COMMENTS ON MODELS OF MULTI-PROCESSOR MULTI-MEMORY BANK COMPUTER SYSTEMS

W. R. Franta and Philip A. Houle

University of Minnesota

ABSTRACT

This paper is concerned with processor degradation produced by access conflicts in multi-processor, multi-memory bank computer systems. Hardware and software parameters which influence performance are outlined and discussed. The incorporation of these parameters in analytic and simulation models is discussed with a view to the predictive merit of the model. Finally a general simulation model of a multi-processor is presented, and results based on its use are analyzed and compared with those of other models.

INTRODUCTION

In this paper we discuss various topics concerning the development of models (analytic and simulation) of multi-processor multi-memory bank computer systems. In all instances we are interested in the degradation in processor performance as a result of memory bank contention. To that end we are interested in various bank configurations including interleaved (partial and complete), separate banks for processor programs and shared data across a collection of banks, shared program banks and separate data banks, etc. The above considerations affect the manner in which references are generated by the processors in the configuration. Since the queuing discipline employed by the banks also affects system (processor) performance we will be interested in studying the effects of several. For example, the first come first served discipline as assumed by the queuing model of the next section (as well as by several other models) is examined. Since many systems employ priority schemes (see, for example, [7]), especially when input-output controllers are included, we shall also be interested in strict priority as well as round-robin service disciplines as memory port disciplines for the banks. Many systems also provide separate access ports for data and instruction references and on contention give priority to the data reference access port. Finally the distribution of processor inter-request times is important, especially on machines for which a variable number of instructions may be held in a machine word.

As a point of departure we next briefly review the literature. Of particular interest are the models discussed in references [5]-[9]. The study in [5] considers the contention in a system consisting of a processor and input-output controller, to be a function of the number of instruction "look-aheads". Employing the degree of "look-ahead" as a parameter, the expected value of processor waiting time due to input-output controller memory requests is derived. The model further assumes that instruction fetches are completely sequential. The model in [7] and [8] calculate the expected memory bandwidth (defined as the average number of requests serviced per memory cycle by a collection of interleaved memory banks). Instruction and data requests are modeled separately and overall bandwidth is calculated as the average of the two individual bandwidths. No interrelationship between data and instruction references is assumed. The implication of this assumption is that the instruction requests can race ahead of the data requests, resulting in an overly optimistic model. This latter deficiency is considered in the model reported in [6].

Unlike the previous models, that of [6] is based upon the theory of Markov chains and in some sense emends the deficiency previously mentioned. More specifically the following assumptions are made

- Each memory bank operates continuously, and cyclically.
- The operation of all memory banks is synchronized.
- No distinction is made between instruction and date references.
- Each processor makes only one memory request per synchronized memory cycle. This implies that under optimal conditions of no conflict two memory cycles are required to fetch an instruction and its associated operand.
- The request pattern of the processors is a sequence of Bernoulli trials. This implies that instructions are not executed sequentially or alternately that interleaving is not modeled.
- If a processor fails to access a bank on a given cycle due to contention, it automatically returns on the ensuing cycle (hence the use of Markov chains).
- Request contentions are decided on a probabilistic basis. By appropriately setting the associated probabilities a partial priority discipline can be modeled.

The steady state probabilities generated allow a processor degradation factor to be computed. Since statistically non-homogeneous processors are

modeled, establishing the transition matrix probabilities is extremely difficult for systems consisting of as few as four processors and two memory modules. The model is not, therefore, amenable to the manipulation of its parameters.

Many of these deficiencies are overcome in the model reported in [9]. The price to be payed, is that the set of processors modeled must be considered statistically identical (except as noted below). This reduces the enormous number of states in the transition matrix of the model of [6] to a manageable number. Several memory bank configurations are studied including interleaved, separate program banks and interleaved shared data banks. Among the assumptions made (some for analytic tractibility) the following are pertinent.

- Overlap is modeled. That is requests for current operand and next instruction are issued simultaneously.
- The memory banks operate cyclically and synchronously as indicated above for the model of [6]. This essentially implies that all instructions require one cycle for execution.
- Bank contentions are resolved by a random selection. This implies that if a processor has a single outstanding request, it is with equal probability an instruction or data reference.
- Each instruction requires one machine word.
- Data references are independent and are made from the allowable banks on an equi-probable basis.
- An instruction requires an operand with a probability b, (Bernoulli trials).
- Program jumps are made with a probability α, (Bernoulli trials). The next bank is selected on an equi-probable basis from among the allowable banks.

For the complement of models reviewed that of [9] most faithfully reproduces the general characteristics of existing systems. We shall comment more on this in a later section.

The models discussed above all view the set of memory banks and processors as a unified system. In some instances it is convenient to consider the delay encountered by a single processor at a given bank. Then given information on the number of references made by that processor, the expected program execution time can be calculated. This approach views each memory request as a request for service from a single server with constant service time. A queuing phenomenon results owing to the generation of similar requests by the remaining processors. Since for many actual systems it is possible for several requests (at least two) to arrive simultaneously, queuing models with batch arrivals are appropriate. We can then consider that during a memory cycle a batch of requests arrives at the memory bank and is allowed to enter a "service buffer" associated with the memory bank at the end (beginning) of each memory cycle. For simplicity the queuing discipline of the buffer may be taken as first-come-first-served. Let E(w) denote the expected wait time per request.

