_{i_1}^1, \Gamma_{i_2}^2, \Gamma_{i_3}^3\},\ 1 \le i_1 \le \ell_1,\ 1 \le i_2 \le \ell_2,\ 1 \le i_3 \le \ell_3$$

in such a way that

(1) $\Gamma_{i_1}^1$ contains no request in $\Gamma_{i_4}^4$, and $r_{i_1+1}^1 \in \Gamma_{i_4}^4$ if $i_1 < \ell_1$.

(2) $\Gamma^2_{i_2}$ contains no request in $\Gamma^4_{i_4}$, and $r^2_{i_2+1} \in \Gamma^4_{i_4}$ if $i_1 < \ell_2$.

(3) $\Gamma^3_{i_3}$ contains no request in $\Gamma^4_{i_4}$, and $r^3_{i_3+1} \in \Gamma^4_{i_4}$ if $i_3 < \ell_3$.

Now Algorithm 5 can be applied to compute the optimal solution of Λ', which in turn can give the solution for this event. Then the optimal solution of Λ will be the solution for that one of the events which gives the greatest length. The preceding approach can be summarized into an algorithm which is quite similar to Algorithm 5.

Although the same approach can be used to obtain the optimal solution for p=5,6,..., the search effort would soon grow out of control. We have not been successful at obtaining optimal solutions for larger values of p. We can however get an algorithm to compute approximate solutions. It is quite practical and easy to implement.

The concept is to use a hierarchical method. All request sequences are first grouped two by two, say $\{\Gamma^1,\Gamma^2\}$, $\{\Gamma^3,\Gamma^4\}$,... The last set may contain only one element if p is odd. Then the optimal (or approximate) solution of each set can be found by some algorithm in Section 3. Call the results the level 1 request sequences and represent them by $\Gamma^{1,1}$, $\Gamma^{1,2}$,... Now the same process can be applied again and again until there is only one request sequence remaining in the next high level. This gives the approximate solution of $\Lambda = \{\Gamma^1,\Gamma^2,\ldots,\Gamma^p\}$. For given p, the solution will be found exactly at level $\lceil \log_2 p \rceil$. To be consistent, the original request sequences are also denoted by $\Gamma^{0,1},\Gamma^{0,2},\ldots,\Gamma^{0,p}$,

that is, they are at level 0.

The whole process is best viewed as a tree. For example, the solution tree for p=6 is shown in Figure 4. Note that $\Gamma^{3,1}$ is the obtained solution.

The following algorithm summarizes the preceding approach.

Algorithm 6. Find an approximation solution of $\Lambda = \{\Gamma^1, \Gamma^2, \ldots, \Gamma^p\}$ by the hierarchical method. ℓ denotes current level, ℓ_{max} is the level where the solution is located, N is the number of request sequences in current level.

Step 1 [initialize] Set $\ell = 0$, $\ell_{max} = \lceil \log_2 p \rceil$, $N=p$.

Step 2 [set up request sequences for next level] For $k=1,2,\ldots,\lfloor N/2 \rfloor$, set $\Gamma^{\ell+1,k}$ = the optimal (or approximate) solution of $\Gamma^{\ell,2k-1}$ and $\Gamma^{\ell,2k}$ by applying some algorithm in Section 4. If N is odd, set $\Gamma^{\ell+1,\lceil N/2 \rceil} = \Gamma^{\ell,N}$.

Step 3 [termination test] Set $\ell = \ell + 1$. If $\ell < \ell_{max}$, set $N = \lceil N/2 \rceil$ and go to step 2.

Step 4 [answer] The solution is $\Gamma^{\ell,1}$.

Example: Given $\Lambda = \{12345, 3784, 67, 7824, 452\}$

We use Algorithm 3 in step 2 of the above algorithm.

