1979 Conference on Simulation, Measurement and Modeling of Computer Systems

AN ANALYSIS OF CENTRAL PROCESSOR - INPUT-OUTPUT PROCESSOR CONTENTION

William D. Strecker
Digital Equipment Corporation
Tewksbury, Massachusetts 01876

Abstract:

Most computer systems have separate central (CPU) and input-output (IOP) processors to permit simultaneous computation and input-output (I/O). It is conventional in such systems to avoid any loss of I/O data by granting the IOP priority over the CPU for memory service. Although this priority discipline is simple to implement it may result in a maximum degradation of CPU performance. In this discussion an analysis of the IOP priority discipline is given together with an analysis of other priority disciplines which require the buffering of IOP requests and results are given showing that only a small amount of buffering is required to produce a noticeable improvement in CPU performance.

Index Terms: CPU, input-output, processor, memory system, contention, I/O interference, priority discipline.

Introduction

In most computer systems input-output (I/O) is not handled directly by the central processing unit (CPU) but rather by separate hardware which is referred to by various names (channel, I/O controller, etc.), but is termed here an input-output processor (IOP). With such a structure it is possible for I/O to take place concurrently with computation; but because the CPU and the IOP contend with one another for main memory (MM) cycles, the performance of either the CPU, the IOP, or both may be degraded with respect to their performance if operating alone.

Because I/O traffic often consists of transfers between MM and electro-mechanical peripheral devices which do not have a random access character, there is a relatively large time penalty associated with a lost I/O datum. For this reason it is conventional to grant the IOP priority over the CPU for MM service. This priority discipline ensures that no I/O data are lost due to CPU utilization of MM and is relatively simple to implement. Offsetting the advantage of simplicity, however, is the disadvantage that for a given average I/O transfer rate, the CPU may suffer a maximum performance degradation. With the addition of hardware providing buffer (or queue) space for IOP requests and implementing different priority rules it is possible to ensure both that no I/O data are lost and that CPU performance degradation is minimized.

This is possible due to the different nature of CPU and IOP requests.[1] When the CPU accesses MM it generally[1] waits until after that request is serviced before issuing its next request. Any delays in servicing its requests due to IOP utilization of MM simply cause the CPU to wait. On the other hand, IOP requests to MM come at a rate determined by the characteristics of the I/O devices served by the IOP. The amount of time elapsing between the issuance of an IOP request and its servicing often does not (in a local sense) affect the performance of the IOP given that all its requests are eventually serviced.

Whether or not the time elapsing between the issuance and servicing of an IOP request is important depends on several factors. These include whether the transfer is an I/O read (transfer from MM to the I/O device) or write (transfer from the I/O device to MM), whether the I/O device has a random access character, and to what extent the I/O device itself is buffered. Since it is often the case that a majority of I/O traffic is I/O writes and I/O devices are buffered, the above conceptual-ization of the IOP is probably a reasonable one.

[1] CPUs with certain features such as "look-ahead" may have several out-standing MM requests at a given time.

It is the purpose of this discussion to analyze the effects of I/O on CPU performance for the conventional IOP priority discipline and for other priority disciplines which involve the buffering of IOP requests. A one CPU, one IOP system is considered together with some special assumptions about its behavior.

Because any realistic evaluation of computer system performance must take into account degradation of CPU performance by I/O, there have been a number of previous investigations of the topic. Shemer and Gupta [1] analyzed the performance of a one CPU system (with "look-ahead") for the conventional IOP priority discipline. Skinner and Asher [2] analyzed multiple CPU systems for the same priority discipline. By another method, this author [3] analyzed the performance of a multiple CPU system for both the IOP priority discipline and another priority discipline involving one level of IOP buffering. Pirtle [4] studied by simulation several different priority and buffering schemes including ones where certain I/O devices (displays for example) could suffer a limited loss of data. Pirtle's paper is recommended as a good introduction to the issues of I/O handling in computer systems.

The Model

The general structure of the system considered is given in Figure 1. The CPU and the IOP are connected to MM through a storage control unit (SCU). The SCU contains one buffer or queue position for a CPU request, one or more queue positions for IOP requests, and priority resolution hardware which determines whether a CPU or an IOP request receives MM service.

The operation of the system is assumed to be synchronized to the cyclic operation of MM as indicated in Figure 2. The MM access and cycle time is t_m. Processor requests (both CPU and IOP) are assumed to arrive at the beginning of an MM cycle.

A processor is termed queued if it is either waiting for or receiving MM service. The CPU is modelled by assuming that when it is not queued there is a stationary probability π_p that it requests each MM cycle. This is equivalent to assuming a geometric distribution for CPU processing time. Suppose T_p is a random variable equal to the CPU processing time between the satisfaction of its current MM request and the issuance of the next. Then the (point) density function for T_p is:

$$f_{T_p}(kt)=\pi_p(1-\pi_p)^k,\ k=0,\ 1,\ \ldots, \quad (1)$$

and the expected value of T_p, t_p, is:

$$t_p = \sum_{k=0}^{\infty} kt_m f_{T_p}(kt_m) = \frac{1-\pi_p}{\pi_p} t_m. \quad (2)$$

Equation (2) may be rewritten to express π_p in terms of t_p:

$$\pi_p = \frac{t_m}{t_m+t_p}. \quad (3)$$

Two distinct models are assumed for the IOP. Most I/O devices are characterized by a constant I/O transfer rate determined by their physical properties. If there is but a single I/O device or a majority of the I/O traffic comes from a single device, then the "regular" model for the IOP is appropriate. In this case the IOP is assumed to issue an MM request once every c cycles (with c an integer greater than one). If there are a number of concurrently active I/O devices each contributing significantly to the aggregate I/O traffic, then the IOP requests to MM take on more of a random character. For this case the "random" model of the IOP is appropriate. In this case there is assumed to be a probability π_o that the IOP requests each MM cycle. The probability π_o is equal to the ratio of the total average I/O rate R_o to the rate at which MM cycles are available $1/t_m$, thus:

$$\pi_o = R_o t_m. \quad (4)$$

If an S cycle sequence of MM cycles is considered and I is a random variable equal to the number of MM cycles requested by the IOP, then the density function of I is clearly:

$$f_I(k) = \binom{S}{k} \pi_o^k (1-\pi_o)^{S-k}. \quad (5)$$

Thus it can be seen that the random model for the IOP is equivalent to a binomial stochastic process with parameter π_o. The binominal process is the discrete analog of the Poisson process.

Several different priority disciplines are considered for the SCU. The first, of course, is the conventional IOP priority discipline for both regular and random I/O. Secondly, a CPU priority discipline is considered for both regular and random I/O where an infinite number of IOP request queue

positions are provided. Lastly, a more realistic variable priority discipline is considered. For random I/O a fixed, arbitrary number of IOP buffer positions are available. Until all of these positions are filled the CPU has priority; otherwise the IOP has priority. For regular I/O the same variable priority scheme is considered, however analytic difficulties force only the case of a single queue position to be examined.

