APPROXIMATIONS FOR SIMULATION RUN-TIME REDUCTION

G.M. Herman
Control Data Corporation
St. Paul, Minnesota

INTRODUCTION

The age of more complex computing systems has made it increasingly important to have reliable means of evaluating the performance of existing systems, and of future systems during their design and development. This latter aspect is only beginning to come into being, and shows great promise of yielding large benefits to the industry. Where most evaluation was, and still is, applied to the evaluation of existing systems for various applications, it is becoming increasingly recognized as a valuable tool for use during the design and development of systems. Results of these evaluations are used to aid in the decisions leading to inclusion of proper hardware and software, incorporated into an effective total system for the task for which it is designed.

In instances where academic know-how and system complexity and variations prevent adequate system evaluation or analysis by computational means, simulation may provide the means of analysis required. This analysis may, however, be expensive and time-consuming on the computer, depending upon the accuracy desired. For there exists within any simulation effort a tradeoff, or law of diminishing returns, if you will, between simulation run-time and simulation accuracy. The accuracy desired is, of course, dependent upon the application at hand.

Effective use of approximations whose accuracy are within a tolerance sufficient to yield results suitable to the application, and incorporation within the model of appropriate known computational methods can greatly reduce the run-time required to achieve a given degree of simulation accuracy. It is to these approximations and computational methods that this paper is directed.

SIMULATION MODEL

This paper discusses and evaluates approximations implemented in a simulation model to provide a 10-to-1 reduction in computational time without unduly sacrificing simulation accuracy. This model was originally constructed using the Univac Simulation Package (USP) 2.0 simulation language for the purpose of studying the organization and control of multi-memory structures (memory hierarchies), and later modeled for evaluation in GPSS for execution on the CDC 3600 systems. The model simulates the activities of a demand-paging multiprogramming computing system consisting of one processor, one direct access local storage bank, and one extended core storage (ECS) bank not directly accessible by the processor. All data transfers between system components (processor, local storage, and ECS) are assumed to be performed on a request/acknowledge basis across a one-word interface.

Programs for input to the simulator are structured as shown in Figure 1. Each is represented by its priority, its start time at which it becomes available to the system for execution, and a linear sequence of segments, each with an identification label, a base execution time, and those pages required to be resident in local storage prior to initiation of segment execution.

EXECUTION TIME

The base execution times employed in Figure 1 reflect the interaction of many parameters whose relationships are not considered in detail in this analysis, but are incorporated inherently within the parameters utilized. Examples would be Average Instruction Execution Time and Processor Sequence Overlap. These base execution times are segment execution times at a defined local storage cycle time, without access conflict for local storage due to ECS/Local transfers. Expressions describing the relationship between execution time and local storage cycle time may be approximated and incorporated into the model to represent performance during a series of runs, each with differing local storage cycle times, without respecification of this base.

Let us assume that the time between processor requests for a hypothetical local storage bank of infinite speed (zero cycle time) may be defined by the probability density function $g(t)$ shown in Figure 2. This function, which will be called the Ultimate Reference Distribution (URD), is essentially a measure of the processor's thruput capability, and defines the lower limit to segment execution time requiring R references.

The following relationship is assumed valid for $g(t)$:

$$\int_{-\infty}^{\infty} g(t)\,dt = 1 \qquad (1)$$

and the mean time between memory requests, t_o, is expressed by the equation:

$$G_o = \int_{-\infty}^{t_o} g(t)\,dt = \int_{t_o}^{\infty} g(t)\,dt = \frac{1}{2} \qquad (2)$$

Let us also assume that the local storage has a cycle time α. There is, then, a probability G_α, where:

$$G_\alpha = \int_{-\infty}^{\alpha} g(t)\,dt \qquad (3)$$

that the memory will be next requested prior to completion of its current cycle, causing an execution delay time equal to $\alpha - t_r$; where t_r is the time of request. The mean execution delay time occurring with probability G_α is $\alpha - t_n$; where t_n is the mean time of request given that $t_r \leq \alpha$, and is expressed by the relationship:

$$G_n = \int_{-\infty}^{t_n} g(t)\,dt = \int_{t_n}^{\alpha} g(t)\,dt = \frac{G_\alpha}{2} \qquad (4)$$