At level 1: $\Gamma^{1,1} = S_*(\{\Gamma^{0,1}, \Gamma^{0,2}\})$

$= S_*(\{12345, 3784\}) = 123784$

$\Gamma^{1,2} = S_*(\{\Gamma^{0,3}, \Gamma^{0,4}\})$

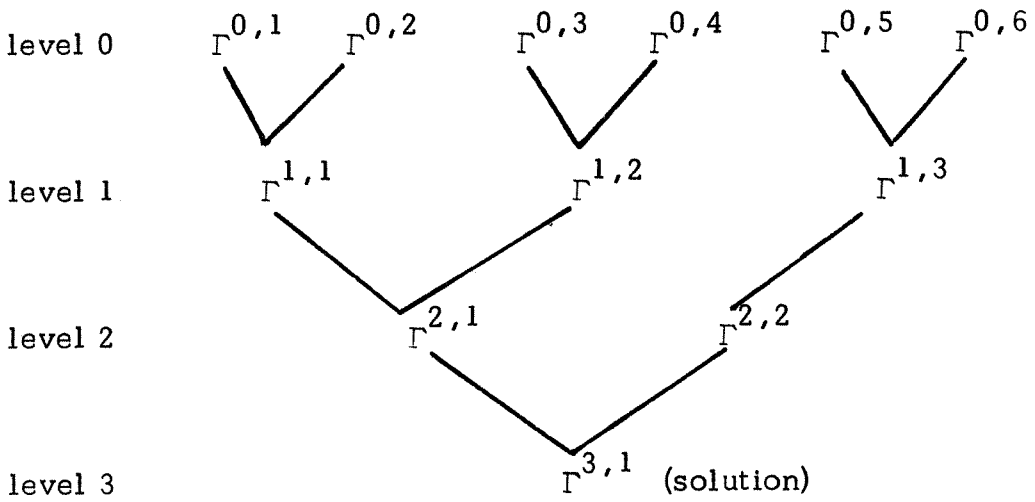


Figure 4. Solution Tree for p = 6

$= S_*(\{67, 7824\}) = 78246$

$\Gamma^{1,3} = \Gamma^{0,5} = 452$

At level 2: $\Gamma^{2,1} = S_*(\{\Gamma^{1,1}, \Gamma^{1,2}\})$

$= S_*(\{123784, 78246\}) = 123784$

$\Gamma^{2,2} = \Gamma^{1,3} = 452$

At level 3: $\Gamma^{3,1} = 1237845$ (solution)

8. AVERAGE BANDWIDTH IN MULTIPROCESSOR SYSTEM

In order to evaluate how much performance improvement can be obtained for multi-processor multi-memory systems, it is necessary to derive some algorithms to compute average bandwidth. We first consider optimal case for $p = 3$.

The algorithm to be used is similar to Algorithm 4. Now that there are three request sequences, the total number of events we need to enumerate is m^3, thus from E_{111} to E_{mmm}. This number can be reduced by noticing that E_{ijk}, E_{ikj}, E_{jik}, E_{jki}, E_{kij}, E_{kji} are six events which give the same average bandwidth and E_{iij}, E_{iji}, E_{jii} are three events which give the same average bandwidth for $i \neq j \neq k$. Therefore the average bandwidth of the optimal system for given m is computed by

$$B = \sum_{\ell_1=1}^{m} \sum_{\ell_2=1}^{\ell_1} \sum_{\ell_3=1}^{\ell_2} (C \cdot P_L(\ell_1) P_L(\ell_2) P_L(\ell_3) B_{\ell_1 \ell_2 \ell_3})$$

where $C = 6$ if $\ell_1 \neq \ell_2 \neq \ell_3$

$= 3$ if $\ell_1 \neq \ell_2 = \ell_3$ or $\ell_1 = \ell_2 \neq \ell_3$

$= 1$ if $\ell_1 = \ell_2 = \ell_3$

The modified Algorithm 4 is given as follows.