In the following analysis the average CPU processing rate R_p and the average number of queued IOP requests $E[n]$ will be determined as a function of π_p, π_o, and t_m for the various priority disciplines. In order to avoid the issue of CPU instruction formats, R_p is measured in terms of MM references by the CPU per unit time.

Analysis

No I/O: In order to establish a base line for CPU performance, R_p is determined in the absence of I/O. Since there is a probability π_p that each MM cycle is used by the CPU and MM cycles are available at rate $1/m$, the average CPU rate is simply:

$$R_p = \frac{\pi_p}{t_m} . \qquad (6)$$

IOP priority-random I/O: With this priority discipline whenever there are both an IOP and a CPU request queued for the same MM cycle, the IOP request is serviced. For random I/O there is a probability π_o that each MM cycle is requested by the IOP. Hence when the CPU requests MM there is a probability $1-\pi_o$ that the CPU request is serviced immediately or in other words in one MM cycle. For the CPU request to be serviced in two MM cycles the first MM cycle must have been requested by the IOP and the second not requested; the probability of this event is $\pi_o(1-\pi_o)$. In general for the CPU request to be serviced in k MM cycles there must be (k-1) MM cycles requested by the IOP followed by one not requested. If T_e is a random variable equal to the time elapsing between the issuance of a CPU request and its completed service, the density function for T_e is:

$$f_{T_e}(kt_m) = \pi_o^{k-1}(1-\pi_o);\ k = 1,2,\ldots . \qquad (7)$$

The expected value of T_e, t_e, is:

$$t_e = \sum_{k=1}^{\infty} kt_m f_{T_e}(kt_m) \qquad (8)$$

$$= \frac{t_m}{1-\pi_o} .$$

Since the average CPU processing time between the satisfaction of MM request and the issuance of the next is t_p, the average time between MM references for the CPU is:

$$t = t_e + t_p$$

$$= \frac{t_m}{1-\pi_o} + \frac{1-\pi_p}{\pi_p} t_m$$

$$= \frac{1 + \pi_o\pi_p - \pi_o}{\pi_p(1-\pi_o)} t_m . \qquad (9)$$

R_p is just $1/t$, hence:

$$R_p = \frac{\pi_p}{t_m}\left(1 - \frac{\pi_o\pi_p}{1 + \pi_o\pi_p - \pi_o}\right) . \qquad (10)$$

Clearly as $\pi_o \to 0$, $R_p \to \pi_p/t_m$ and as $\pi_o \to 1$, $R_p \to 0$ which corresponds with an intuitive notion of how the system should be behaving. For random I/O it is convenient to define $E[n]$ as the average number of IOP requests queued just after the beginning of an arbitrary MM cycle.

Since there is a probability π_o that each MM cycle is requested by the IOP, the average number of queued IOP requests is just:

$$E[n] = \pi_o . \qquad (11)$$

At times it may be of interest to know the average time elapsing between the issuance of an IOP request and its completed service. For the case of random I/O this may be obtained rather simply. A result first given by Little [5] shows that under rather general conditions this elapsed time is given by the ratio of the average queue length to the average arrival rate to the queue. For random I/O this ratio is $E[n]/(\pi_o/t_m)$. Applying this to the result just derived, an elapsed time of t_m results which of course is just what would be expected for the IOP priority discipline.

CPU priority-random I/O: With this priority discipline whenever there are CPU and IOP requests queued for the same MM cycle, the CPU request is serviced. Because the IOP, unlike the CPU, continues to issue MM requests even when its previous requests have not been

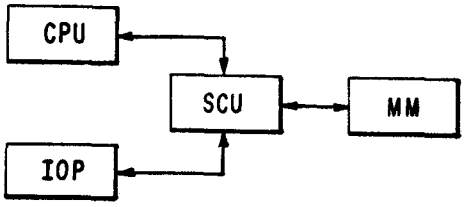

FIG. 1 SYSTEM STRUCTURE

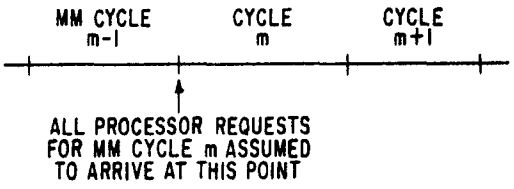

FIG. 2 SYSTEM OPERATION

serviced, the number of queued IOP requests can conceivably become arbitrarily large. Because the CPU always has priority, the CPU processing rate is just that given by (6):

$$R_p = \frac{\pi_p}{t_m}, \qquad (12)$$

and the quantities of interest to determine are the maximum rate of I/O permissible and the value of E[n].

Let us characterize the state of the system by S(k) where k is the number of IOP requests queued just after the beginning of an arbitrary MM cycle. Suppose the system was in state S(i) just after the beginning of the last MM cycle and consider the probability P_{ij} that the system is in state S(j) just after the beginning of the current MM cycle. Suppose i is greater than zero. If j=i-1 then there must have been neither a CPU request for the last MM cycle or an IOP request for the current MM cycle. If j=i there must have been either a CPU request for the last MM cycle and no IOP request for the current MM cycle or no CPU request for the last cycle and an IOP request for the current cycle. If j=i+1 there must have been both a CPU request for the last MM cycle and an IOP request for the current MM cycle. Summarizing the above and assigning the appropriate probabilities to the events given yields:

$$p_{ij} = \begin{cases} (1-\pi_p)(1-\pi_o), & j=i-1 \\ \pi_p(1-\pi_o) + \pi_o(1-\pi_p), & j=i \\ \pi_o\pi_p, & j=i+1 \end{cases}, \quad i>0. \qquad (13a)$$

Since the number of queued IOP requests cannot change by more than one for each MM cycle:

$$p_{ij} = 0, \quad |j-1|>1 \qquad (13b)$$

Finally:

$$p_{00} = 1-\pi_o \text{ and:} \qquad (13c)$$

$$p_{01} = \pi_o, \qquad (13d)$$

since it is only required that an IOP request be not made or made respectively for the current MM cycle.