In [6] an upper bound on E(w) is given for the GI/G/1 queue as

$$E[w] \leq \frac{\lambda(\sigma_a^2 + \sigma_g^2)}{2(1-\rho)}$$

where λ = arrival rate, σ_a^2 = variance of inter-arrival time distribution, σ_q^2 = variance of service time distribution. For our purposes $\sigma_a^2 = 0$ and σ_g^2 is calculated as follows. Let k be the batch size, and t the cycle time. Then the expected batch service time is E[k]t, and $\sigma_g^2 = \sigma_k^2 t^2$ and $\lambda = 1/E(k)$. In [1] the Laplace-Stieljes transform of the waiting time distribution is computed for a batch input general service time model, for which the batch inter-arrival times are assumed to follow the exponential distribution. The transform is given in terms of the transform of the service time distribution and the generating function of batch sizes. Differentiation of the expression and repeated use of L'Hopital's rule produces E(w) in terms of λ, E(k), $E(k^2)$, etc. Other pertinent remarks on batch queues may be found in [3,11,12]. For our purposes, a more convenient and somewhat simpler model can be developed, (which does not require Poisson input assumptions) by considering that batches of size j arrive at the memory bank with probability C_j, j=0,1,...,n during each memory cycle. An analysis of such a system is presented next.

QUEUING MODEL

Consider a memory bank with cycle time t. During each service period, batches of requests arrive from the remaining processors. The batch of requests is allowed to join the memory bank queue at the end of each memory cycle. Let C_j equal the probability that j requests arrive during a service period, with j=0, 1,..., ℓ, $\ell \leq 2p$, the number of processors. The situation is described by the model depicted in figure 0.

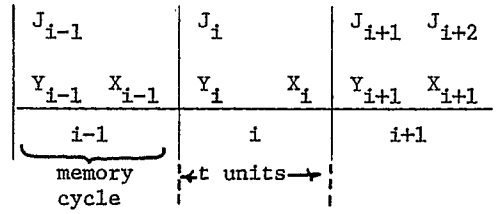

Figure 0.

In the figure

- J_i = The number of requests generated during the (i-1)st service period.
- Y_i = Number of requests in the system (in service and pending) at the beginning of the i-th service period.
- X_i = Number of requests in the system following completion of the i-th service period but preceding acknowledgement of the arrivals J_i.

We now describe the system as a queuing model. In statistical equilibrium, the distributions associated with the designated random variables are independent of the memory cycle period.

Consider

$$G_j(Z) = \sum_{j=0} C_j Z^j$$

as given. Let $P_n = P_r\{Y=n\}$ be the steady state probabilities for Y, and t equal the memory cycle time. The random variables Y form a Markov chain, and the values can be related across memory cycle periods by

$$Y_{i+1} = \max [(Y_i - 1), 0] + J_{i+1} \quad (1)$$

or as $Y_{i+1} = r_i + J_{i+1}$ with $r_i = \max [Y_i - 1), 0]$. Let $G_j(Z)$ denote a generating function, $E[\cdot]$ an expected value, and $V(\cdot)$ a variance. In statistical equilibrium eqn(1) implies

$$G_Y(A) = G_r(Z) \cdot G_j(Z) \quad (2)$$

It is easily seen that

$$G_r(Z) = \frac{1}{Z} G_Y(Z) - \frac{P_o}{Z} + P_o \quad (3)$$

so that substitution of (3) into (2) yields

$$G_Y(Z) = [\frac{1}{Z}(G_Y(Z) - P_o) + P_o] G_j(Z)$$

solving for $G_Y(Z)$ yields

$$G_Y(Z) = \frac{P_o(Z-1)}{Z - G_j(Z)} G_j(Z) \quad (4)$$

Since it is required that $G_Y(1)=1$, a single application of L'Hopital's rule shows that

$$P_o = 1 - E(j)$$

which further indicates that $0<E(J)<1$ is a necessary and sufficient condition for statistical stability. For reasons analogous to those given for (1) and (2) it is seen that

$$Y_i = X_{i-1} + J_i$$

so that $G_Y(Z) = G_X(Z) G_j(Z)$ or by (4) that

$$G_X(Z) = \frac{P_o(Z-1)}{Z - G_j(Z)}$$

with $P_o = 1 - E[j]$. Since $E[X] = G_X(1)$ a single differentiation and two applications of L'Hopital's rule yields

$$E[X] = \frac{E(J^2) - E(J)}{2[1-E(J)]}$$

and since $V(J) = E(J^2) - E^2(J)$

$$E(X) = \frac{V(J) + E^2(J) - E(J)}{2(1-E(J))} \quad (5)$$

Let w_n* denote the delay experienced by the nth request for service. During the period w_n+1 we expect $E(k)(w_n+1)$ requests to be generated. Following service of the nth request we expect $E(Y) = E(x) + E(k)$ requests to be in the system. Therefore in the mean

$$E(k)[E(w)+1] = E(x) + E(k)$$

Solving for E(w) yields

$$E(w) = \frac{E(X)}{E(K)}$$

*w_n is measured in memory cycles.

which after algebraic manipulation, use of eqn(5), and multiplication by t yields

$$E(W) = 1/2 \left[\frac{V(k)}{E(k)} \frac{1}{1-E(k)} - 1\right] t \quad (6)$$

time units.

By considering the steady state equations of detailed balance the probabilities P_n can be calculated giving

$$P_1 = \frac{1-C_o}{C_o} P_o$$

and

$$P_n = \frac{1}{C_o} [P_{n-1} - C_{n-1}(P_o + P_1) - \sum_{J=1}^{n-2} C_j P_{n-j}]$$

for $n \geq 2$.

