The Degradation in Memory Utilization Due to Dependencies

C. V. RAMAMOORTHY AND BENJAMIN W. WAH

Abstract—**In this correspondence we have presented a technique to find the degradation in memory and buffer utilization due to dependencies of accesses issued by a pipelined computer. When a dependency occurs, the request stream to the memory is interrupted and the utilization of the memory decreases. The dependency is then resolved in the pipelined computer. New requests are generated and the utilization gradually builds up to a maximum. The delay between the time a dependency occurs and the time the dependency is resolved depends highly on the method used in the pipelined computer to resolve the dependency. Hence, the evaluations of a specific computer is too restrictive in scope. A general organization of a pipelined computer is proposed. We evaluate the degradation in memory utilization due to dependencies using a Markovian model. This is validated against a detailed simulation model. The analysis is based on the memory configuration (Organization I) and simulation results with no dependencies that are presented in a separate paper in this issue.**

Index Terms—**Access dependencies, buffer utilization, interleaved memories, Markovian model, memory utilization, pipelined computer.**

I. Introduction

A *dependency* is a logical relationship between two addresses such that the second cannot be accessed (written or read) until the first has been accessed [3]. In a pipelined computer, the lookahead unit prefetches instructions and prepares them for execution. This can be done for instructions which are independent of each other. For two instructions that are dependent, the latter instruction cannot be executed until the execution of the first instruction is complete.

Anderson *et al.* have identified three main sources of concurrency limitations which tend to reduce the performance of the pipe [2]. These are: 1) register interlock, 2) branching, and 3) interrupts. Various methods have been introduced to solve these dependency problems [6]. For example, register interlocks can be solved by using forwarding; the sequentialism due to interrupts can be improved by using imprecise interrupts as in IBM 360/91. The most predominant effect on the performance of the memory is due to branching. When a conditional or unconditional jump instruction is encountered, request supply to the memory can either be discontinued until the condition code is set and the target instruction is returned from the memory, or some future (guessed) instructions can be prepared for execution. During the time interval when no request is sent to the memory, the memory is not fully utilized.

None of the previous work considers the effects of dependencies on the memory performance because they are difficult to determine. These effects vary strongly with the configuration of the pipe and the strategies employed in the pipe to resolve them. Request rate to the memory may also decrease for other reasons. For example, in the IBM 360/91 there is a small number of instruction buffers in the CPU that serve as another level of the memory hierarchy. When a small loop occurs such that all the instructions of the loop fit in the instructions buffers, instructions accesses to the memory stop until execution of the loop is completed. Other machines may have different approaches. Therefore, the evaluation of dependency effects for a specific machine is too restrictive.

The objective of our study is to provide the computer architect with a method of finding the most efficient memory configuration to support a pipelined processor when the dependency effects are taken into account. We assume that the pipe can issue a maximum of one request per unit time to the memory. The degree of lookahead or the number of prefetches and the characteristics of benchmark programs are known. The designer has to search for a memory configuration which has the highest utilization and can support the desired degree of lookahead. The search is exhaustive. In this correspondence we compute the memory utilization when the dependency effects are included for different degrees of interleaving and lookahead. Our technique can, therefore, be used by the designer to search for an efficient memory configuration.

The analysis and simulations are based on the performance results of a general memory organization (Organization I), which is described in a separate paper in this issue. Readers should refer to the aforementioned paper for the description of the memory organization and the accompanying assumptions.

In the rest of this correspondence an approximation method is presented to estimate the degradation in memory utilization due to dependencies. Only an approximation is used because otherwise the analysis would not be mathematically tractable. The next section discusses the access characteristics of the pipe. Section III presents an organization of the pipe and the memory. Section IV discusses the variation of the number of requests in the memory buffers. We then present a Markovian model in Section V to evaluate the number of requests in the buffers at the beginning of a dependency. The average memory and buffer utilization are evaluated in Section VI using the stationary probabilities obtained from the Markovian model. We illustrate the technique with a simple example in Section VII and compare the approximations with actual simulations in Section VIII.

II. Access Characteristics of the Instruction Pipe

Consider a conditional jump instruction. When a conditional jump occurs, the condition code is set earlier by an instruction that may still be in the pipe. Until that instruction finishes and sets the condition code, the jump instruction cannot proceed and the memory is idle. It is assumed that the pipe prefetches from both the successful and the unsuccessful branches of the conditional jump, but does not decode the target instruction. If it is an unsuccessful jump, the pipe can proceed after the condition code is set. If it is a successful jump, the memory is idle until both the condition code is set and the prefetched target instruction is returned from the memory. An unconditional jump can be modeled as a successful conditional jump in which the condition code is available immediately. The effect of register interlocks on the memory performance is very small because they can be solved by other methods [6]. Last, an interrupt is the same as a successful conditional jump in which the entire pipe has to be emptied. Therefore, without loss of generality all dependencies can be represented as a successful (the jump is taken) or an unsuccessful conditional jump.