It is assumed that the system is in equilibrium such that there exist equilibrium probabilities p(k) for the states S(k). (For a discussion of equilibrium in queuing systems see [6].) The usual equation relating the equilibrium probabilities is:

$$p(j) = \sum_{i=0}^{\infty} p_{ij}p(i). \qquad (14)$$

Now let us introduce the following notation:

$$a_0 = (1-\pi_o)(1-\pi_p) \qquad (15a)$$

$$a_2 = \pi_o\pi_p \qquad (15b)$$

$$a_1 = \pi_o(1-\pi_p) + \pi_p(1-\pi_o) \qquad (15c)$$

$$= 1-a_0-a_2.$$

Substituting (13) and (15) into (14) gives the following set of equations:

$$p(j)=a_0p(j+1) + (1-a_0-a_2)p(j) \quad (16a)$$
$$+a_2p(j-1),\ j>1,$$

$$p(1)=a_0p(2) + (1-a_0-a_2)p(1) \quad (16b)$$
$$+\pi_o p(0)$$

$$p(0)=(1-\pi_o)p(0) + a_0p(1). \quad (16c)$$

Equation (16a) is a second order difference equation in p(j) and may be solved in the usual manner [7] by assuming a solution of the form:

$$p(j) = A\alpha^j,\ j = 1,2,\ldots, \quad (17)$$

with A a constant.

Substituting (17) in (16a) gives the characteristic equation:

$$0 = a_0\alpha^2-(a_0+a_2)\alpha + a_2$$
$$= (a_0\alpha-a_2)(\alpha-1) \quad (18)$$

For equilibrium to exist there must be a root α of (18) such that $\emptyset < \alpha < 1$. Hence:

$$\alpha = \frac{a_2}{a_0} < 1 \quad (19a)$$

or:

$$\frac{\pi_p\pi_o}{(1-\pi_o)(1-\pi_p)} < 1 \quad (19b)$$

which may be written:

$$\pi_o < 1-\pi_p. \quad (19c)$$

Equation (19c) is simply the reasonable result that the sum of the probabilities that the CPU and the IOP request each MM cycle cannot exceed one. The value of p(0) can be obtained from (16b) or (16c)[2]; here (16c) is employed:

$$p(0) = \frac{a_0p(1)}{\pi_o}$$
$$= \frac{a_0A}{\pi_o}\frac{a_2}{a_0}$$
$$= A\pi_p. \quad (20)$$

The value of A is determined by the requirement that the probabilities p(k) sum to one:

$$1 = \sum_{k=0}^{\infty} p(k)$$
$$=A(\pi_p + \sum_{k=1}^{\infty}\alpha^k)$$
$$=A(\pi_p + \frac{\alpha}{1-\alpha}) \quad (21)$$

or

$$A = \frac{1-\alpha}{(1-\alpha)\pi_p+\alpha} \quad (22)$$

E[n] is obtained from the expression for p(k):

$$E[n]= \sum_{k=0}^{\infty} kp(k)$$
$$= A \sum_{k=1}^{\infty} k\alpha^k$$
$$= \frac{\alpha}{1-\alpha}\frac{1}{\alpha+\pi_p(1-\alpha)}. \quad (23)$$

From this it is seen clearly that as $\alpha\to0$, $E[n]\to0$ and as $\alpha\to1$, $E[n]\to\infty$.

Variable priority - random I/O with fixed IOP queue space: For this priority discipline the rule is implemented that until the IOP queue is filled the CPU has priority and otherwise the IOP has priority. This is handled as an extension of the previous analysis and the same notation will apply here. Suppose there are ℓ ($\ell > 2$) queue positions. Then the system can be in states $S(k)$, $k=\emptyset, 1, 2, \ldots \ell$. If $p(\ell)$ is the probability of being in state $S(\ell)$, then the CPU will run at a rate specified by (6), π_p/t_m, a fraction $1-p(\ell)$ of the time and at rate zero a fraction $p(\ell)$ of the time. Thus the average CPU rate is:

$$R_p = (1-p(\ell))\frac{\pi_p}{t_m}. \quad (24)$$

---

[2]There is one redundant equation in the set specified by (16).

The transition probabilities p_{ij} are similar to those given in the previous case except obviously $p_{ij}=\emptyset$ for $i>\ell$ or $j>\ell$ and $p_{\ell(\ell-1)}= 1-\pi_o$ and $p_{\ell\ell} = \pi_o$. The latter two arise because the CPU is blocked when the system is in state $s(\ell)$. Substituting these p_{ij} in (14) gives the following set of equations:

$$p(j) = a_0 p(j+1) + (1-a_0-a_2)\, p(j) + a_2 p(j-1), \quad 1<j<\ell-1, \tag{25a}$$

$$p(\ell) = \pi_o p(\ell) + a_2\, p(\ell-1) \tag{25b}$$

$$p(\ell-1) = (1-\pi_o)\, p(\ell) + (1-a_0-a_2)\, p(\ell-1) + a_2\, p(\ell-2) \tag{25c}$$

$$p(1) = a_0 p(2) + (1-a_0-a_2)\, p(1) + \pi_o p(0) \tag{25d}$$

$$p(0) = (1-\pi_o)\, p(0) + a_0\, p(1). \tag{25e}$$

The equations (25 a, d, e) are the same as those given by (16) for the same arguments of $p(\ )$. Hence $p(j)$, $\emptyset < j < \ell$ must be of the same form as given by (17) and (19a) above. (The constant A is different of course.) The probability $p(\ell)$ can be expressed in terms of $p(\ell-1)$ by using either (25b) or (25c). Using (25b):

$$p(\ell) = \frac{a_2}{1-\pi_o}\, p(\ell-1)$$

$$= A(1-\pi_p)\left(\frac{a_2}{a_0}\right)^{\ell} \tag{26}$$

The factor A is determined by the usual requirement:

$$1 = \sum_{k=0}^{\ell} p(k)$$

$$= A\left(\pi_p + \sum_{k=1}^{\ell-1} \alpha^k + (1-\pi_p)\alpha^{\ell}\right)$$

$$= A\left(\pi_p + \frac{\alpha}{1-\alpha}\right)(1-\alpha^{\ell}). \tag{27}$$

Hence:

$$A = \frac{1}{1-\alpha^{\ell}} \cdot \frac{1-\alpha}{\pi_p(1-\alpha)+\alpha} \tag{28}$$

E[n] can now be determined:

$$E[n] = \sum_{k=0}^{\ell} k p(k)$$

$$= A \sum_{k=1}^{\ell-1} k\alpha^k + \ell(1-\pi_p)\alpha^{\ell}. \tag{29a}$$

After considerable reduction this becomes:

$$E[n] = A\,\frac{\alpha(1-\alpha^{\ell})}{(1-\alpha)^2} - \ell\alpha^{\ell}\,\frac{\pi_p+(\alpha-1)\pi_p}{1-\alpha} \tag{29b}$$

Substituting for A, the following results:

$$E[n] = \frac{\alpha}{1-\alpha}\,\frac{1}{\pi_p(1-\alpha)+\alpha} - \frac{\ell\alpha^{\ell}}{1-\alpha^{\ell}}. \tag{29c}$$

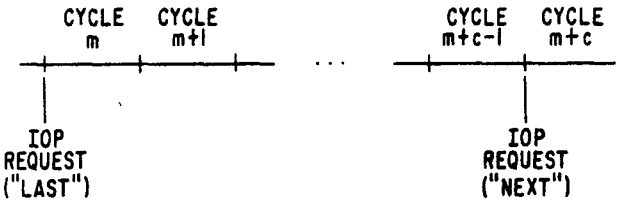

FIG. 3 SYSTEM OPERATION WITH REGULAR I/O

This expression should be compared with (23) for the CPU priority case. As $\ell\to\infty$ the average queue size for the variable priority case approaches the average queue size for the CPU priority case as would be expected intuitively.

