Computation Structure Group Memo # 39

Experimental Data for the Working Set Model

By

Lawrence Seligman

Massachusetts Institute of Technology

Project MAC

Abstract

The working set of a "typical" small computer program has been measured: A semiconductor buffer memory system similar to one proposed by Gibson [6] is postulated and its effect on the instruction execution rate is predicted using working set model techniques. The results are verified by simulation.

Introduction

There are today few tools beyond intuition and experience which the computer system architect [1] can use in pursuing his trade. No inclusive theoretic framework exists in which the designer can model a proposed design, characterize its performance, and finally choose parameters to meet cost and performance goals. With the advent of low-cost, high density integrated circuits, new system organizations are becoming economically feasible, organizations outside the designer's realm of experience. The evaluation of these potential organizations presents the opportunity to bring modeling tools to bear.

This study first concentrates on a particular tool, Denning's working set model [4], using it to characterize the addressing patterns of a number of programs running on a small computer. Secondly, it uses this characterization to predict the effect on system performance of the addition of a (semiconductor) buffer memory between the central processor and main (core) memory. The buffer memory helps match the instruction execution rate of the processor to the instruction and data fetch rate of the memory, which is, typically, four times slower.

Background

The earliest organization of this type is the Atlas system [7] which introduced many of the concepts of paging. The memory name space is divided into groups of words called pages. At any time some of the pages

are physically located in a fast memory, the rest in a slower memory. When a word requested is not in the fast memory, the page containing it is moved from the slow memory to the fast memory before the request is granted. Similar techniques are used today in various time-sharing systems. Little is known about the nature of good algorithm for choosing the page to be replaced in the fast memory when a new page must be brought in.

The I.B.M. System 360 model 85 [3,8] and Gibson's somewhat more general work [6] is addressed to a similar problem except that the fast memory is composed of semiconductors (as opposed to core in Atlas) and the slow memory is core (rather that Drum or Disk). These papers report simulation studies used to choose parameters of the buffer memory system; address sequences for a number of programs were collected and then analyzed by programs which simulated the buffer operation and kept appropriate statistics.

Others, for example Coffman and Varian [2], have made empirical studies of system performance in conceptually similar environments. None of these are particularly relevent to our work. However, Denning's work [4] on the working set model for program behavior provides precisely the tool one needs to characterize the addressing patterns generated by the central processor of Figure 1. Rather than using his model for balanced resource allocation in time-sharing systems, we use it to measure

the portion of core memory that is actively in use by a running program. Following is a short explication of our adaptation of the working set model.

The working set $W(t,\tau)$ of a process at time t is the collection of pages referenced in the last τ processor cycles, i.e., during the process time interval $(t-\tau+1,t)$. The working set size $\omega(t,\tau)$ is the number of pages in $W(t,\tau)$. We shall use only $\omega(\tau)$, the working set size averaged over an entire program. The missing page probability $\lambda(\tau)$ is the average rate in process time that a page (re)enters $W(t,\tau)$; $\omega(\tau)$ and $\lambda(\tau)$ are related in the limit by $\omega(\tau) = \int_0^{\tau} \lambda(u)du$ as Denning shows in [5].

The Problem

The problem studied is the relative performance of the two computer systems pictured in Figure 1. The (a) system is a typical, synchronous small computer organization in which the processor requests a word from the memory every C_e seconds, the system cycle time. The (b) system has a similar central processor which is capable of requesting a word every C seconds. If the word is contained in the buffer memory, it is sent immediately; if not, the buffer memory reads b words from main memory, storing them in the buffer, and after a delay of $n \cdot C$ seconds sends the requested word to the processor.