The numerical results we derive in this correspondence are based on a trace of 500 000 instruction executions. The characteristics of these traces have been shown in a separate paper in this issue and will not be repeated here. In Table I we have shown the distribution of the number of instructions executed between two jumps (conditional and unconditional). Using the Kolmogorov–Smirnov test to see if the distance is exponentially distributed, and taking the middle value in the range as the corresponding X-value, it is found that with a critical value of 0.20, the hypothesis of exponential distribution is accepted.

Another point that is typical with the traces is that the average distance between the instruction setting the condition code and the conditional jump instruction at the decode segment is very small (1.7). For unconditional jumps, the condition code is assumed to be available immediately and the jump is taken. Since this distance is very small, we can safely assume that the distance is zero without causing too much error. Further, because the pipe prefetches the successful target

Manuscript received December 4, 1980; revised June 10, 1981. This work was supported in part by Ballistic Missile Defense Contract DASG-60-77-C-0138, the National Science Foundation under Grant MCS 77-27293, and the Purdue Research Foundation under a Summer Faculty Grant.

C. V. Ramamoorthy is with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720.

B. W. Wah is with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

0018-9340/81/1000-0813$00.75

TABLE I
DISTRIBUTION OF NUMBER OF INSTRUCTIONS EXECUTED BETWEEN TWO JUMPS

Number of instructions	% of total occurences	Number of instructions	% of total occurences
1 - 4	23.6	21 - 24	1.0
5 - 8	24.8	25 - 28	0.9
9 - 12	2.3	29 - 32	7.5
13 - 16	24.1	33 - 36	0
17 - 20	14.5	$\geq$ 37	1.3

instructions irrespective of the type of jumps, we can treat all dependent events to behave identically. Specifically, prefetches for the unsuccessful branch have been done before the occurrence of the dependent instruction. When a dependent instruction occurs, the request stream to the memory does not stop until the requests of the target instructions for the successful branch are issued. At this time, requests to the memory stop until the dependency is resolved.

III. ORGANIZATION OF THE PIPELINE AND THE INTERLEAVED MEMORY SYSTEM

The pipeline is assumed to be linear with L segments and it takes equal amount of time to pass through each segment. A parameter of the pipe that varies with the memory configuration is the degree of prefetch. The number of prefetched instructions should be kept as small as possible because when a conditional jump is encountered, one of the two branches is not traversed and, therefore, the prefetched instructions for the nontraversed branch are wasted. On the other hand, the degree of prefetch should be high enough so that the pipe can be kept busy all the time. Let

m = number of memory modules,
b = number of buffers in memory organization,
$u_{m,b}$ = expected utilization of each memory module without prefetches and dependency effects,
$w_{m,b}$ = expected number of waiting memory cycles of the request without prefetches and dependency effects,
$u'_{m,b}$ = expected utilization of memory module with prefetches and without dependency effects,
$w'_{m,b}$ = expected number of waiting memory cycles with prefetches and without dependency effects,
C = memory cycle time,
r = average number of requests generated per instruction executed,
g = number of instructions per instruction word,
f = number of prefetched memory words,
$1/\lambda$ = average number of instructions executed between two dependencies.

We assume, without loss of generality, that $C/m = 1$. This means that all the results are normalized with respect to a subcycle of the memory. Since the pipe can issue at a maximum rate of one request per subcycle, the speed of the pipe is constant. We also assume that the pipe is executing at an average speed that is the same as the memory, that is, at a rate of $(u_{m,b} * m)/(r * C)$ instructions per unit time. Because it takes an average time of W $(= w_{m,b} * C)$ to fetch an instruction, the pipe would have executed $g * f$ instructions in this time interval at a rate of $(u_{m,b} * m)/(r * C)$ if no dependency occurs. Therefore

$$\frac{g * f}{u_{m,b} * m/(r * C)} > w_{m,b} * C.$$

We set

$$f = \left\lceil \frac{w_{m,b} * u_{m,b} * m}{g * r} \right\rceil \tag{1a}$$

where $\lceil x \rceil$ is the smallest integer larger than or equal to x. The value of f established here only uses the average behavior of the memory system. A high value of f would be necessary if the worst case memory parameters are used. From the statistics of the traces, $g = 2.787$ and $r = 0.6 + 1/2.787 = 0.959$.