IOP Priority - regular I/O: Now the various priority disciplines will be examined for regular I/O. Recall that regular I/O is characterized by an IOP request once every c MM cycles. A diagram of this is given in Figure 3. The IOP requests MM cycles m, m+c, m+2c, and so forth. Now consider the c cycle sequence of MM cycles m through m+c-1. Since the IOP has priority, a CPU request for cycle m is serviced during cycle m+1. The probability cycle m+1 services a CPU request is the probability the CPU requested either cycle m or cycle m+1 (but not both since that is impossible) and is thus $2\pi_p - \pi_p^2$. The probability that any specified one of the remaining c-2 cycles of the sequence is used by the CPU is π_p. Thus the average number of cycles used during the c cycle sequence is $(2\pi_p - \pi_p^2) + (c-2)\,\pi_p$ and the average CPU rate is:

$$R_p = \frac{2\pi_p - \pi_p^2 + (c-2)\pi_p}{ct_m}$$

$$= \frac{\pi_p}{t_m}\left(1-\frac{\pi_p}{c}\right), c > 1. \qquad (30)$$

As $c \to \infty$ (the I/O rate goes to zero) the expression for R_p given by (30) approaches (6) which gives R_p for the no I/O case.

For regular I/O it is convenient to define E[n] as the average number of IOP requests queued just after the arrival of an IOP request. Then for the IOP priority case, clearly:

$$E[n] = 1. \qquad (31)$$

The average elapsed time between the issuance of an IOP request and its completed service cannot be obtained so simply for regular I/O as for random I/O. Little's result [5] requires that the average number of queued IOP requests be known where the average is taken over all MM cycles - not just over those which receive IOP requests. Since the average number of queued IOP requests where the average is taken over MM cycles which receive IOP requests is obviously higher than where the average is taken over all MM cycles, the ratio $E[n]/(1/ct_m) = cE[n]t_m$ gives an upper bound on the average elapsed IOP waiting time for the regular I/O case.

CPU Priority - regular I/O: Since the CPU always has priority, the average CPU rate will be just that given by (6):

$$R_p = \frac{\pi_p}{t_m}, \qquad (32)$$

and again the analysis focuses on the determination of E[n] and the maximum permissable I/O rate.

Let us characterize the state of the system by S(k), k = 1, 2, ... , where k is the number of queued IOP requests measured just after the arrival of an IOP request. Refer to Figure 4. (Note the difference in definition here from that used for the random I/O analysis. There the state was defined for all MM cycles; here it is defined only for MM cycles which receive IOP requests.) Suppose there were i IOP requests queued just after the arrival of the last IOP request and consider the probability p_{ij} that there are j IOP requests queued just after the arrival of the next IOP request. Referring again to Figure 4, suppose there are r IOP requests serviced during cycles m through m+c-1. Then there are i-r+1 IOP requests queued after the beginning of cycle m+c. Setting i-r+1 equal to j, r is equal to i+1-j. If $j > 1$, then for i+1-j requests to have been serviced during the c cycle sequence, the CPU must have requested c-(i+1-j) MM cycles and not requested i+1-j. Since the i+1-j cycles can be chosen in $\binom{c}{i+1-j}$ ways, p_{ij} can be written:

$$p_{ij} = \binom{c}{i+1-j}(1-\pi_p)^{i+1-j}\pi_p^{c-(i+1-j)}, \quad j>1 \qquad (33)$$

The system is assumed to be in equilibrium such that there exist equilibrium probabilities p(k) for the states S(k). The equation relating the equilibrium probabilities is:

$$p(j) = \sum_{i=1}^{\infty} p_{ij}p(i). \qquad (34)$$

It should be noted at this point that there is no expression given for p_{i1}. The expression for p_{i1} is much more complex than that for p_{ij}, $j > 1$. However, the set of equations (34) contains one redundant equation. In the present analysis the equation for p(1) is not used and hence p_{i1} is not needed.

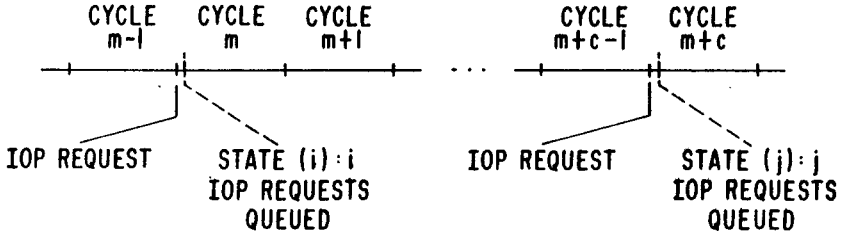

FIG. 4 STATE DETERMINATION FOR REGULAR I/O

Let us again assume a solution of the form:

$$p(k) = A\alpha^k, \quad k=1, 2, \ldots, \tag{35}$$

where A is a constant. Substituting (33) and (35) in (34) gives:

$$A\alpha^j = \sum_{i=1}^{\infty} p_{ij} A\alpha^i \tag{36}$$

$$= A \sum_{i=1}^{\infty} \binom{c}{i+1-j} (1-\pi_p)^{i+1-j} \pi_p^{c-(i+1-j)} \alpha^i, \; j>1.$$

After multiplying both sides of (36) by $A^{-1}\alpha^{1-j}$, the following results:

$$\alpha = \sum_{i=1}^{\infty} \binom{c}{i+1-j} (\alpha-\alpha\pi_p)^{i+1-j} \pi_p^{c-(i+1-j)}. \tag{37}$$

Since $j > 1$ the binomial theorem can be applied to the right side of (37) giving:

$$\alpha = (\alpha-\alpha\pi_p + \pi_p)^c. \tag{38}$$

The latter is a c^{th} order polynomial in α which has one root $\alpha = 1$. For equilibrium to exist there must be a root α such that $0 < \alpha < 1$.

The relationship between c and π_p such that there is a root α, $0 < \alpha < 1$, can by established by considering Figure 5. Here $f_1(\alpha) = \alpha$ and $f_2(\alpha) = (\alpha-\alpha\pi_p+\pi_p)^c$. are plotted against α. The plot of f_2 can take either the form of the lower broken line (where there is a root α, $0 < \alpha < 1$) or of the upper broken line (where there is no root α, $0 < \alpha < 1$). For the plot of f_2 to have the form of the lower broken line clearly $\frac{df_2(1)}{d\alpha} > 1$. Since:

$$\frac{df_2}{d\alpha} = c(\alpha-\alpha\pi_p+\pi_p)^{c-1}(1-\pi_p) \tag{39}$$

this requirement becomes:

$$c(1-\pi_p) > 1 \tag{40}$$

or:

$$\frac{1}{c} + \pi_p < 1. \tag{41}$$

Informally, the latter relation states that the sum of the fraction of MM cycles used by the CPU and the fraction of MM cyles used by the IOP cannot exceed one.