There is a probability $1-G_\alpha$ that the next storage reference will occur after the memory cycle time has completed -- producing a local storage idle time equal to $t_r - \alpha$; where t_r is again the time of request. The mean memory idle time occurring with probability $1-G_\alpha$ is $t_m - \alpha$; where t_m is the mean time of request given that $t_r \geq \alpha$, and is expressed by the relationship:

$$G_m = \int_{-\infty}^{t_m} g(t)\,dt = G_\alpha + \frac{1-G_\alpha}{2} = \frac{G_\alpha + 1}{2} \qquad (5)$$

It follows that,

$$t_o = G_\alpha t_n + (1-G_\alpha) t_m \qquad (6)$$

Task execution time (ET) as a function of local storage cycle time may then be expressed by the relationship:

$$ET = R\left[t_o + G_\alpha(\alpha - t_n)\right] \qquad (7)$$

$$= R\left[\alpha G_\alpha + t_m(1-G_\alpha)\right] \qquad (8)$$

where R is the number of requests required for a given segment. Execution times may then be approximated to the degree required by approximating the integral expressions used to derive G_α, t_n, and t_m. Figure 9 shows the relationship between time per reference and local storage cycle time for normal Ultimate Reference Distributions of means (t_o) of 5.0, 3.0, 2.05, and 1.0 usec.

A system of this nature does not exhibit independent memory request inter-arrival time distributions, and such distributions cannot adequately express the Execution Time/Cycle Time relationship. Note that the expression for execution time, equations (7) and (8), do not assume independent inter-arrival time of storage requests, but accounts for inter-arrival dependence and execution delay as a function of processor storage request distribution and storage bank utilization.

MEMORY ACCESS

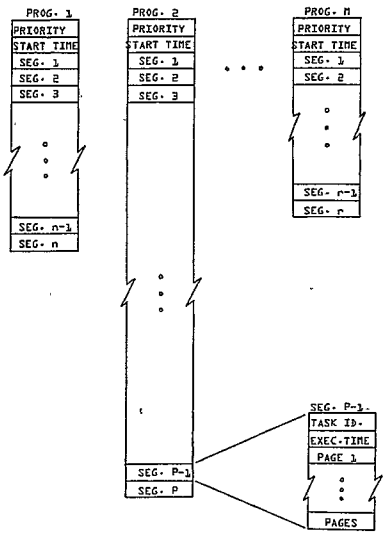
There are within the model two approximations introduced, inherently, by the method of local storage access employed to avoid instruction-level simulation. The first, task execution is delayed for the full duration of all local storage service times associated with ECS/Local transfers. And second, ECS service requests for local storage are granted instantaneous response regardless of the state of local storage at the time of interruption.

Granting immediate response to ECS service requests for local storage omits transfer delays that should occur. Although it is common practice to assign priority to transfer service requests over those for execution reference service, preemption of local storage while its cycle is in progress (preemptive resume priority discipline) does not accurately portray the physical

environment to be simulated. Transfer service requests to local memory occurring while a memory cycle is in progress should wait until the current cycle is complete before response is granted [non-preemptive priority discipline]. It is assumed in this analysis that these waits cause serial delays of block transfers not accounted for in the model.