A parameter related to f is the number of segments in the pipe. With a higher degree of interleaving, the pipe can issue more prefetches and the number of segments increases. If f memory words are prefetched, it means that $g * f$ instructions are prefetched and the pipe length should be $g * f$. A more accurate measure of f is to use the value before we take the ceiling. The required pipe length is

$$L = \left\lceil \frac{w_{m,b} * u_{m,b} * m}{r} \right\rceil. \tag{1b}$$

At any time the maximum number of outstanding requests are f prefetches and the operand fetches from L currently executing instructions. Using a value of L before taking the ceiling, the maximum number of buffers needed is therefore

$$\max b = \left\lceil \left(r - \frac{1}{g}\right) * l + f - u_{m,b} * m \right\rceil. \tag{2}$$

The performance of the MWFMF algorithm on the memory organization[1] with and without prefetches and without the effects of dependencies are shown in Table II. In the execution of instructions in a pipelined computer, the operand address is not known until the corresponding instruction is decoded. That is, the instructions and the corresponding operands are not fetched together. In our simulations we assume that instructions and their corresponding operands are fetched together. The errors introduced due to this assumption are negligible because the correlation between the instruction and data accesses is very small. The memory utilization is higher when prefetches for both the successful and unsuccessful branches are included because the addresses of the prefetches are sequential and this increases the sequentiality in the access stream.

We digress at this point to find the values of m and b for a given value of f. Suppose the computer architect decides that f should be 2. From Table II the following combinations of m and b are feasible: (2,2), (2,3), (4,1), (4,2), (8,1), and (12,1). Applying (1b) and (2), we have obtained the maximum values of b as 3, 4, 2, 3, 2, and 2, respectively. Since all the values of b selected are less than the maximum values of b, all the configurations are feasible. All these different configurations must be evaluated while taking into account the effects of dependencies, and the one with the maximum utilization is chosen. As can be seen from the results in Table IV later, the configuration with $m = 2$ and $b = 2$ has the highest utilization.

The memory organization has m modules, each with constant speed. The performance of this organization is difficult to analyze. In the approximate organization, the memory is assumed to be a single server with a constant service time of rate u_{avg} and a finite buffer space of length b. The service rate u_{avg} is constant and is independent of the number of requests in the buffers. Since in the original organization the memory utilization or the service rate is lower when the number of requests in the buffers is smaller, we expect that u_{avg} to lie between the maximum memory utilization (when the buffers are full) and the minimum utilization (when there is one request in the buffers). The evaluation of u_{avg} is shown in Section V. The service discipline in the buffers in FIFO and the waiting time for a request to be serviced is W $(= m * w_{m,b})$. This means that when a dependency occurs, the maximum time to resolve a dependency is W. Since we treat all dependent events to behave identically, we assume that the dependency will always be resolved in time W after the dependency occurs (time for instruction to pass through the pipe). The arrival rate to the approximate organization is the same as before. The approxi-

[1] The memory organization and the optimal scheduling algorithm—MWFMF—are discussed in a different paper in this issue.

TABLE II
PERFORMANCE OF MEMORY USING MWFMF ALGORITHM WITHOUT THE EFFECTS OF DEPENDENCIES

		With prefetches on the target branch		With prefetches on both the Successful and unsuccessful branches		
m	b	Memory Utilization $u_{m,b}$	Waiting Cycles $w_{m,b}$	# of Prefetch f	Memory Utilization $u'_{m,b}$	Waiting Cycles $w'_{m,b}$
2	1	0.727	1.69	1	0.735	1.68
	2	0.882	2.13	2	0.901	2.11
	3	0.928	2.62	2	0.937	2.60
	4	0.960	3.08	3	0.959	3.07
4	1	0.515	1.49	2	0.548	1.46
	2	0.673	1.74	2	0.715	1.70
	3	0.776	1.97	3	0.807	1.93
	4	0.831	2.20	3	0.858	2.16
8	1	0.385	1.33	2	0.391	1.32
	2	0.533	1.47	3	0.561	1.45
	3	0.612	1.61	3	0.641	1.58
	4	0.686	1.73	4	0.727	1.69
12	1	0.330	1.22	2	0.378	1.22
	2	0.472	1.35	3	0.500	1.33
	3	0.533	1.45	4	0.577	1.43
	4	0.614	1.54	5	0.655	1.51
16	1	0.300	1.21	3	0.305	1.20
	2	0.416	1.30	4	0.415	1.30
	3	0.511	1.37	5	0.514	1.37
	4	0.570	1.44	5	0.564	1.44

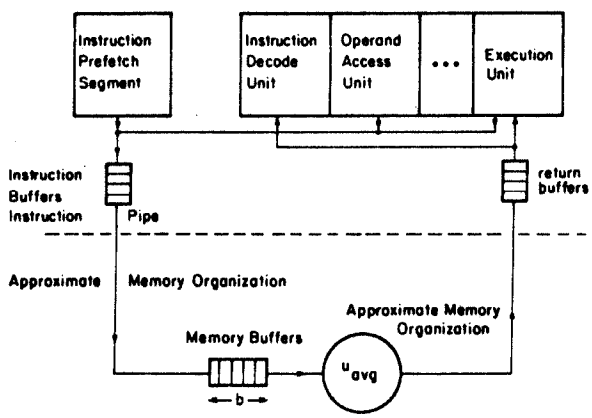

Fig. 1. Approximate instruction pipe-memory organization.

mate memory organization together with the instruction pipe is shown in Fig. 1.