Given that π_p and c are such that (41) holds, A in (35) can be found from the usual requirement:

$$1 = \sum_{k=1}^{\infty} p(k)$$

$$= A \sum_{k=1}^{\infty} \alpha^k \tag{42}$$

$$= A \frac{\alpha}{1-\alpha}$$

Hence:

$$A = \frac{1-\alpha}{\alpha} \tag{43}$$

and thus:

$$p(k) = (1-\alpha)\alpha^{k-1}. \tag{44}$$

From this E[n] can be obtained:

$$E[n] = \sum_{k=1}^{\infty} kp(k)$$

$$= (1-\alpha) \sum_{k=1}^{\infty} k\alpha^{k-1}$$

$$= \frac{1}{1-\alpha}. \tag{45}$$

It can be seen from this relation that as $\alpha \to 0$ (reflecting the no I/O case), $E[n] \to 1$, and as $\alpha \to 1$ (reflecting the complete utilization of all the MM cycles), $E[n] \to \infty$.

Variable Priority - regular I/O: Unlike the random I/O case, there does not appear to be a convenient general solution for variable priority for regular I/O. In order to indicate the type of analysis required, a discussion for the case of the single IOP queue position follows.

Referring to Figure 3, consider the c cycle sequence of MM cycles m through m+c-1. For MM cycles m through m+c-2 the CPU has priority. If the IOP request (which arrived at the beginning of cycle m) has not received service by the end of cycle m+c-2, then the IOP request has priority for cycle m+c-1. At the beginning of an MM cycle which receives an IOP request the system can be in one of two states. The state S(0)

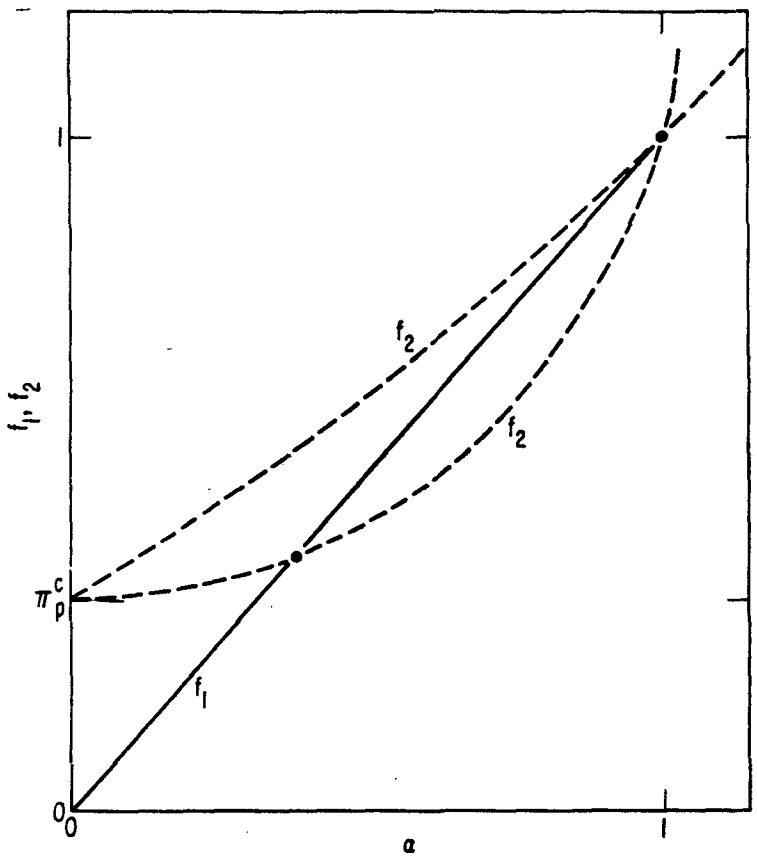

FIG. 5 GRAPH TO DETERMINE RELATION BETWEEN c AND π_p

is characterized by the CPU being able to request that MM cycle. The state S(1) is characterized by the CPU being queued as a result of its having requested the previous MM cycle but not getting it as a result of the IOP exercising priority. When the system is in state S(1) the CPU necessarily gets the current MM cycle.

It is assumed that the system is in equilibrium and there exist equilibrium probabilities p(Ø) and p(1) for the states S(Ø) and S(1). Necessarily:

$$p(1) = 1-p(0). \qquad (46)$$

Now let us consider the probability $p_{ØØ}$ that the system is in S(Ø) just after the arrival of the current IOP request given that it was in S(Ø) just after the arrival of the last IOP request. For this to occur there must have been at least one of the c MM cycles m through m+c-1 free of an IOP request. Hence

$$p_{00} = 1-\pi_p^c . \qquad (47)$$

If the system was in S(1) just after the arrival of the last IOP request and in S(Ø) just after the current IOP request, then cycle m was used to service the CPU and at least one of the c-1 MM cycles m+1 through m+c-1 must have been free of a CPU request. Thus:

$$p_{10} = 1-\pi_p^{c-1} \qquad (48)$$

The equation relating the equilibrium probabilities is:

$$p(0) = p_{00}p(0) + p_{10}p(1). \qquad (49)$$

Substituting (46), (47), and (48) in (49) gives:

$$p(0) - (1-\pi_p^c)\,p(0) + (1-\pi_p^{c-1})(1-p(0)) \qquad (50)$$

or:

$$p(0) = \frac{1-\pi_p^{c-1}}{1-\pi_p^{c-1}+\pi_p^{c}} \quad . \qquad (51)$$

Then from (46):

$$p(1) = \frac{\pi_p^{c}}{1-\pi_p^{c-1}+\pi_p^{c}} \quad . \qquad (52)$$

Each time the system enters S(1) the CPU is blocked for one cycle out of a c cycle sequence. Since the average rate at which the unblocked CPU executes is π_p/t_m, R_p for this case is:

$$R_p = \frac{\pi_p}{t_m}\left(1-p(1)\frac{1}{c}\right)$$

$$= \frac{\pi_p}{t_m}\left(1 - \frac{\pi_p}{c}\,\frac{\pi_p^{c-1}}{1-\pi_p^{c-1}+\pi_p^{c}}\right) \quad . \qquad (53)$$

This expression should be compared with (30) for the IOP priority case. It is easy to show that the factor $\pi_p^{c-1}/(1-\pi_p^{c-1}+\pi_p^{c})$ is less than one so R_p is higher for the variable priority case than for the IOP priority case. Because $c > 2$ and $0 < \pi_p < 1$, the following are true:

$$(1-\pi_p)(1-\pi_p^{c-1}) > 0 \qquad (54)$$

and

$$\pi_p^{c-1}-\pi_p < 0. \qquad (55)$$

Thus:

$$(1-\pi_p)(1-\pi_p^{c-1}) > \pi_p^{c-1}-\pi_p \qquad (56)$$

or:

$$1-\pi_p^{c-1}+\pi_p^{c} > \pi_p^{c-1} \qquad (57)$$

and so:

$$\frac{p^{c-1}}{1-\pi_p^{c-1}+\pi_p^{c}} \qquad (58)$$

The number of IOP requests queued just after the arrival of an IOP request is clearly one for this case:

$$E[n] = 1. \qquad (59)$$

Summary of Results

All of the formulas derived are summarized in Table 1.