Some estimation of the error introduced by these assumptions may be obtained by comparison of results derived using a preemptive priority discipline to those derived using a non-preemptive priority discipline. Figures 3 thru 7 show results obtained using an instruction-level simulation model constructed using GPSS for error analysis. Figures 3 and 4 show time per processor reference as a function of local storage and ECS cycle times for preemptive and non-preemptive priority disciplines respectively. Figure 5 shows the time per transfer as a function of local storage and ECS cycle times for preemptive and non-preemptive priority disciplines respectively. The URD was assumed to be normal with a standard deviation of $0.25\,t_o$. Figures 6 and 7 show the error incurred when the assumptions are employed as a function of local storage and ECS cycle times for processor references and transfers respectively. Whether such approximations and their associated errors are justified is dependent upon the application. It is noted that errors become increasingly larger as local storage and ECS cycle times approach equal values, and that errors are significantly less when differing appreciably.

Fig. 1



PROGRAM STRUCTURE

STATE MODEL

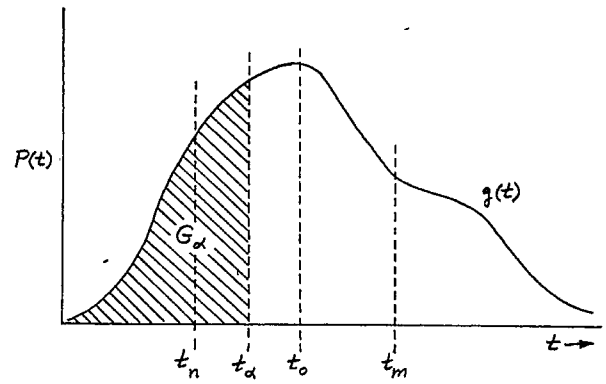
In instances where suitable states may be defined, it may be beneficial to employ a state simulation model [Fig. 8] employing conflict factors approximating system overhead. Unlike a similar concept employed by J.H. Katz[2] in which factors for the states were derived from empirical data and retained in their entirety, the factors associated with each state encountered, for which factors do not already exist, are derived by limited use of an associated detailed simulation model, or derived by computational methods to an accuracy dictated by the application. These are then retained in an associative memory for later use without time-consuming recomputation when the state is again encountered. Use of these approximating factors permit high simulation accuracy of complex systems with shorter run times than would be obtained using conventional simulation techniques.

References

1. B.P. Ochsner, "Controlling a Multiprocessor System", Bell Lab. Rec., Feb. 1966, 59-62.

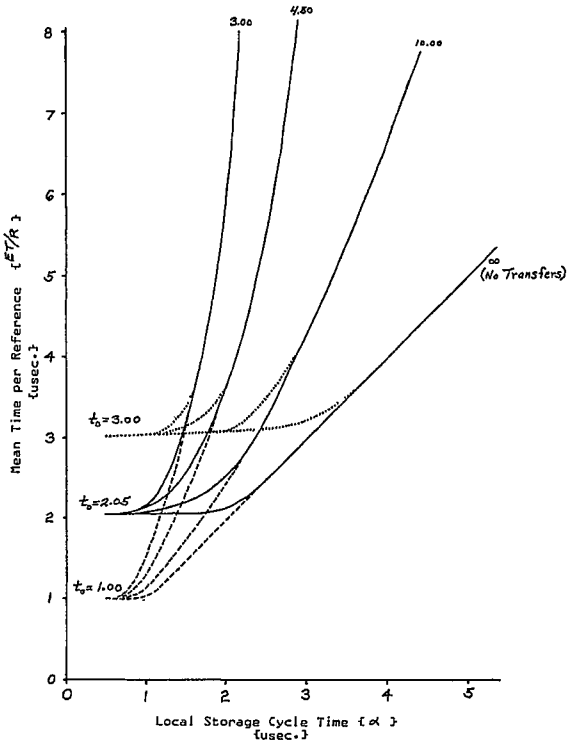
2. J.H. Katz, "Simulation of a Multiprocessor Computer System", Proc. SJCC, 1966, 127-139.