The approximation made here is based on a technique developed in [5], where the performance of a central server model is analyzed by combining the servers into a single server. However, our organization is not directly analyzable with the techniques developed in [5] because the memory modules have a constant service time distribution and the queue size is fixed and finite. In fact, this model is analyzable only with simulations or approximation. We will present the approximation in the next few sections.

IV. Variation of the Number of Requests in the Memory Buffers

The variation of the number of requests in the memory buffers is a discrete process. However, it would be more convenient if the variation is represented as a continuous process. Since only the average performance is sought, a continuous approximation will give a solution that is as good as the discrete process solution.

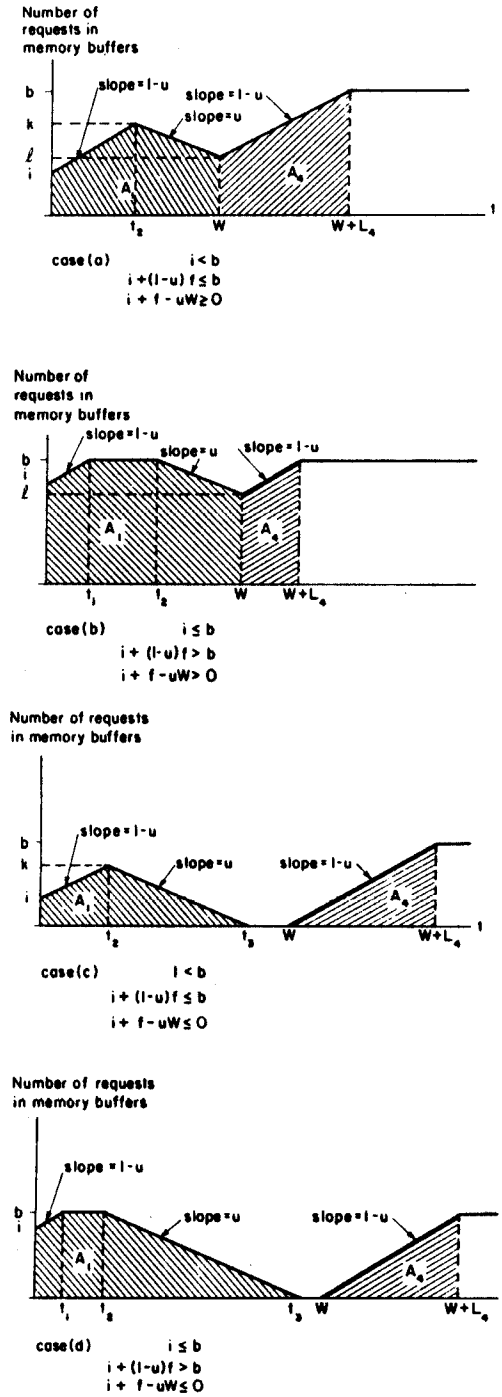

Fig. 2. The variation of the number of requests in the memory buffers as dependency occurs.

There are four possible cases for the variation of the number of requests in the memory buffers as a dependency occurs (Fig. 2). Let us examine these cases in more detail. Suppose that at the time a dependency occurs there are i requests in the buffers. This dependency will be resolved in an average time W. In the meantime, there are still f prefetches to be done before the request stream stops. If $i < b$, then the number of requests in the buffers is increasing at a rate of $1 - u$ because requests are arriving at a rate of one per unit time, while requests are serviced at a rate of u per unit time.[2] Assume that $k = i$

[2] u is the service rate taken at this time. The average service rate is going to be u_{avg}.

$+ (1-u)f$, that is, k is the number of requests in the buffers after the f prefetches have been made while assuming that the number of requests in the buffers continues to increase at a rate of $1-u$. In cases (a) and (c), $k \le b$, while in cases (b) and (d), $k > b$. When $k > b$, the number of requests in the buffers stops to increase as it reaches b and continues to be b until the f prefetches are made. When the buffers are full, the requests arrive at the same rate as they are serviced, that is, u. Using a continuous approximation and referring to Fig. 2

$$t_1 = \begin{cases} f & \text{(cases a, c)} \\ \dfrac{(b-i)}{(1-u)} + \dfrac{(f-(b-i)/(1-u))}{u} = \dfrac{f-b+i}{u}. & \text{(cases b, d)} \end{cases}$$

When all the prefetches have been made, the number of requests in the buffers will be serviced at a rate u for a time $(W - t_1)$. The number of requests in the buffers at time W at which the dependency is resolved, is

$$l = \begin{cases} i + (1-u)f - (W-f)u = i + f - uW & \text{(case a)} \\ b - (W - t_1)u = i + f - uW & \text{(case b)} \\ 0. & \text{(cases, c, d)} \end{cases} \quad (3)$$

l is taken to be zero when $i + f - uW < 0$.