Examples and Discussion

The expressions for R_p and E[n] derived previously have been evaluated for certain sets of values for π_p, π_0, and ℓ. The purpose of this is to give some insight into the CPU performance improvements that can be realized by using other than an IOP priority discipline.

The results of the evaluations are displayed graphically in Figures 6 through 12. The graphs take the form of a sumultaneous plot of the CPU rate R_p (expressed as a fraction of $1/t_m$) and E[n] against the I/O rate (expressed as a fraction of $1/t_m$ and thus equal to π_0). The parameter π_p is not shown explicitly but rather t_p since it is a more readily interpreted measure of basic CPU speed (π_p is given in terms of t_p by equation (2)). Furthermore t_p is given a normalized form being expressed in terms of t_m.

Figures 6, 7, and 8 are for random I/O with t_p equal to 0.25 t_m, 1.5 t_m and 9 t_m respectively. For the variable priority ℓ, the number of IOP buffer positions, is equal to three. For the CPU priority discipline, the CPU rate is of course constant out to the point of the maximum permissable I/O rate. At very low and very high I/O rates the IOP and the variable priority disciplines yield comparable CPU rates, but at intermediate I/O rates the variable priority discipline leads to noticeably better CPU performance. For the CPU priority discipline E[n] increases very rapidly as the point of maximum permissable I/O rate is approached while for the variable priority discipline E[n] increases to the IOP queue size as the I/O rate goes to one. Figure 9 is for random I/O with $t_p = 1.5\ t_m$. The CPU rate is plotted against the I/O rate for different values of (E[n] is not plotted). A rather significant improvement is noted in going from the IOP priority discipline to the variable priority discipline with $\ell=3$. A lesser improvement is noted in going from $\ell=3$ to $\ell=8$. The effect of larger ℓ is to hold the CPU rate up to its maximum value for higher I/O rates but once a certain I/O rate but once a certain I/O rate is reached (about 0.6 in this case) the CPU rate falls off abruptly. An encouraging finding here is that only a moderate number of IOP queue positions are needed to realize most of the performance improvement attainable by the variable priority discipline.

Figures 10, 11, and 12 are for regular I/O again with t_p equal to 0.25 t_m, 1.5 t_m, and 9 t_m respectively. Although the plots are drawn as smooth curves, they are of course meaningful only at those I/O rates marked with c=10 through c=2. In comparison with the plots for random I/O, it can be seen that for a given t_p and I/O rate the CPU rate for the IOP priority discipline is higher for regular I/O than for random. This is a well known queuing theory result and is most noticeable for t_p equal to 1.5 t_m and 9 t_m. Because of the way it is defined, E[n] for regular I/O is not directly comparable with E[n] for random I/O. The variable priority discpline even with single IOP queue position is noticeably better than the IOP discipline particularly for t_p equal to 0.25 t_m and 1.5 t_m.

The applicability to real systems of the analysis and results of this discussion of course depends on how well the basic model reflects the operation of those systems. An increasingly prevalent feature in computer systems is the so called "cache" memory [8]. From the standpoint of the current discussion the importance of the cache is that it is often a single memory with a cycle and access time equal to the processor cycle time and having a synchronous interface with the processor. If the IOP also interfaces with the cache, the operation of such a system is well represented by the model. The analysis presented here together with some extensions has been used rather successfully to predict the performance of computers with cache memories.

		π_p	
IOP	Regular	$\frac{\pi_p}{t_m}\left(1-\frac{\pi_p}{c}\right)$	1
IOP	Random	$\frac{\pi_p}{t_m}\left(1-\frac{\pi_o\pi_p}{1+\pi_o\pi_p-\pi_o}\right)$	π_o
CPU	Regular	$\frac{\pi_p}{t_m}$	$\frac{1}{1-\alpha}$ where α is solution of $\alpha = (\alpha-\alpha\pi_p+\pi_p)^c \quad 0 < \alpha < 1$
CPU	Random	$\frac{\pi_p}{t_m}$	$\frac{\alpha}{1-\alpha}\cdot\frac{1}{\alpha+\pi_p(1-\alpha)}$ where $\alpha = \frac{\pi_p\pi_o}{(1-\pi_p)(1-\pi_o)} \quad 0 < \alpha < 1$
Variable (One level)	Regular	$\frac{\pi_p}{t_m}\left(1-\frac{\pi_p}{c}\cdot\frac{\pi_p^{c-1}}{1-\pi_p^{c-1}+\pi_p^c}\right)$	1
Variable (ℓ level)	Random	$\frac{\pi_p}{t_m}\left(1-\frac{(1-\pi_p)}{(1-\alpha^\ell)}\cdot\frac{1-\alpha}{\pi_p(1-\alpha)+\alpha}\cdot\alpha^\ell\right)$	$\frac{\alpha}{1-\alpha}\cdot\frac{1}{\alpha+\pi_p(1-\alpha)}-\frac{\ell\alpha^\ell}{1-\alpha^\ell}$ where $\alpha = \frac{\pi_p\pi_o}{(1-\pi_p)(1-\pi_o)} \quad 0 < \alpha < 1$