Fig. 2



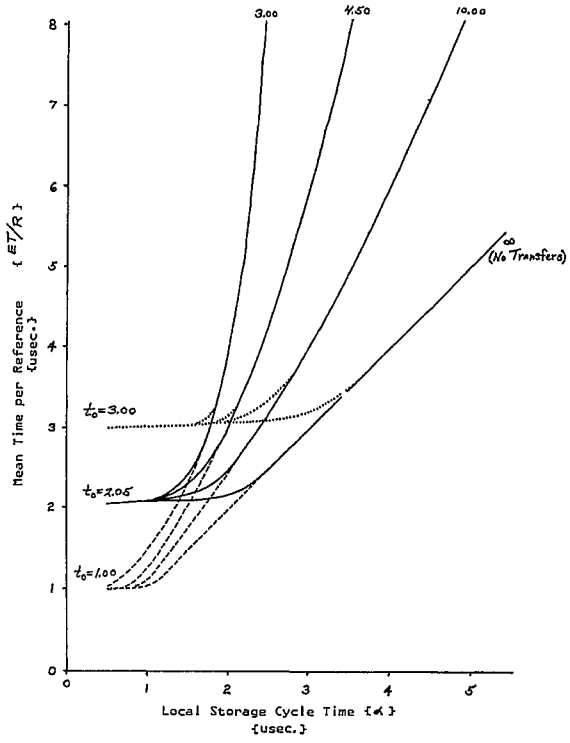
ULTIMATE REFERENCE DISTRIBUTION

Fig. 3



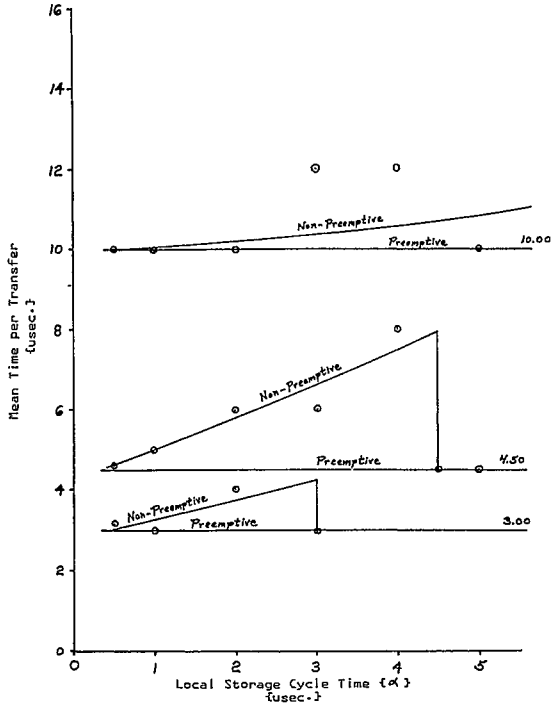
PREEMPTIVE REFERENCES

Fig. 4



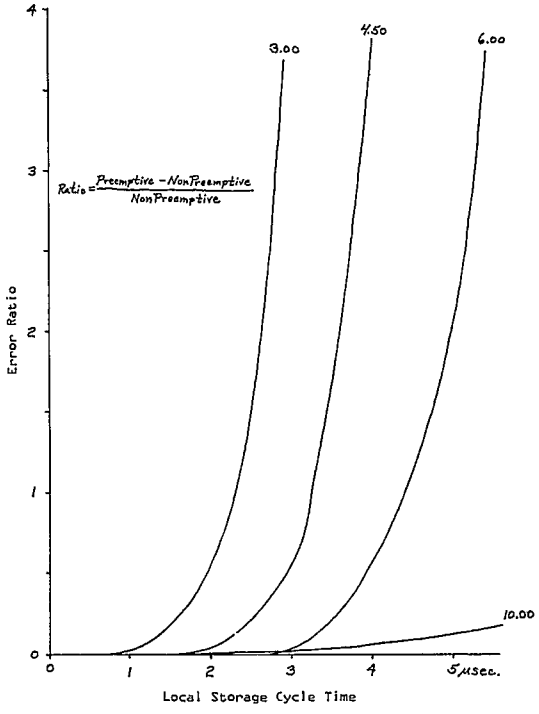
NON-PREEMPTIVE REFERENCES

Fig. 5