We have, therefore, identified four different cases for the variation of the number of requests in the memory buffers. At time W, the dependency is resolved and the number of requests in the buffers continues to increase until either a new dependency occurs or the number reaches b. In the latter case the number will remain at b until a new dependency occurs and requests will arrive and be serviced at a rate u. In the next section we will present a Markov model that estimates the number of requests in the buffers at the beginning of a dependency.

V. A Markov Model

We present in this section a Markov model that evaluates the stationary probability of the number of requests in the memory buffers at the beginning of a dependency. Since the resultant number of requests in the buffers must be integral at the beginning of a dependency, the possible number of states in the Markov model is at most $b + 1$, where a state indicates the number of requests in the buffers. An entry $p_{i,j}$ in the transition matrix P of the Markov model indicates the probability that the number of requests is i and j at the beginning of the current and the next dependencies. At the beginning of a dependency, the number of requests in the buffers is i. The dependency is subsequently resolved at W (Fig. 2) and new requests continue to enter into the buffers. The number of requests that enter the buffers before the next dependency occurs is exponentially distributed (Section II). The number of requests in the buffers at the next dependency is j. Using the continuous approximation, we take the probability $p_{i,j}$ to be the probability that the resultant number of requests in the buffers is between $j - 0.5$ and $j + 0.5$.

Since l is the number of requests in the buffers at time W after the dependency is resolved ($l = i + f - uW$, cases a, b; $l = 0$, cases c, d), then by using (3)

$$p_{i,j} = \begin{cases} \exp\left[-\lambda \dfrac{\max(j-0.5, l) - l}{1-u}\right] & \\ \quad - \exp\left[-\lambda \dfrac{j+0.5-l}{1-u}\right] & b - 0.5 \ge j + 0.5 > l \\ \exp\left[-\lambda \dfrac{\max(b-0.5, l) - l}{1-u}\right] & j = b \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

It can be verified easily that the process is memoryless, stationary, irreducible, positive recurrent, and aperiodic. Therefore, there exists a unique stationary probability vector $\pi = \{\pi_0, \pi_1, \cdots, \pi_b\}$ such that

$$\pi = \pi P^T \quad (5)$$

where P^T is the transpose of the transition matrix. The probability π_i is the probability that the number of requests in the buffers is i at the beginning of a dependency.

Recall that in the approximate memory organization we have assumed that the utilization is independent of the number of requests in the buffers. However, the utilization actually decreases as the number of requests in the buffers decreases, as evident from the simulation results in Table II. When dependency effects are present, the number of requests in the buffers varies. Therefore, the value of u used in (3) and (4) and in subsequent derivations cannot be the same as the value of $u'_{m,b}$. Instead of using a value of u that is a function of b, we use a simpler measure, the average utilization that can be calculated by first obtaining the memory utilization for the same values of m and f and different values of b. The distribution on the number of requests in the buffers are then evaluated and the value of u is an average utilization weighted over these probabilities.

In order to obtain the memory utilization for the same values of m and f and different values of b, the simulation results in Table II are not sufficient. However, as an approximation the utilization for the same values of m, but different values of f and b in Table II may be used instead. The distribution on the number of requests in the buffers are calculated by using the probabilities in P and summing over the fraction of time that the buffers contain a certain number of requests. If B is a random variable indicating the number of requests in the buffers, then

$$\Pr(B = k) = \sum_i \pi_i \sum_j P_{i,j} \Pr(B = k \mid i, j) \quad (6)$$

where $\Pr(B = k \mid i, j)$ is the fraction of time that B is k condition on i and j, and can be computed directly from Fig. 2. $p_{i,j}$ and π_i are given by (3), (4), and (5), respectively.

Using (6), the average memory utilization for nonzero buffer contents can be calculated by weighting with respect to the nonzero buffer probabilities and normalizing it.

$$u'_{\text{avg}} = \frac{\sum_{k=1}^{b} u'_{m,k} \Pr(B = k)}{\sum_{k=1}^{b} \Pr(B = k)}. \quad (7)$$

To recapitulate, we start by finding the transition matrix (3) and (4) and evaluating the stationary probabilities (5). Then we evaluate the distribution on the number of requests in the buffers (6) and find the average utilization (7). Notice that the value of u is used in (3). Therefore, our technique is recursive. We start by substituting $u_{\text{avg}} = u'_{m,b}$ in (3), and obtain a new value of u'_{avg} from (4), (5), (6), and (7). This is substituted as u_{avg} into (3) again until the difference between u_{avg} and u'_{avg} is very small. It can be shown that u_{avg} and π always converge to a stationary value $\bar{u}_{\text{avg}}$ and $\bar{\pi}$. The average buffer and memory utilization can be found from $\bar{\pi}$. This is shown in the next section.