Table 1. Summary of Results

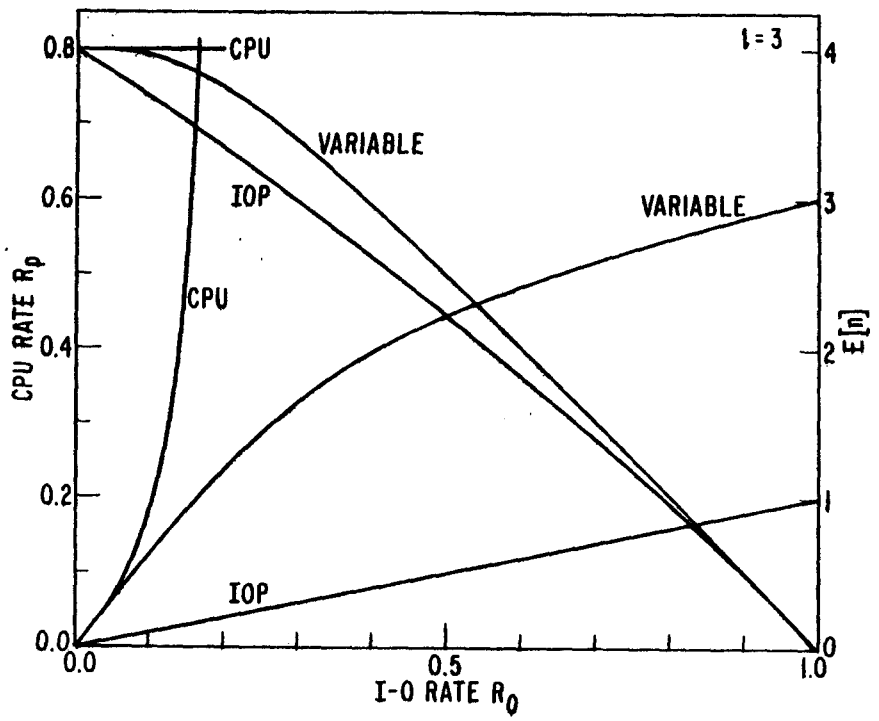

FIG. 6 RESULTS FOR $t_p = 0.25\ t_m$ (RANDOM)

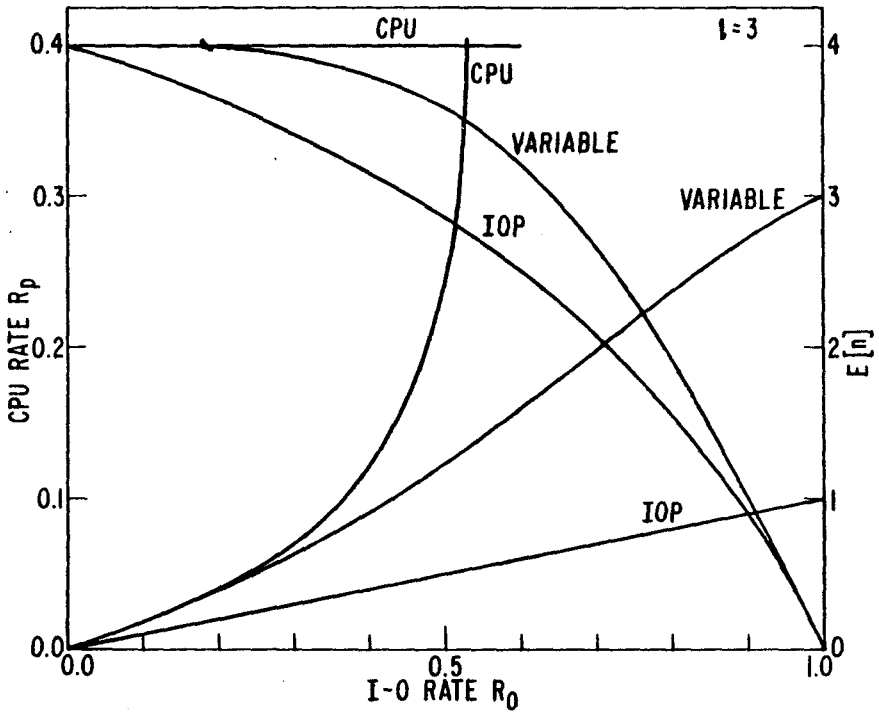

FIG. 7 RESULTS FOR $t_p = 1.5\ t_m$ (RANDOM)

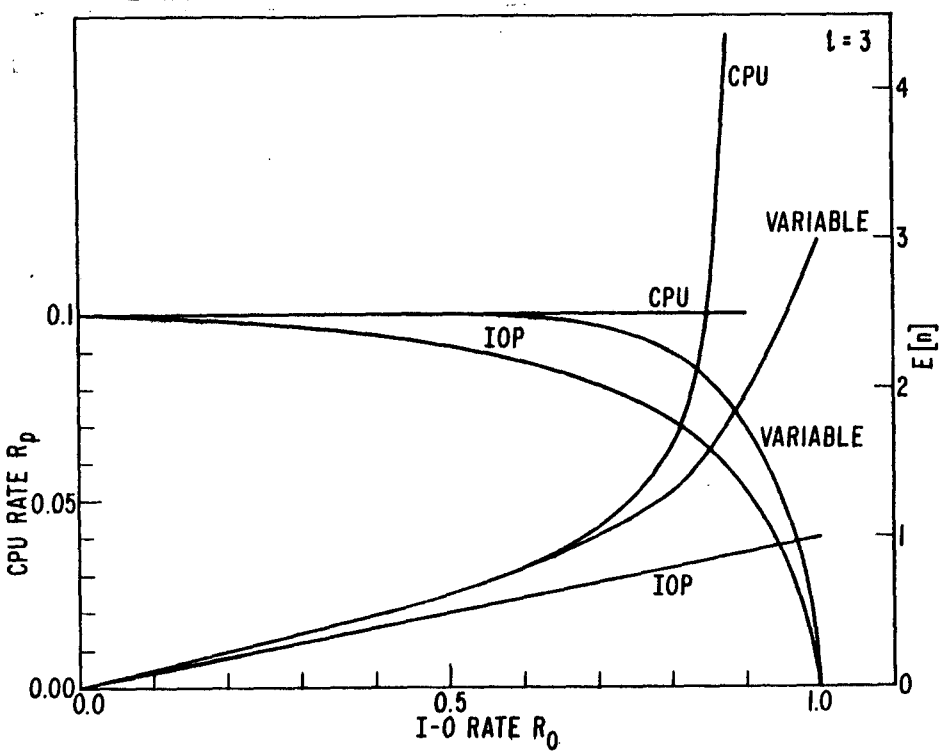

FIG. 8 RESULTS FOR $t_p = 9\,t_m$ (RANDOM)