Algorithm 4': Algorithm to compute average bandwidth for $p = 3$ with optimal solution. Γ^1 stores the first request sequence with length

ℓ_1; Γ^2 stores the second request sequence with length ℓ_2; Γ^3 stores the third request sequence with length ℓ_3. Note that $m \geq \ell_1 \geq \ell_2 \geq \ell_3 \geq 1$ and m is total number of memory modules. For event $E_{\ell_1\ell_2\ell_3}$, we let $\Gamma^1 = 12\ldots\ell_1$, Γ^2 could be any request sequence which is the permutation of ℓ_2 objects from set $\{1,2,\ldots,m\}$, and Γ^3 could be any request sequence which is the permutation of ℓ_3 objects from set $\{1,2,\ldots,m\}$. Thus there are $P^m_{\ell_2}\ P^m_{\ell_3}$ cases in $E_{\ell_1\ell_2\ell_3}$ stored in array N[1:m].

Step 1 [initialize] Set $\ell_1 = 1$, $\ell_2 = 1$, $\ell_3 = 1$, B = 0.

Step 2 [iteration begin] Set $\Gamma^1 = 12\ldots\ell_1$, $\Gamma^2 = 12\ldots\ell_2$, $\Gamma^3 = 12\ldots\ell_3$, t = 1, t' = 1, $t_{max} = P^m_{\ell_2}\ P^m_{\ell_3}$, $t'_{max} = P^m_{\ell_3}$, N[1:m] = 0

Step 3 [inner loop begin] Apply Algorithm 5 to find the optimal solution of $\{\Gamma^1,\Gamma^2,\Gamma^3\}$. Set $\ell_0 = \ell g(S_*(\{\Gamma^1,\Gamma^2,\Gamma^3\}))$, $N[\ell_0] = N[\ell_0] + 1$, t = t + 1, t' = t' + 1.

Step 4 [change Γ^3 fast] If $t' \leq t'_{max}$, then set $\Gamma^3 = next(\Gamma^3)$ and go to step 3.

Step 5 [inner loop end] If $t \leq t_{max}$, then set $\Gamma^2 = next(\Gamma^2)$, $\Gamma^3 = 12\ldots\ell_3$, t' = 1, and go to step 3.

Step 6 [compute average bandwidth] Set $B = B + C \cdot P_L(\ell_1)$.

$$P_L(\ell_2)P_L(\ell_3)\left(\sum_{\ell=1}^{m}\frac{N[\ell]}{t_{max}}\cdot\ell\right)$$ where C=6 if $\ell_1\neq\ell_2\neq\ell_3$,

= 3 if $\ell_1\neq\ell_2=\ell_3$ or $\ell_1=\ell_2\neq\ell_3$, = 1 if $\ell_1=\ell_2=\ell_3$.

Step 7 Set $\ell_3 = \ell_3 + 1$. If $\ell_3 \leq \ell_2$, go to step 2.

Step 8 Set $\ell_2 = \ell_2 + 1$, $\ell_3 = 1$. If $\ell_2 \leq \ell_1$, go to step 2.

Step 9 [iteration end] Set $\ell_1 = \ell_1 + 1$, $\ell_2 = 1$, $\ell_3 = 1$.

If $\ell_1 \leq m$, go to step 2, otherwise stop.

Since there are a total of

$$\sum_{\ell_1=1}^{m}\sum_{\ell_2=1}^{\ell_1}\sum_{\ell_3=1}^{\ell_2} P^m_{\ell_2}\,P^m_{\ell_3}$$

cases which must be examined in order to find an optimal solution by the preceding algorithm, the computational effort grows more rapidly than for the two-processor system. We can only introduce some rules, as previously, to cut the number of events enumerations. For example.