VI. The Average Buffer and Memory Utilization

A simple approximation on the average buffer utilization can be found by calculating the expected value of the distribution function in (6). However, this is not quite accurate due to the discretization. We will calculate the average buffer and memory utilization using integration.

Referring to Fig. 2(a), the average number of requests in the buffers for case (a) when it starts with i requests at the beginning of a dependency is

$$BN_a(i) = \int_0^{L_4} \frac{A_1 + (i + f - uW)\delta + \dfrac{(1 - \overline{u_{\text{avg}}})\delta^2}{2}}{W + \delta} \lambda e^{-\lambda\delta} d\delta + \int_{L_4}^{\infty} \frac{(A_1 + A_4) + (\delta - L_4)\dfrac{b}{u_{\text{avg}}}}{W + L_4 + \dfrac{\delta - L_4}{u_{\text{avg}}}} \lambda e^{-\lambda\delta} d\delta.$$

The other cases in Fig. 2 can be similarly evaluated. A general formula for the average number of requests in the buffers is

$$\begin{aligned} BN(i) = {} & A_2 e^{\lambda W}\{E_1(\lambda W) - E_1[\lambda(L_4 + W)]\} \\ & + e^{-\lambda L_4}\left[-\frac{L_4(1-\overline{u_{avg}})}{2} - \frac{(1-\overline{u_{avg}})}{2\lambda} - A_3\right] \\ & + \left[\frac{(1-\overline{u_{avg}})}{2\lambda} + A_3\right] \\ & + \lambda\overline{u_{avg}}\left[A_1 + A_4 - \frac{L_4 b}{\overline{u_{avg}}} - b(W + L_5)\right]e^{\lambda\overline{u_{avg}}(W+L_5)} \quad (8) \\ & * E_1\{\lambda[L_4 + \overline{u_{avg}}(W + L_5)]\} \\ & + b\, e^{-\lambda L_4}. \end{aligned}$$

The expressions for the various variables in (8) for the four different cases are shown in Table III. The function $E_1(x)$ is the exponential integral of x [1].

$$E_1(x) = \int_x^\infty \frac{e^{-t}}{t}\,dt \qquad (|\arg x| < \pi).$$

Similarly, the memory utilization can be obtained by first finding the average fraction of time that the buffer content is nonzero. For cases (a) and (b), this fraction is 1. For case (c) the average fraction is

$$\begin{aligned} MF_c(i) = {} & 1 - \int_0^{L_4} \frac{W - f - \dfrac{i + (1-\overline{u_{avg}})f}{\overline{u_{avg}}}}{W + \delta}\lambda e^{-\lambda\delta}d\delta \\ & + \int_{L_4}^\infty \frac{W - f + \dfrac{i - (1-\overline{u_{avg}})f}{\overline{u_{avg}}}}{W + \dfrac{\delta - L_4}{\overline{u_{avg}}}}\lambda e^{-\lambda\delta}d\delta. \end{aligned}$$

Case (d) can be similarly evaluated. A general formula for the average fraction of time of nonzero buffer contents is

$$\begin{aligned} MF = {} & 1 - L_0 e^{\lambda W}E_1(\lambda W) - E_1\left[\lambda\left(\frac{b}{1-u} + W\right)\right] \\ & - L_0 u e^{\lambda(Wu-b)}E_1\left[\lambda\left(\frac{b}{1-u} + Wu - b\right)\right]. \quad (9) \end{aligned}$$

The expressions for L_0 for the four cases are shown in Table III.

By weighting over the stationary probabilities $\bar{\pi}$, we have the average buffer utilization BU and the average fraction of time of nonzero buffer contents MF as

$$BU = \frac{1}{b}\sum_{i=0}^{b}\bar{\pi}(i)BN(i) \quad (10)$$

$$MF = \sum_{i=0}^{b}\bar{\pi}(i)MF(i). \quad (11)$$

The resulting memory utilization is MU

$$MU = u'_{m,b} * MF \quad (12)$$

where $u'_{m,b}$ is the memory utilization shown in Table II without the dependency effects. We have, therefore, evaluated the average buffer and memory utilization. By using Little's Formula [4], the average waiting time can also be calculated.

We have obtained the throughput of the memory in terms of its utilization. To obtain the throughput of the pipe, we have to eliminate the unnecessary fetches performed on the unsuccessful branch. Since we know f, r, and the probability of an instruction being a jump P_j, the throughput of the pipe is $MU/(r + P_j f)$ instructions per unit time. Before the approximation is compared with the simulations, a simple example is shown to illustrate the technique.

VII. A Simple Example to Illustrate the Technique

Suppose $m = 4$, $b = 2$. From Table II we have $u'_{4,1} = 0.548$, $u'_{4,2} = 0.715$, $W = 4 * 1.7 = 6.8$. f is 2 (1) and $\lambda = 1/11.4$.