Since P_o is given above, the process is easily mechanized. It must be noted that although $C_j=0$ for $J > l$, the same is not necessarily true for P_j. If we let

$$E(Y) = \sum_n P_n N$$

and $C_j' = C_j \cdot J/E(J)$*

then it is also true that

$$E(W) = \left\{ \sum_{n=1} \sum_{J=1} P_n C_j' (n-1 + \frac{J-1}{2}) + P_o \sum_{J=1} C_j' \frac{(J-1)}{2} \right\} t$$

which reduces to

$$E(W) = \left\{E(Y) - 1 + P_o + 1/2 \left[\frac{V(J) + E^2(J) - 1}{E(J)}\right]\right\} t \quad (7)$$

Eqns (6) and (7) are numerically equivalent. The value of V(W) is easily computed using the values of P_n as, for example,

$$V(W) = \sum_{n=1} \sum_{j=1} \left\{(n-1 + \frac{J-1}{2} - E(W)\right\}^2 P_n C_j' + P_o \left\{\sum_{J=1} \frac{J-1}{2} - E(W)\right\}^2 C_j'$$

number processors	E(J) = 8	12	16
1	-14	.07	.06
2	.30	.11	.13
3	.33	.25	.18

EXPERIMENTAL DATA

To investigate the behavior of programs as regards bank references, data was collected on program behavior for several widely differing systems. Using a simulator designed for research purposes (see [2]) data was collected on the behavior of programs running on the CDC 6600. In particular

*E(J) is the arrival intensity and thus C_j' represents the intensity for batches of size J.

the behavior of an operational APL* processor, a heavily used FORTRAN* compiler and several application programs were investigated. In all cases complete address traces were collected as program execution was simulated.** The traces included all program code executed including input-output and file manipulation subroutines. In all cases the program behavior was in no way altered due to the simulation process. The trace information provided several statistics. First operation code use distributions were established. Such data is pertinent to the simulation of models providing for asynchronous program, memory module behavior. Secondly it provided for the construction of certain transition matrices relating the banks from which successive instruction words and operands are fetched. A summary of instruction use is given by Figure (4). Two types of transition matrices were constructed, one for data, one for instruction fetches. Each is a matrix of the form $T_K = t^K_{ij}$,i, j=1,...,16, K=D(Data), I(instruction), where t^K_{ij} equals the probability that if the last reference of type k came from bank i, that the next reference comes from bank j. A similar but less comprehensive study was made on operational code for the machine described in [4]. Each trace accounted for well over 500000 memory references.

Several points of information can be gleaned from the data. In summary:

- The probability of a jump ranged from .15-.35 with Fortran and APL near the upper mark.
- The assumption that the next reference (following a jump) can be selected at random from among all the banks (for interleaved banking) is statistically reasonable.
- No single model on data references faithfully reproduces program behavior. For example on MNF

$$t_{i,\,i+2 \pmod{16}} = \{.48,.36,.42,.40,.45,\ .39,.43,.43,.48,.40,.47,.44,.52,.32\}$$

with the remaining entries quite uniform in nature. This predominant behavior can be attributed to symbol table manipulation. It can be argued that the above pattern is obvious and to be expected. It was also expected that a similar pattern would emerge for APL, as it too does considerable table manipulation. In fact no dominant pattern was observed for APL, i.e., its data reference matrix most nearly indicated a uniformly distributed reference pattern. Many application programs demonstrated the same type of behavior. Several produced main diagonal dominant matrices, while several others produced uniform like matrices much as for APL. In several cases, the a priori prediction of behavior was in error. The collected data did allow certain parameters to be established. In particular we find that:

- The probability that an instruction requires an operand can be estimated on the basis of operation code utilization.
- The distribution of the number of instructions per word can be established.

THE SIMULATION MODEL

Let M_i for i=1,2,...m represent the m memory modules of the model. Addresses for memory references consist of integer values j such that $i \leq j \leq m$. Memory references are denoted R_{ik} for memory module M_i from the K^{th} PE. A set Q_i associated with M_i contains outstanding memory requests to be processed by M_i. After request R_{ik} is executed by M_i, the k^{th} PE is notified.

The addressing modes from the PEs to the M_i are determined by the transitions within individual PEs. Each PE generates memory requests based on its particular characteristics. For example, if instruction words are stored in consecutive addresses which are from interleaved memory modules, then the k^{th} PE would generate memory references R_{1k}, R_{2k}, $\ldots R_{m,k} R_1, \ldots$, etc.

The structure of the processing elements is based on the concept of an instruction cycle. Within the cycle, various transitions may occur which affect the result in memory requests. An individual PE has an associated program state s. This state is an integer value $1 \leq s \leq m$ and can be considered a memory module location $\overline{M}_s$ of the current instruction word.

The transitions within the instruction cycle of a PE are described by a function

$$F(T,s)$$

where s is a state number and T is a transition matrix. T is defined as

$$T = \{A_{i,j}\}$$

where

$$A_{ij} = \sum_{k=1}^{j} P_{ik}$$

and P_{ij} is the probability of a transition from state i to state k. For example, if PE_k is in state S_k, a jump instruction is given by

$$S_k := F(T_\alpha, S_k)$$

where T_α is the matrix representing memory transitions during jumps. If an instruction word is required, the memory request results for

$$M_{F(T_\gamma, S_k)} .$$

If an operand reference occurs, the resulting memory request is for

$$M_{F(T_\beta, S_k)} .$$

Note that T_γ and T_β represent the matrix of transitions for instructions and operands respectively.

With the definition of a transition matrix T as

*See reference [13] for a description of APL and reference [15] for a description of the FORTRAN.

**The applications programs were FORTRAN programs. They included an analysis of the program used to analyze the address traces collected in the other cases.

the controlling factor in memory requests produced by PEs, a completely general simulation program can be constructed. For example, a model with three interleaved memory models would employ

$$\begin{bmatrix} 0 & 1.0 & 1.0 \\ 0 & 0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix}$$

as the transition matrix for the retrieval of the next instruction word. If all instructions come from the same module with a probability of .9 then

$$\begin{bmatrix} .9 & .95 & 1.0 \\ .05 & .95 & 1.0 \\ .05 & .1 & 1.0 \end{bmatrix}$$

might be an appropriate transition matrix.