Let $\Gamma^1 = r^1_1 r^1_2 \cdots r^1_{\ell_1}$, $\Gamma^2 = r^2_1 r^2_2 \cdots r^2_{\ell_2}$, $\Gamma^3 = r^3_1 r^3_2 \cdots r^3_{\ell_3}$,

and $\ell_1 \geq \ell_2 \geq \ell_3$

Rule 1. If $m \geq \ell_1 \geq \ell_2 = \ell_3 = 1$, then

$$Pr(B_{\ell_1\ell_2\ell_3} = \ell_1) = \left(\frac{\ell_1}{m}\right)^2$$

$$Pr(B_{\ell_1\ell_2\ell_3} = \ell_1 + 1) = \frac{(m-\ell_1)(2\ell_1+1)}{m^2}$$

$$\Pr(B_{\ell_1\ell_2\ell_3} = \ell_1 + 2) = \frac{(m-\ell_1)(m-\ell_1-1)}{m^2}$$

Rule 2. If $\ell_1 = m$, then $\Pr(B_{\ell_1\ell_2\ell_3} = m) = 1$

In the same manner, we can further modify Algorithm 4 to compute the average bandwidth for p = 4. It is necessary to add one more level of iteration for ℓ_4 and change the formula in Step 6 to be

$$B = B + C \cdot P_L(\ell_1)P_L(\ell_2)P_L(\ell_3)P_L(\ell_4)\left(\sum_{\ell=1}^{m} \frac{N[\ell]}{t_{max}} \cdot \ell\right)$$

where $C = 24$ if $\ell_1 \neq \ell_2 \neq \ell_3 \neq \ell_4$

$= 12$ if $\ell_1 = \ell_2 \neq \ell_3 \neq \ell_4$ or $\ell_1 \neq \ell_2 = \ell_3 \neq \ell_4$ or $\ell_1 \neq \ell_2 \neq \ell_3 = \ell_4$

$= 6$ if $\ell_1 = \ell_2 \neq \ell_3 = \ell_4$

$= 4$ if $\ell_1 \neq \ell_2 = \ell_3 = \ell_4$ or $\ell_1 = \ell_2 = \ell_3 \neq \ell_4$

$= 1$ if $\ell_1 = \ell_2 = \ell_3 = \ell_4$

The process is quite similar and therefore not presented. Note that the whole procedure can be extended to p = 5, 6, etc.

Table 4 summarizes typical computational results for small p and m. Two interesting points are worthy of mention. First, for given p, the larger the number of memory modules, the larger the average bandwidth. This coincides with our intuition since theoretically the average bandwidth

should be maximal when m is equal to the total number of words in a system, although the hardware cost would then be catastrophic also.

Table 4 Average Bandwidth for System with Optimal Solution

p \ m	2	4	6
1	1.5	2.219	2.775
2	1.875	3.125	4.075
3	1.969	3.576	4.828
4	1.992	3.802	

Second, for given m, the larger the number of processors, the larger the average bandwidth. In addition, it becomes saturated when p is large. This also coincides with our intuition since the limit value for average bandwidth is m.

For large values of p and m it takes too much computation time to determine the theoretic average bandwidth. We turn now to another important topic--finding the average bandwidth for Algorithm 6.

As shown before, there are $\lceil \log_2 p \rceil$ levels in the solution tree of Algorithm 6. At level 0, the length distribution function of request sequences is given in Theorem 3, i.e.,

$$P_L(\ell) = \frac{P_{\ell-1}^{m-1} \cdot \ell}{m^{\ell}}$$

Let us first consider the system using the lookahead = 0 algorithm to find all approximate solutions at all levels. At level 1, for a given node, its ancestors in level 0 may be one or two. When it is one, the probability to find a request sequence with length ℓ, $P_{L,1}(\ell)$, is equal to $P_L(\ell)$. When it is two, that probability, $P_{L,2}(\ell)$, can be found by the following algorithm.

Algorithm 7. Algorithm to find $P_{L,2}(\ell)$, $1 \le \ell \le m$, which uses lookahead = 0 algorithm. $P_L(\cdot)$ is defined in Theorem 3; $P_0(\cdot)$ is defined in Theorem 4.