TABLE III
Expressions for Various Variables in (8) and (9) (Note That $u = u_{avg}$ in All These Expressions)

Case (a)	$A_1 = W(i + f) - \frac{f^2 + uW^2}{2}$ $A_2 = A_1\lambda + \frac{(1-u)\lambda W^2}{2} - (i + f - uW)\lambda W$ $A_3 = (i + f - uW) - \frac{(1-u)W}{2}$ $A_4 = \frac{b^2 - (i + f - uW)^2}{2(1-u)}$ $\qquad L_0 = 0$ $L_4 = \frac{b - i - f + uW}{1-u}$ $\qquad L_5 = \frac{-L_4(1-u)}{u}$
Case (b)	$A_1 = \frac{b^2 - i^2}{2(1-u)} + \frac{b(f - fu - b + i)}{u(1-u)} + \frac{b^2 - (f + i - uW)^2}{2u}$ expressions for A_2, A_3, A_4, L_0, L_4 and L_5 are the same as Case (a)
Case (c)	$A_1 = \frac{[2i + (1-u)f]f}{2} + \frac{[i + (1-u)f]^2}{2u}$ $A_2 = A_1\lambda + \frac{(1-u)\lambda W^2}{2}$ $\qquad A_3 = -\frac{(1-u)W}{2}$ $A_4 = \frac{b^2}{2(1-u)}$ $\qquad L_0 = \lambda\left[W - f - \frac{i + (1-u)f}{u}\right]$ $L_4 = \frac{b}{1-u}$ $\qquad L_5 = \frac{-L_4(1-u)}{u}$
Case (d)	$A_1 = \frac{b^2 - i^2}{2(1-u)} + \frac{b(f - fu - b + i)}{u(1-u)} + \frac{b^2}{2u}$ expressions for A_2, A_3, A_4, L_0, L_4 and L_5 are the same as Case (c).

Starting with $u_{avg} = u'_{4,2}$, we have from (3) and (4) a transition matrix

$$P = \begin{bmatrix} 0.143 & 0.227 & 0.630 \\ 0.143 & 0.227 & 0.630 \\ 0.143 & 0.227 & 0.630 \end{bmatrix}$$

and the stationary probability vector π (5)

$$\pi = [0.143 \quad 0.227 \quad 0.630].$$

Summing over the fraction of time that the buffers contain B requests and applying (6) we have

$$\Pr(B = 0) = 0.226$$

$$\Pr(B = 1) = 0.345$$

$$\Pr(B = 2) = 0.429.$$

From (7), u'_{avg} is found to be 0.641. Since u'_{avg} differs from the previous u_{avg}, we apply (3)–(7) over again substituting u'_{avg} for u_{avg}.

In summary, starting with $u = u_{avg} = 0.715$, the sequence of u_{avg}'s obtained is 0.641, 0.651, and 0.650. We stop at this point and take $u_{avg} = 0.650$. The stationary probability vector at this point is

$$\bar{\pi} = [0.118 \quad 0.196 \quad 0.686].$$

Applying (8) and (9), we have

$$BN(0) = 0.849 \qquad MF(0) = 0.744 \qquad \text{(case c)}$$

$$BN(1) = 1.114 \qquad MF(1) = 0.849 \qquad \text{(case c)}$$

$$BN(2) = 1.412 \qquad MF(2) = 0.955. \qquad \text{(case d)}$$

Finally, applying (10)–(12), we have

$$BU = 0.643 \qquad MU = 0.650.$$

TABLE IV
PERFORMANCE OF MEMORY USING MWFMF ALGORITHM WITH THE EFFECTS OF DEPENDENCIES

		Simulation			Approximation		
m	b	Memory Utilization	Waiting Cycles	Buffer Utilization	Memory Utilization MU	Waiting Cycles WU	Buffer Utilization BU
2	1	0.700	1.050	0.770	0.671	1.057	0.748
	2	0.828	1.302	0.664	0.873	1.233	0.640
	3	0.838	1.429	0.519	0.864	1.309	0.466
	4	0.899	1.572	0.482	0.899	1.439	0.422
4	1	0.532	0.667	0.888	0.528	0.670	0.887
	2	0.652	0.811	0.731	0.650	0.745	0.643
	3	0.746	0.896	0.643	0.770	0.881	0.648
	4	0.760	0.999	0.569	0.789	0.922	0.530
8	1	0.366	0.393	0.784	0.346	0.427	0.837
	2	0.500	0.473	0.695	0.516	0.472	0.717
	3	0.574	0.549	0.649	0.567	0.507	0.578
	4	0.650	0.557	0.561	0.673	0.556	0.580
12	1	0.364	0.250	0.730	0.336	0.286	0.817
	2	0.426	0.322	0.610	0.405	0.350	0.647
	3	0.482	0.373	0.558	0.489	0.388	0.596
	4	0.583	0.402	0.529	0.595	0.386	0.540
16	1	0.272	0.222	0.692	0.258	0.260	0.815
	2	0.358	0.284	0.634	0.363	0.309	0.717
	3	0.437	0.314	0.585	0.463	0.324	0.645
	4	0.466	0.339	0.515	0.489	0.344	0.550