Because the function F(T,s) and the transition matrix T control the behavior of PEs towards the memory modules of the system, it is necessary that individual PEs have unique characteristics. In terms of a simulation program, it must be possible for differing PEs to have differing matrices T. The approach taken was to define groups of PEs denoted G_n. Each G_n contains N PEs all identical in behavior. This structure enables the simulation model to handle differing processor types, for example, one type of PE may be for compute while another might process I/O requests.

To describe the actions during the instruction cycle of a PE we need to define the following probabilities. Let α represent the probability of a jump instruction. If an instruction is not a jump, then it issues memory requests for an instruction word and operand word with probabilities γ and β respectively. The cycle is completed by an execution delay which overlaps the memory request processing and may or may not extend beyond. The execution delay is determined by a draw from an empirical distribution obtained from actual machine data.

Individual memory requests provide information linking them to the PE which generated them. In addition, each memory request has an associated attribute called its priority. The linkage to the generating PE enables the processing of the request to be synchronized with the PE. The priority balue is used to order the requests in the memory module queue to model priority bussing.

The memory module removes memory requests from the waiting queue and processes them using a unit of time called a memory cycle. The queue disciplines simulated include random selection and priority. The algorithm for random selection is obvious. The priority queue is characterized by an ordered set of memory requests $\{R_{s,kj}\}$. Associated with each $R_{s,kj}$ is a priority P_{kj}. The selection discipline chooses the earliest smallest P_{kj}. For the priority bussing studied here, operand memory references were set to $P_{kj} = 1$ and instruction memory references to $P_{kj} = 2$.

Figure 1 shows the flow of the simulation of the PE. The symbol S_o indicates an original state for the individual PE. The first decision determines if the command is a jump instruction. A jump instruction has no operand other than the address to which the jump is being made. In the case of the non-jump instruction, an operand memory β reference occurs with probability and an instruction memory reference occurs with probability γ. The delay E is a random draw from an empirical distribution supplied as data to the simulation.

Figure 2 shows a modified definition of the PE model. This model differs from that shown in Figure 1 in its description of multiple instructions in an individual instruction word. As is evident from Figure 2, the advance to each next instruction generates a memory reference. The interaction between commands and instruction words is controlled by an empirically supplied distribution of the number of commands per word. This distribution determines the value of IPW.

An additional difference between this model and that in Figure 1 is the interaction between operand memory references and the instruction execution time. An operand memory reference occurs with probability β. However, if an operand memory reference does occur, the retrieval of the operand from the appropriate bank constitutes the execution time of the instruction. This model difference implies that machines exhibiting many accumulators with simple memory loads and stores are well represented.

SIMULATION EXPERIMENTS AND RESULTS

This section discusses the simulation experiments and summarizes the results of these experiments. The simulation experiments were based on three machine descriptions and the two PE models described above. The results are displayed as a series of graphs and tables which we discuss below.

The different PE models were described earlier. In addition, three modes of instruction word memory referencing were used, two types of memory queue disciplines were used to explore bussing effects, and the memory modules were synchronized and nonsynchronized. The transition matrices used for jumps and operand references were established as uniformally random.

The three modes of memory referencing for instruction memory requests are uniformally random, individually banked, and interleaved. For uniformally random references, all modules are equally probable for the next instruction word reference. In the case where the instruction references are individually banked, all instruction memory references are to the same module until a jump occurs. Interleaved references go to each memory module in sequence.

Recall that the individual memory modules are modeled as servers of a queue. Each module operates concurrently with the other memory modules of the system. With random selection as the queue discipline, all memory requests have immediate and equal chance at the memory modules. The priority queue discipline insures that operand references are served ahead of instruction word references. In the case of synchronized memory modules, all modules initiate their service cycles together. If a memory request enters a queue during a memory cycle and that particular module is inactive, the request must wait for the start of the next synchronized cycle. If the memory