Step 1 [initialize] Set $\ell_1 = 1$, $\ell_2 = 1$, $P_{L,2}[1:m] = 0$

Step 2 [accumulate] For $\ell = \ell_1$ to $\min(\ell_1 + \ell_2, m)$ do

$$P_{L,2}[\ell] = P_{L,2}[\ell] + C \cdot P_L(\ell_1) P_L(\ell_2) P_0(\ell)$$

where $C = 1$ if $\ell_1 = \ell_2$, and 2 otherwise.

Step 3 [inner loop end] Set $\ell_2 = \ell_2 + 1$. If $\ell_2 \le \ell_1$, go to step 2.

Step 4 [outer loop end] Set $\ell_1 = \ell_1 + 1$, $\ell_2 = 1$. If $\ell_1 \le m$, go to step 2, otherwise stop.

Then consider a given node in level 2. All its ancestors in level 0 may be three or four. The same algorithm can be used to give $P_{L,3}(\ell)$ and $P_{L,4}(\ell)$ except the formula in step 2 should be changed to

$$P_{L,3}[\ell] = P_{L,3}[\ell] + C \cdot P_{L,2}[\ell_1] P_{L,1}[\ell_2] P_0(\ell)$$

and

$$P_{L,4}[\ell] = P_{L,4}[\ell] + C \cdot P_{L,2}[\ell_1] P_{L,2}[\ell_2] P_0(\ell)$$

Generally there are 2^{N-1} possible cases for given node in level N. The corresponding $P_{L,i}[\ell]$, $2^{N-1} + 1 \leq i \leq 2^N$ can be found in Algorithm 7 except the formula in step 2 is changed to be

$$P_{L,i}[\ell] = P_{L,i}[\ell] + C \cdot P_{L,2^{N-1}}[\ell_1] \cdot P_{L,i-2^{N-1}}[\ell_2] P_0(\ell)$$

Applying the above results, the average bandwidth for given P is

$$\sum_{\ell=1}^{m} P_{L,p}[\ell] \cdot \ell$$

Table 5 summarizes typical computational results for small p and m. The agreement with the exact results in Table 4 is excellent.

Table 5 Average Bandwidth for System with Lookahead = 0

p \ m	2	4	6	8	16	32	64
1	1.5	2.219	2.775	3.245	4.704	6.774	9.706
2	1.875	3.068	3.981	4.751	7.134	10.510	15.287
3	1.969	3.435	4.565	5.521	8.489	12.700	18.666
4	1.992	3.611	4.858	5.914	9.197	13.861	20.472

Next, consider the system using Algorithm 3. The procedure to find average bandwidth for p=3 is complete the same as Algorithm 4' except the first part in Step 3 should be changed into "Apply Algorithm 6 to find the approximate solution of $\{\Gamma^1, \Gamma^2, \Gamma^3\}$, and set its length to be ℓ_0". A similar approach can be applied to p=4, 5, etc.

9. PHYSICAL IMPLEMENTATION AND DESIGN CRITERION

Algorithms which claim practicality should consider the implementation problem. Let us focus on Algorithm 6.

The basic problem is to design the selector, which is shown in Figure 2. A modular approach is adopted because of its many advantages. Let the selector module, or SM for short, be a functional unit which can find the optimal (or approximate) solution of the request sequence set with two elements according to some one of the algorithms given in Section 4. Then completing the detailed implementation of the selector part of Figure 2 gives Figure 5. Note that SCN stands for Scanner which is used to determine the request sequence $\Gamma^{0,j}$ from the request queue Q^j. $BF_{i,j}$ stands for buffer which stores the request $\Gamma^{i,j}$.