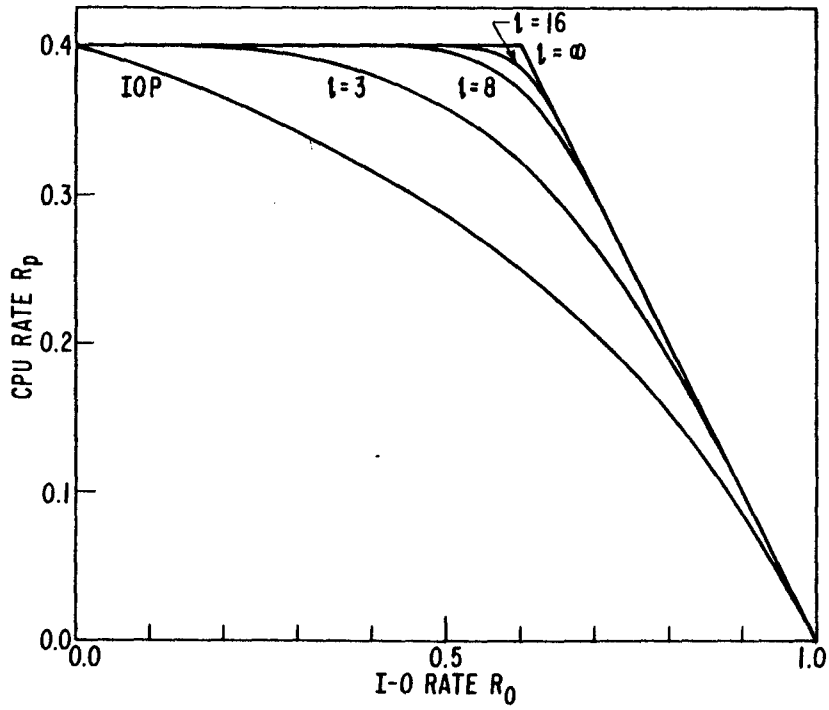

FIG. 9 RESULTS FOR $t_p = 1.5\,t_m$ (RANDOM) WITH VARYING IOP QUEUE SIZES

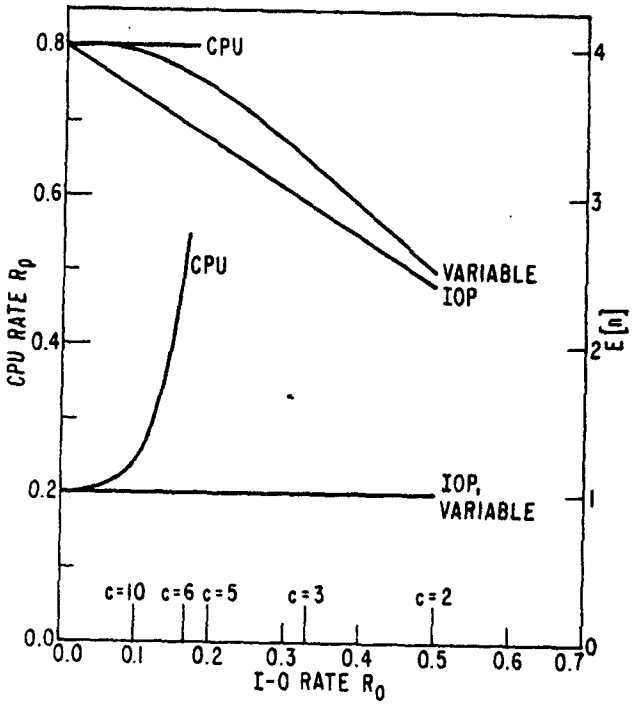

FIG. 10 RESULTS FOR $t_p = 0.25\ t_m$ (REGULAR)

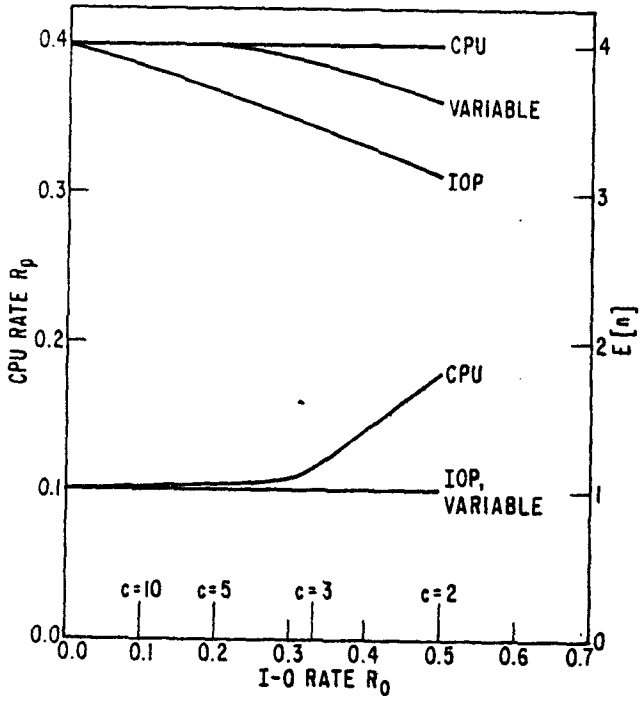

FIG. 11 RESULTS FOR $t_p = 1.5\ t_m$ (REGULAR)

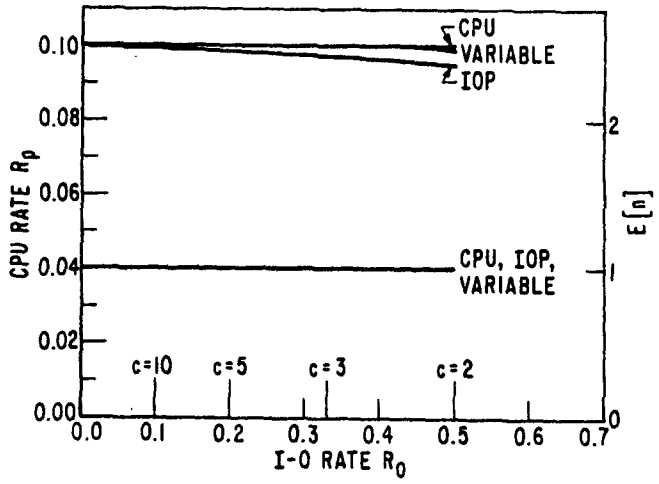

FIG. 12 RESULTS FOR $t_p = 9\ t_m$ (REGULAR)

References

[1] J. Shemer and S. Gupta, "A Simplified Analysis of Processor 'Look Ahead' and Simultaneous Operation of a Multimodule Main Memory", IEEE Trans. E.C., Vol. c-18, Jan. 1969, pp. 64-71.

[2] C. Skinner and J. Asher, "Effect of Storage Contention on System Performance", IBM Systems Journal, vol. 8, No. 4, 1969, pp. 319-333.

[3] W. Strecker, "An Analysis of the Instruction Execution Rate in Certain Computer Structures", Ph.D. thesis, Carnegie-Mellon University, 1970, ch. 6.

[4] M. Pirtle, "Interconnection of Processors and Memory", AFIPS Proc. FJCC 1967, Thompson Book Co., Washington, pp. 621-633.

[5] J. Little, "A Proof of the Queueing Formula: L= W", Opns, Res., vol. 9, Mar. 1961, pp. 383-387.

[6] D. Cox, and W. Smith, Queues, Methuen and Co., London, 1961.

[7] S. Goldberg, Introduction to Difference Equations, Wiley, New York, 1958.

[8] C. Conti, "Concepts for Buffer Storage", IEEE Comp. Group News, vol. 2, Mar. 1969, pp. 9-13.