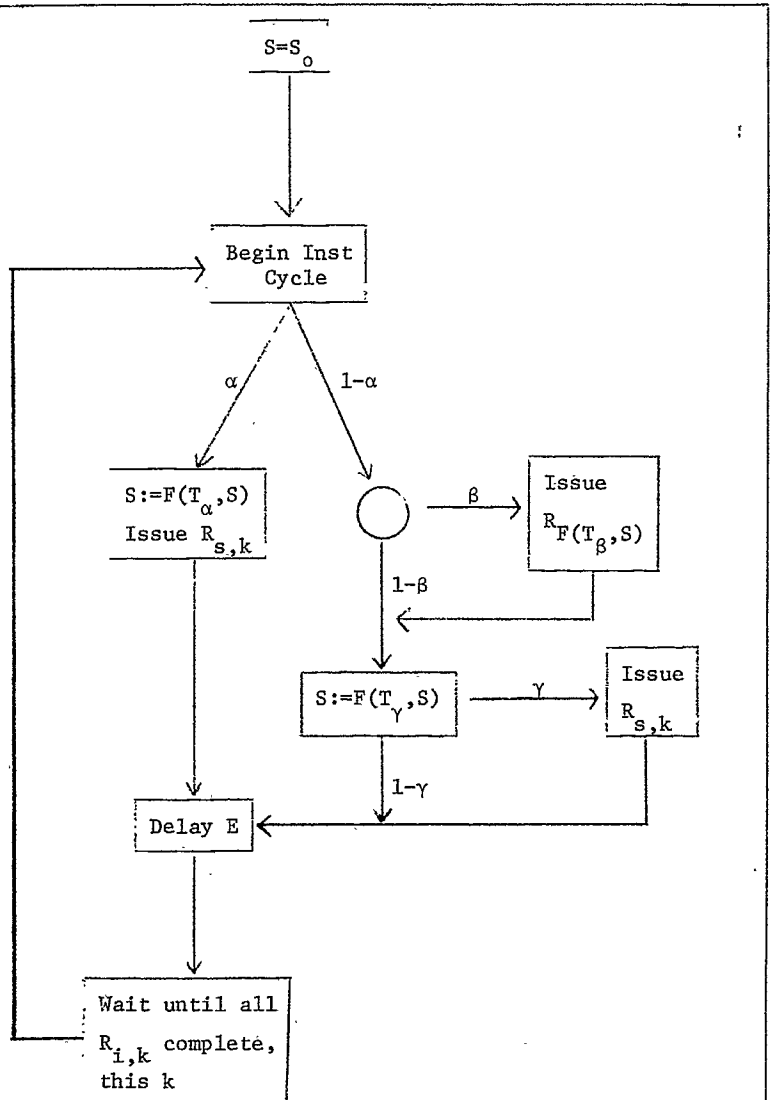

FIGURE 1. Flow of execution for an instruction cycle in the PE.

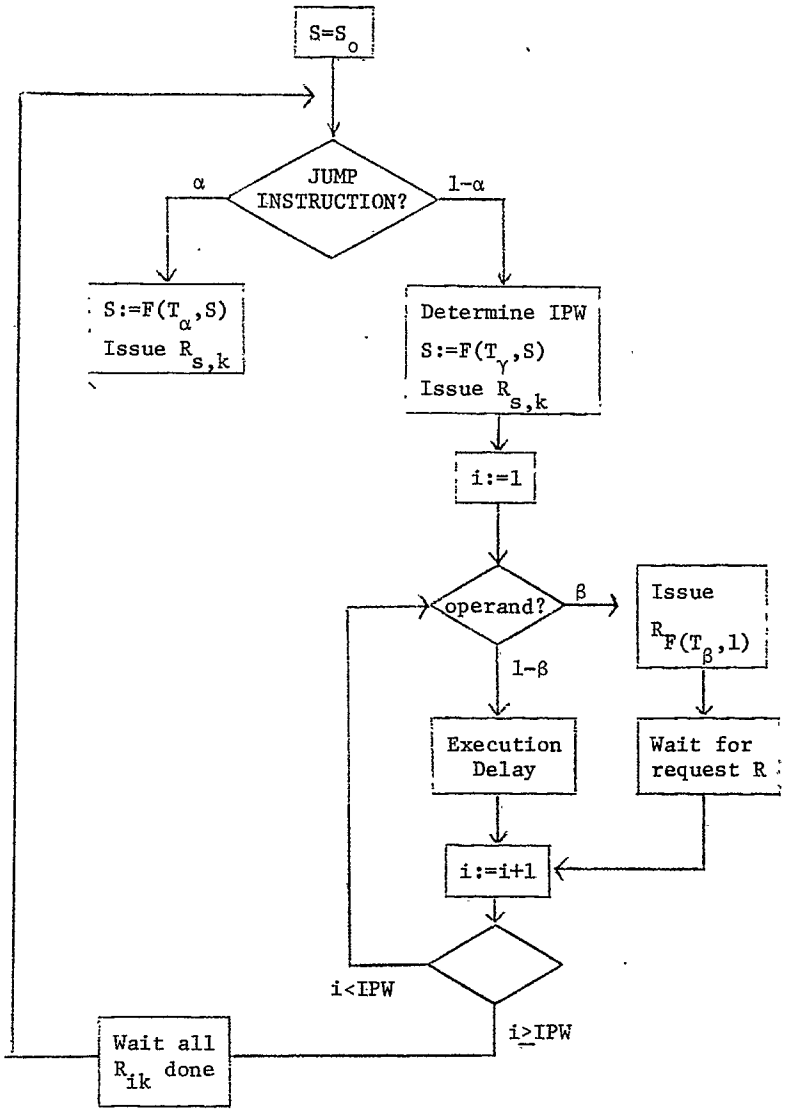

FIGURE 2. Flow of execution for an instruction word in the modified PE.

modules operate asynchronously, only a small (5%) delay is assumed. This assumption is arbitrary.

The figures and tables discussed below display the instruction execution rate (IER). This quantity is the number of instructions executed in the total configuration per memory cycle.

In Table 1 various IER values are presented for an idealized machine. All instructions require one memory cycle. These results compare very well with those obtained from the analytic study of Sastry [9]. Figure 3 shows the plot of the uniform case and some points from the work of Sastry are plotted for comparison. The favorable comparison indicates general validity of the behavior of the program.

As indicated in earlier sections, this study involves two real machines and their instruction mixes. Data describing a CDC 6600 program execution was used to generate the distribution detailed in Figure 4. Figure 5 shows the distribution of instruction mix for a single address computer, the UYK-7.

In Table 2 we summarize the results obtained using the UYK-7 instruction mix and memory cycle (1.5 microseconds) time. The values indicate the resulting IER. Several observations can be made from this table. First, the effect of uniform, banked, or interleaved instruction word references is minor compared to altering the number of memory modules or processors. Secondly, the change from uniform to priority bussing also has minor effects on the IER. Finally, Table 2 shows the effect of synchronizing the memory cycles of the memory modules. In Figure 6 we have applied the symbol "s" for the corresponding points with synchronized memory modules. Notice that these points remain close the the asynchronous points for a small number of memory modules. As the number of processors increases so does the point at which the synchronous and asynchronous points significantly differ. This can be explained in terms of the utilization of individual memory modules. As the utilization of a memory module decreases, the probability of a memory request finding the module idle increases. In the synchronous case, an idle memory module is characterized by a latency of a half a memory cycle. This delay causes a degradation in the response of the individual modules which, in turn, is displayed by the lower IER.

In Figures 6, 7, and 8 we have plotted the values for random bussing.