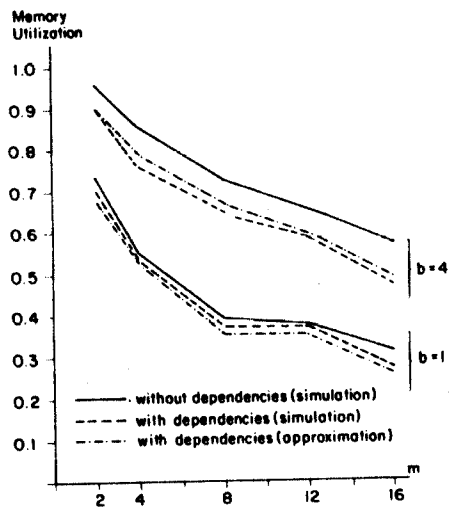

Fig. 3. A plot of the decrease in memory utilization with and without the effects of dependencies.

VIII. COMPARISON WITH SIMULATION RESULTS

In order to see how good the approximation is, it is necessary to compare the approximation results against the simulation results. A simulation program is written in ASPOL with a pipe of L segments (1b) which prefetches f memory words (1a) ahead of time. The execution trace is used as input to the pipe. At the start of a dependency, f artificial instruction prefetches on the successful branch are generated (the f instruction fetches on the unsuccessful branch have already been issued earlier). If the jump is successful, then the request stream will be reestablished when the condition code is set and the first jump target instruction is fetched. If the conditional jump is unsuccessful, it waits for a time W, which is the time it takes for the condition code to be set.

The simulation is run for a maximum of 500 000 instruction executions, but may stop earlier if the results stabilize. The simulation results together with the approximations are shown in Table IV. It is seen that the approximate memory utilization is very close to the simulation results, with less than 5 percent error under most of the cases. The buffer utilization and the waiting cycles are not as accurate because we assume that the memory utilization remains the same irrespective of the number of the requests in the buffers. The utilization is actually higher when the buffers are full and smaller otherwise. Although an average value u_{avg} has been calculated, it is not as accurate as exact numerical integration. However, we are still satisfied with the approximation because the approximate memory utilization, which is a more important measure on the memory performance than buffer utilization, is very close to the simulation results. The degradation in memory utilization due to dependencies is further illustrated in Fig. 3. It is seen that dependencies cause a degradation in memory utilization, and is more pronounced when the buffer size is large (a larger number of prefetches). Furthermore, the curves shown are not smooth because of the different degrees of prefetch in each case. The readers must be cautioned that the two sets of curves in Fig. 3 cannot be directly compared because they correspond to pipes with different degrees of prefetch.

The above estimations only give an average value for the performance. It also indicates the fraction of the memory's unused throughput. In fact, if the memory can be utilized in some other way (e.g., for peripheral processing) when a dependency occurs, the degradation may not be so significant. The above analysis also reveals the fact that when the occurrences of dependent requests are frequent, it is not beneficial to use a pipelined computer in a batch mode. High degree of program interleaving using multiprogramming would help in reducing the degradation due to dependencies.

REFERENCES

[1] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions*. New York: Dover, 1965.
[2] D. W. Anderson, F. J. Sparacio, and R. M. Tomasulo, "The IBM system 360 model 91: Machine philosophy and instruction handling," *IBM J. Res. Develop.*, pp. 8-24, Jan. 1967.
[3] D. Y. Chang *et al.*, "On the effective bandwidth of parallel memories." *IEEE Trans. Comput.*, vol. C-26, pp. 480-490, May 1977.
[4] J. D. C. Little, "A proof of the queuing formula $L = \lambda W$," *Oper. Res.*, vol. 9, pp. 383-387, 1961.
[5] C. H. Sauer and K. M. Chandy, "Approximate analysis of central server models," *IBM J. Res. Develop.*, pp. 301-313, May 1975.
[6] R. M. Tomasulo, "An efficient algorithm for exploiting multiple arithmetic units," *IBM J. Res. Develop.*, pp 25-33, Jan. 1967.

Correction to "The MPG System: A Machine-Independent Efficient Microprogram Generator"

The above paper[1] in the June 1981 issue of the TRANSACTIONS was incorrectly identified as a Survey Paper. This paper is a Regular Paper and should have appeared under the heading "Microprocessors."

[1] T. Baba and H. Hagiwara, *IEEE Trans. Comput.*, vol. C-30, pp. 373-395, June 1981.

0018-9340/81/1000-0818$00.75