Since the final solution $\Gamma^{\ell_{max},1}$ consists of one request subsequence from each original request sequence $\Gamma^{0,j}$, $1 \leq j \leq p$, and each original request sequence is determined from its corresponding request queue, it is sufficient to attach to each request queue a pointer to this answer. This means that no buffer is necessary. The solutions obtained at each level are indicated by the contents of related pointers. The process of finding solutions results in the change of pointer contents. This approach saves not only the hardware cost of buffers but also the time to load each request sequence $\Gamma^{i,j}$ into $BF_{i,j}$.

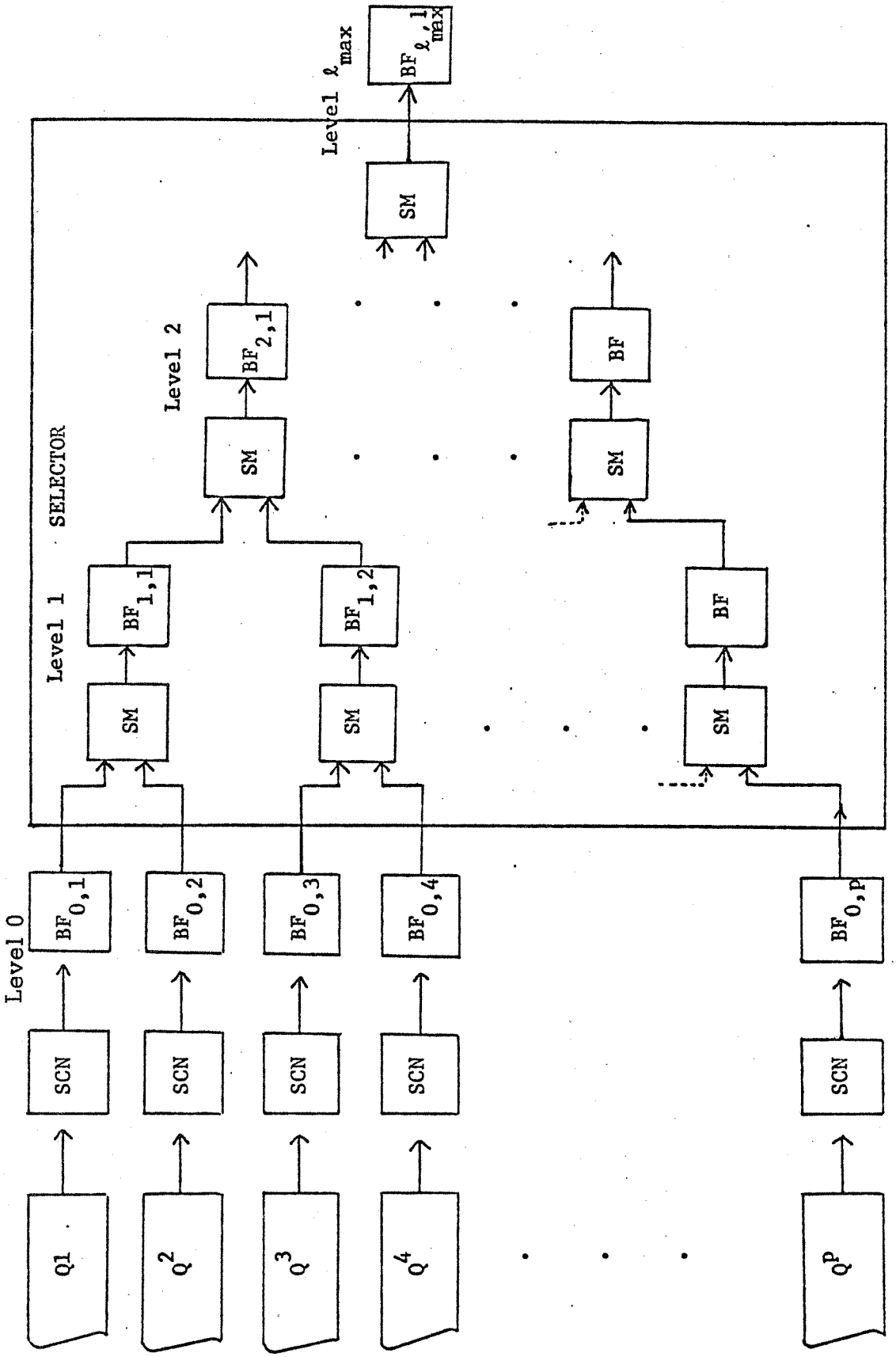


Figure 5. Logical implications of algorithm 6.