Finally, Figure 9 shows the results of the simulation using the CDC 6600 data and the modified PE model. This plot demonstrates the effect of allowing multiple instructions per instruction word. Rather than a probability of instruction word reference γ, the modified PE definition uses a distribution describing the number of instructions per word. Figure 9 indicates the values used. The increase in performance due to the multiple instruction words is significant.

CONCLUSION

Several of the results given here have been qualitatively known. In essence, this study has attempted to quantify these qualitative results. Our results relate to the mode of instruction referencing, to the type of bussing between processors and memory modules, to the synchronization of memory modules, to the number of commands per instruction reference, and to the relationship between probabilistic and simulation models. Specifically, for the loadings assumed

- Assuming no dedicated memory module assignments, the mode of instruction reference to the memory modules has a minor effect on the total configuration performance.
- Priority bussing for operand memory references caused no significant improvement over random selection. from all memory requests.
- The effect of synchronous memory modules is significant degradation in performance when individual modules exhibit lower utilizations. Configurations with a large number of processors will perform significantly better if individual memory modules can initiate memory cycles at the arrival of the request.
- Multiple commands per instruction word significantly increases performance, quantitatively reported in Figure 9.
- Markovian and simulation models produce results which faithfully predict actual system performance. Simple queuing models, such as the one reported, provide the machinery to develop simple first estimates of system performance. In some instances the quantification of the parameters may be easier for such models than for more elaborate models. When properly parameterized, the presented queuing model yielded results consistent with those of the model reported in [9].

REFERENCES

1. Saaty, Thomas L., Elements of Queuing Theory, McGraw-Hill, 1961.
(see especially pp. 40-42, Chapters 6 and 7, especially section 7-5.)
2. Olson, Gene H., Implementation of a Machine Simulator for the CDC 6000 Series Computers, M.S. Thesis, Dept. of Computer Sciences, Univ. of Minn., June 1973.
(Developed as a research tool, the program produces among other things, address strings for program executions. Each is appropriately tagged for later analysis.)
3. Chan, W.C., "Computer-Controlled Queuing System with Constant-Access Cycle and General Service Times", Proc. IEEE, 117 May 1970, pp. 927-930.
(This paper develops a queuing model for a Poisson input constant service time model with various ancillary considerations.)
4. UNIVAC, AN/UYK-y Military Computer Technical Description.
(Describes a general purpose 32 bit word, one

Memory Modules	3 Processors IER		5 Processors IER	
	Uniform	Interleaved	Uniform	Interleaved
2	1.01	1.12	1.10	1.16
4	1.58	1.61	1.86	1.99
6	1.87	1.91	2.33	2.54
8	1.98	2.10	NA	NA

TABLE 1. A comparison of uniformally random and interleaved instruction references for the machine with instruction times equal to one memory cycle.

	Memory Modules	Random Bussing Asynchronous Memory			Random Bussing Synchronous Memory			Priority Bussing Asynchronous Memory		
		Unit	Banked	Int.	Unit	Banked	Int.	Unit	Banked	Int.
	2	1.11	1.07	1.14	1.10	1.07	1.11	1.10	1.02	1.11
	4	1.49	1.58	1.56	1.44	1.53	1.52	1.49	1.65	1.58
P=3	6	1.60	1.80	1.74	1.41	1.55	1.47	1.59	1.83	1.73
	16	1.88	1.95	1.97	1.65	1.62	1.53	1.82	1.95	1.97
	32	2.00	2.00	1.97	1.67	1.65	1.83	1.94	2.00	1.97
	2	1.26	1.28	1.31	1.23	1.26	1.29	1.21	1.28	1.29
	4	2.16	2.31	2.24	2.18	2.27	2.25	2.21	2.33	2.13
P=8	6	2.80	3.03	3.09	2.76	2.94	3.05	2.93	3.05	2.94
	16	4.56	4.51	4.64	3.83	4.13	3.69	4.35	4.48	4.48
	32	5.04	5.07	5.23	4.35	4.61	4.51	5.09	5.13	5.26
	2	1.38	1.37	1.38	1.35	1.37	1.35	1.35	1.32	1.32
	4	2.27	2.43	2.34	2.25	2.39	2.30	2.36	2.51	2.30
P=12	6	3.06	3.35	3.33	3.03	3.32	3.29	3.09	3.41	3.18
	16	5.07	5.29	6.03	5.63	5.19	5.76	5.55	5.54	5.97
	32	7.18	6.87	7.21	6.30	6.04	5.90	7.09	6.86	7.17

TABLE 2. A comparison of bussing discipline, instruction reference discipline, and memory synchronization for differing numbers of processors and memory modules.

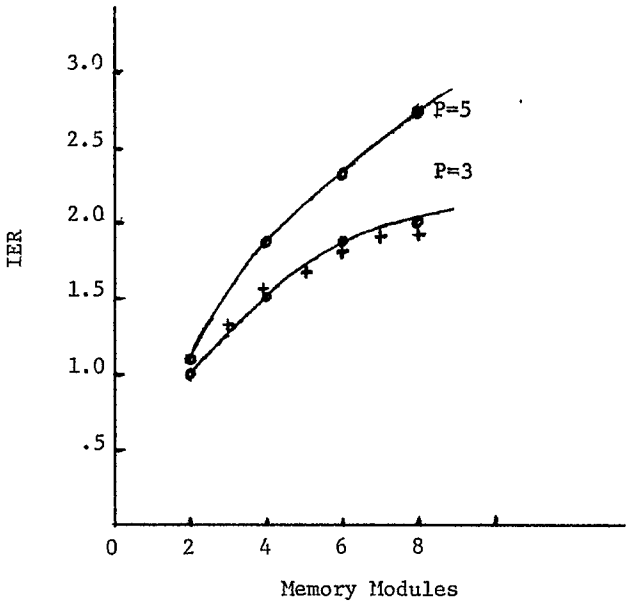

FIGURE 3. IER as a function of memory modules. The + indicates a theoretical point.