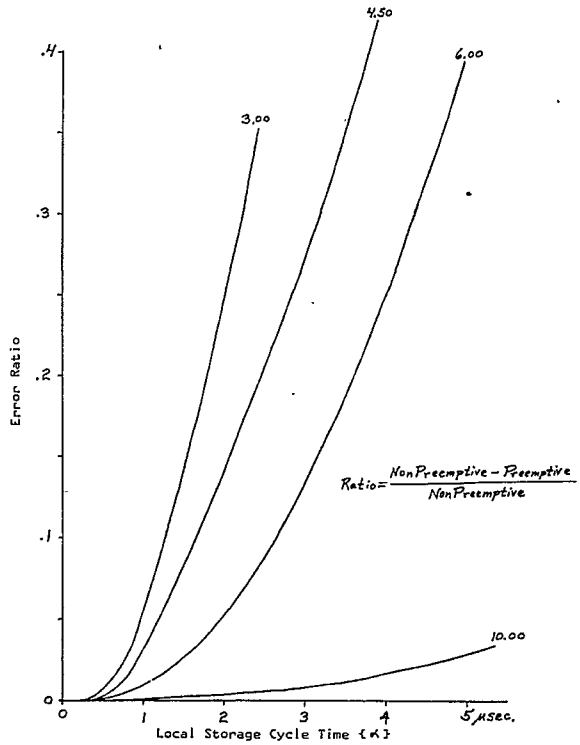
TRANSFERS

Fig. 6



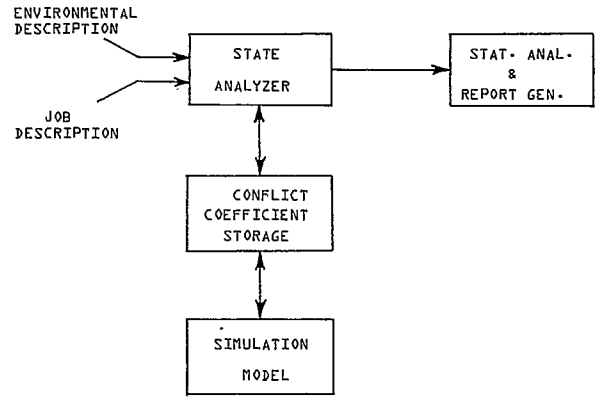
REFERENCE ERROR

Fig. 7



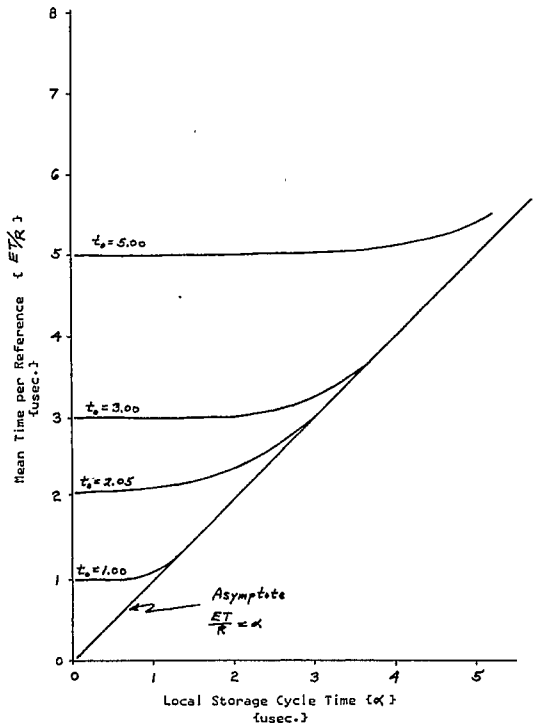
TRANSFER ERROR

Fig. 8



BLOCK DIAGRAM, STATE SIMULATION MODEL

Fig. 9



REFERENCES WITHOUT TRANSFER