From the results of Section 8, the average bandwidth will increase when the number of processors increases. The number of levels in the selectors will also increase as the number of processors increases. The more the total levels, the more the delay from the input request queues to the output solution. Therefore there should exist some limit beyond which no gain could be obtained by increasing the number of processors. It is desirable to derive such upper bounds to serve as a design criterion.

Let the delay in scanner be T_k and the delay in selector module be T_s. The total delay in this system is

$$T_k + NT_s$$

where N is the number of levels which is equal to $\lceil \log_2 p \rceil$. The memory cycles required to supply this delay is equal to

$$\frac{T_k + NT_s}{T_m}$$

where T_m is the memory cycle time.

Since delay should not exceed one memory cycle, then the design criterion would be

$$\frac{T_k + NT_s}{T_m} \leq 1$$

or

$$N \leq \frac{T_m - T_k}{T_s}$$

or

$$P \geq 2^{(\frac{T_m - T_k}{T_s})}$$

For example, the memory cycle time of a large computer system might be 600 ns. If the delays in scanner and selector modules could be designed not to exceed 100 ns, then total number of processors could be as great as 2^5 or 32. (Remark: According to the result from our primary stage of logic design, the delay in selector modules could be made as small as 67.5 ns for m=16.)

10. CONCLUSION

The above discussions demonstrate a practical means to improve the memory system performance by properly scheduling memory requests in multi-processor multi-memory system. The entire approach concentrates in the optimization problem after memory requests have been queued. Therefore these schemes can be applied to any kind of configuration in multi-memory system, including multiple independent memory system and interleaved memory system.

It is well known that memory requests are not independent. It should be the case, however, that an interleaved memory with a sizeable number of modules would alow reasonably random patterns to each module. Even if the independence criteria fails severely, the techniques presented herein determine locally optimal request sequence sets for each memory cycle. Substantial performance improvement should be expected even if global optimality is not attained.

There are still many interesting questions to be explored. Examples are the effect of patterned reference requests on actual performance and the algorithm for optimal request sequences for large m and p.

BIBLIOGRAPHY

[1] C. J. Conti, "Concepts for buffer storage", IEEE Computer Group News, vol. 2, no. 8, pp. 9-13, March 1969.

[2] R. M. Meade, "On memory system design", in 1970 Fall Joint Computer Conf., AFIPS Conf. Proc., vol. 37, pp. 33-43.

[3] C. V. Ramamoorthy and K. M. Chandy, "Optimization of memory hierarchies in multiprogrammed systems," J. ACM, vol. 17, no. 3, pp. 426-445, July 1970.

[4] P. J. Denning, "The working set model for program behavior," Comm. ACM, vol. 11, no. 5, pp. 323-333, May 1968.

[5] J. M. Kurtzberg, "On the memory conflict problem in multiprocessor systems", IEEE Trans. Comp., vol. C-23, no. 3, pp. 286-293, March 1974.

[6] G. J. Burnett and E. G. Coffman, Jr., "A study of interleaved memory systems", in 1970 Spring Joint Comp. Conf., AFIPS Conf. Proc., vol. 36, pp. 467-474.

[7] G. J. Burnett and E. G. Coffman, Jr., "Analysis of interleaved memory systems using blockage buffers", Comm. ACM, vol. 18, no. 2, pp. 91-95, Feb. 1975.

[8] H. Hellerman, Digital Computer System Principles, New York: McGraw-Hill, 1973, 2nd Ed., pp. 244-245.