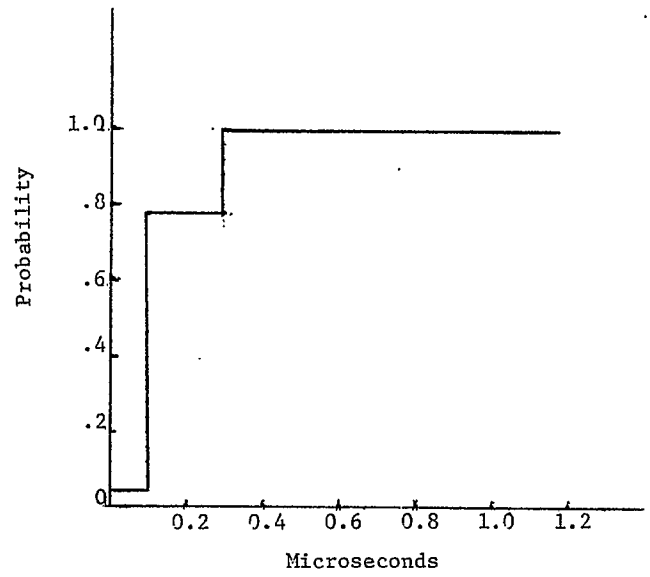

FIGURE 4. CDC 6600 instruction mix with α=.195 and β=.247.

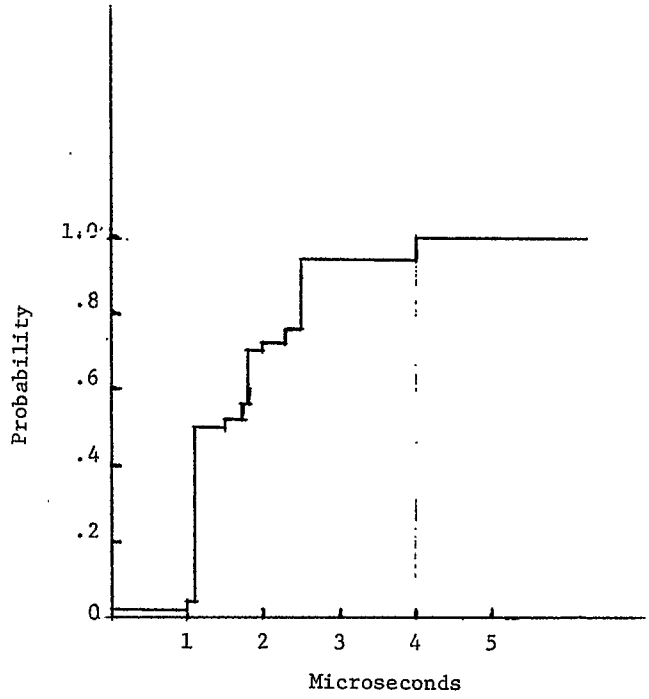

FIGURE 5. UYK-7 instruction mix with α=0.2, β=.833, and γ=.833.

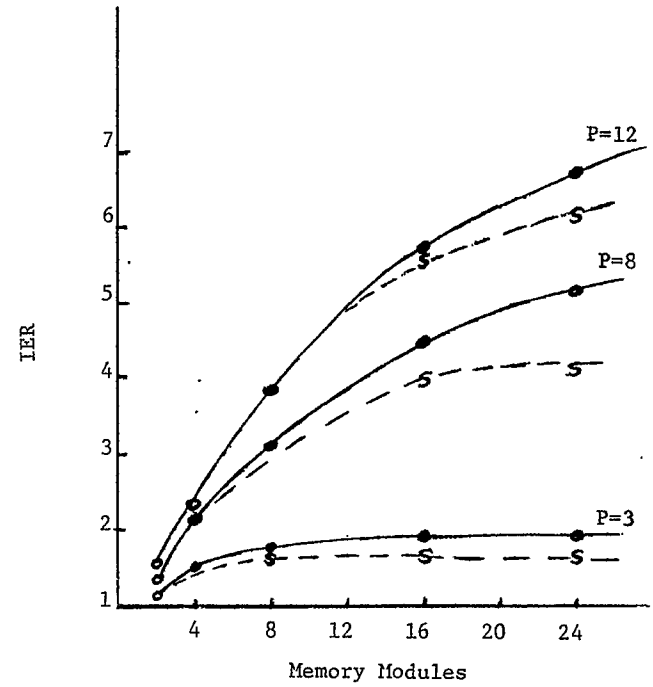

FIGURE 6. IER for UYK-7 loading with all references uniform. The dashed curves represent the IER for synchronized memory modules.

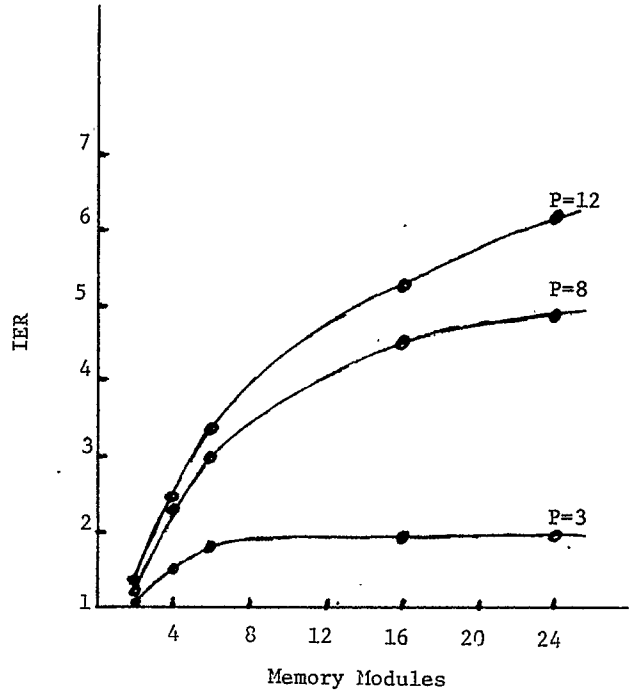

FIGURE 7. IER for UYK-7 loading with dedicated bank instruction references, and uniform operand and jump references.

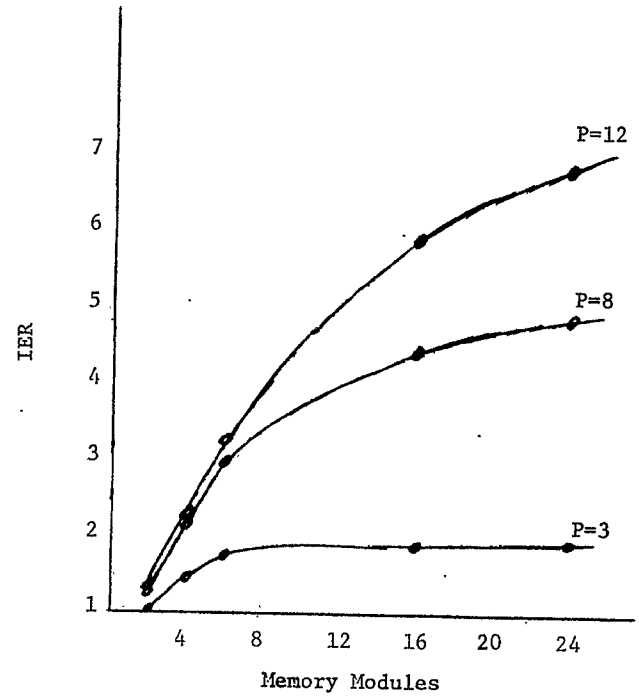

FIGURE 8. IER for UYK-7 loading with interleaved instruction references, and uniform operand and jump references.

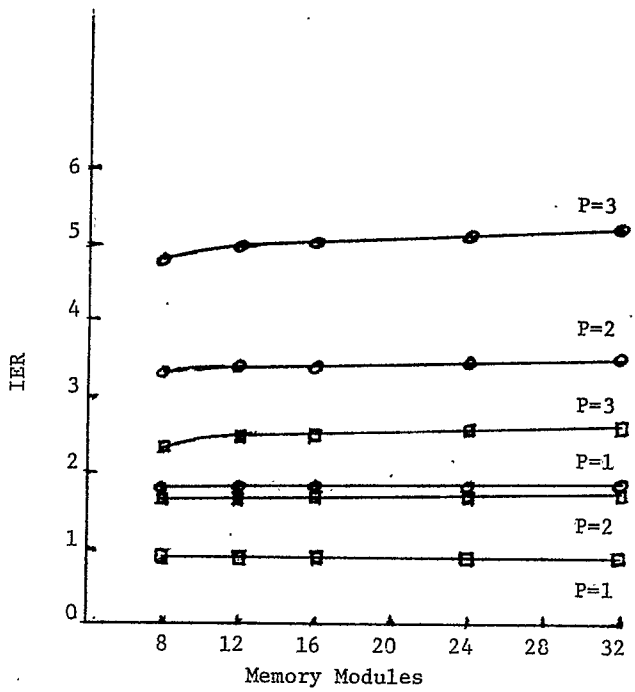

FIGURE 9. IER for CDC 6600 loading with interleaved instruction references and uniform operand and jump references. 0 indicates points using N instructions per word using P(N) of Table 3 while □ indicates points for single word instructions.

N	P(N)
1	0.0
2	0.076
3	0.447
4	0.477

TABLE 3. Probability of N instructions per word in CDC 6600 loading.

instruction per word machine.)
5. Shemer, J.E. and S.C. Gupta, "A Simplified Analysis of Processor 'Look-Ahead' and Simultaneous Operation of a Multimodule Main Memory," IEEE trans. on Computers, C-18, 1, Jan. 1969, pp. 64-71.
6. Skinner, C.E. and J.R. Asher, "Effects of Storage Contention on System Performance," IBM Systems Journal 8, 4, 1969, pp. 319-333.
7. Burnett,G.J. and E.C. Coffman, "A Study of Interleaved Memory Systems," Proc. AFIPS 1970 SJCC, Vol. 36, pp. 467-474.
8. Burnett, G.J. and E.G. Coffman, Jr., "A Combinatorial Problem Related to Interleaved Memory Systems," JACM, 20, 1, Jan. 1973, pp. 39-45.
9. Sastry, Kuchibhotla, "Markovian Models for Performance Evaluation of Multiprocessor, Multimemory Computer Systems," Ph.D. thesis, Univ. of Minn., June 1973.
(Presented at the Computer Science Conference, Columbus, Ohio, 20-21 February 1973.)
10. Marhsall, K.T., "Some Inequalities in Queuing," Oper. Res. 16, 1968, pp. 651-655.
11. Soriano, A., "On the Problem of Batch Arrivals and Its Application to a Scheduling System," Oper. Res. 13, 1965, pp. 398-407.
12. Foster, F.G., "Batched Queuing Processes," Oper. Res. 12, 1964, pp. 441-449.
13. Franta, W.R. and G.R. Mansfield, APL/KRONOS an Abridged Description, Tech. Report UCC No. 3, Univ. Computer Center, Oct. 1972.
14. Dahl, O., and Myhrhoug, Bjorn and Kristen Nygaard, Simula Common Base Language, Norwegian Computing Center, Oslo, 3, Norway.
15. ----, MNF Reference Manual, Univ. Computer Center, Univ. of Minn., 1971.