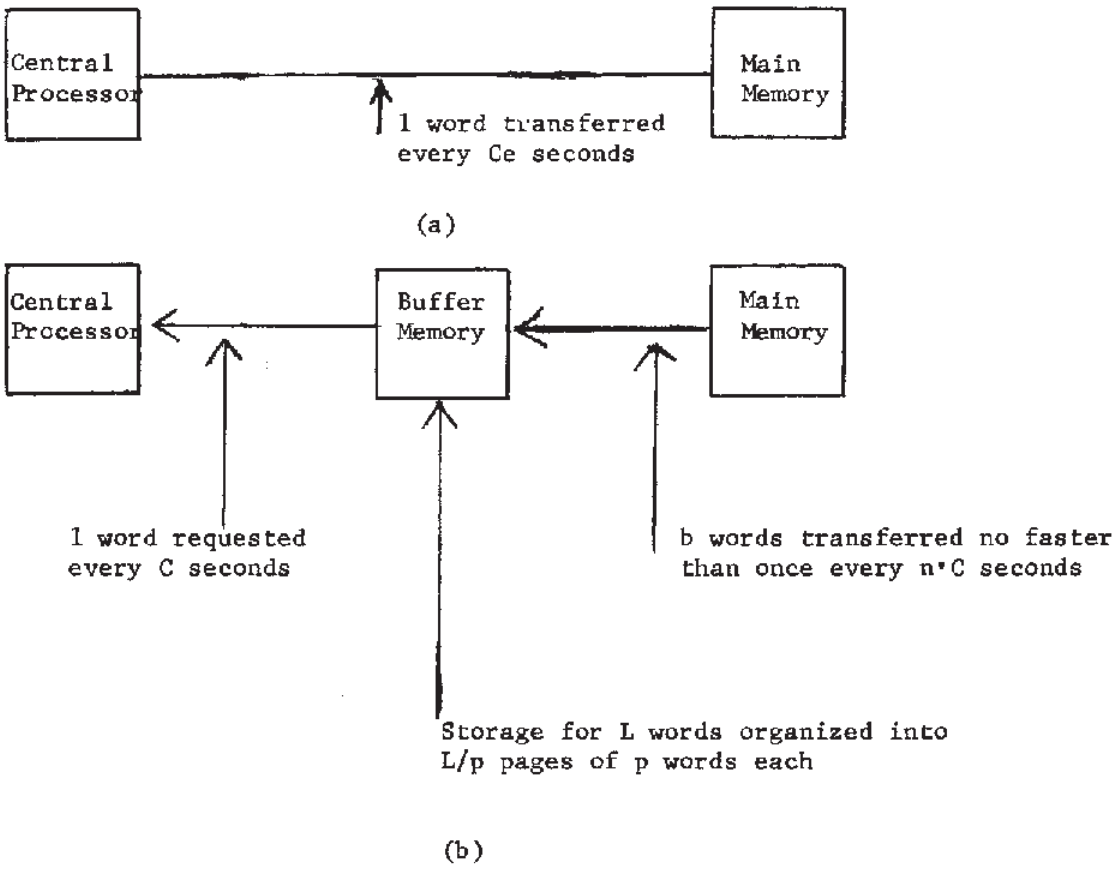

Central Processor
Main Memory
1 word transferred every Ce seconds
(a)
Central Processor
Buffer Memory
Main Memory
1 word requested every C seconds
b words transferred no faster than once every n•C seconds
Storage for L words organized into L/p pages of p words each
(b)

Figure 1

The buffer memory contains L words grouped into L/p pages of p words each; the latter is the unit of space allocation within the buffer memory . For generality we assume that transfers between the main memory and the buffer take place in blocks of b words such that each page contains p/b blocks. If p=b, one is modeling the conventional paged memory organization; if p=b=1, one is modeling an associative memory. In any practical implementation L,p,b would all be powers of 2, and the mappings from addresses to pages and from pages to blocks would be quotient after division by their size ratio.

By "relative performance" we mean the cycle time C_e of the type (a) system which would complete execution of a typical program at the same time as a type (b) system with cycle time C and system characteristics L,p,b,n. We call C_e the effective cycle time of system (b). This definition is highly suspect for two reasons. First, is the concept of a typical program well defined? We argue that the concept is indeed meaningful since the variation in measured quantities among many diverse programs is small, at least for the particular small computer system studied. One can make a strong case that in a more complex processor, the programs are, typically, far more diverse. Secondly, we have not sufficiently characterized the buffer memory's page replacement policy, i.e., the page chosen to be dumped from the memory when a new page renters W(t,T).

The thrust of Denning's work is that the number of buffer memory

pages allocated to a job should depend dynamically on the number of pages required to keep it reasonably active(the looseness of the term "reasonably" is a problem with his model). Since the buffer memory size is fixed, the working set algorithm corresponds to choosing the least recently used page as the one to be replaced.

Methodology

Our work has two major parts: first, to measure the working sets of several programs on a particular small computer; the second, is to relate system performance with a buffer memory to the measured working set data. Three major programs were developed:

(a) A processor simulator to produce magnetic tapes containing address sequences and other data as a result of each processor cycle.

(b) The working set analysis routine producing $\omega(\tau)$ and $\lambda(\tau)$ measurements.

(c) A buffer memory simulator, as used by Gibson, to verify the performance predictions made on the basis of working set analysis. The simulator's algorithms are intentionally quite different from the algorithms used in (b).

Six programs were made part of the study including a Fortran compiler, a macro assembler, 2 hand coded applications programs, and 2 compiled programs. A composite "program" was formed by taking one tape of 72,000 addresses from each of the 6 groups of tapes. This latter "program" is the source of the data presented in the accompanying figures.

Measurements

The basic measurement is the plots of working set size and missing page probabilities as functions of τ. The concept of block size will be introduced later. In Figure 2 we display the data by plotting $\omega(\tau)$ vs $\lambda(\tau)$; a family of curves results from varying the page size. The data is from the composite "program", but all the programs behaved approximately the same. The log-log plot was chosen to best present the data points.

One can connect points from the curves of Figure 2 corresponding to fixed buffer memory sizes; these are presented in Figure 3. Interestingly, the missing page probability (for this data) is minimized for a page size which increases slowly with total memory size. Note that the associative memory organization is not optimun; to borrow a phrase from economics, the marginal utility of the extra words fetched in a page is higher than that of those displaced. One can also connect points from the curves of figure 2 corresponding to equal traffic in words between core memory and the buffer memory. For every word transferred between the processor and the buffer memory, ther are $\lambda \cdot p$ words transferred between the buffer and main memories (neglecting blocking). The half memory traffic curve of Figure 4 connects points in which the word transfer rate seen by the main memory is half the rate seen by the processor.

Figure 2 $\lambda(\tau)$ vs $\omega(\tau)$ showing τ

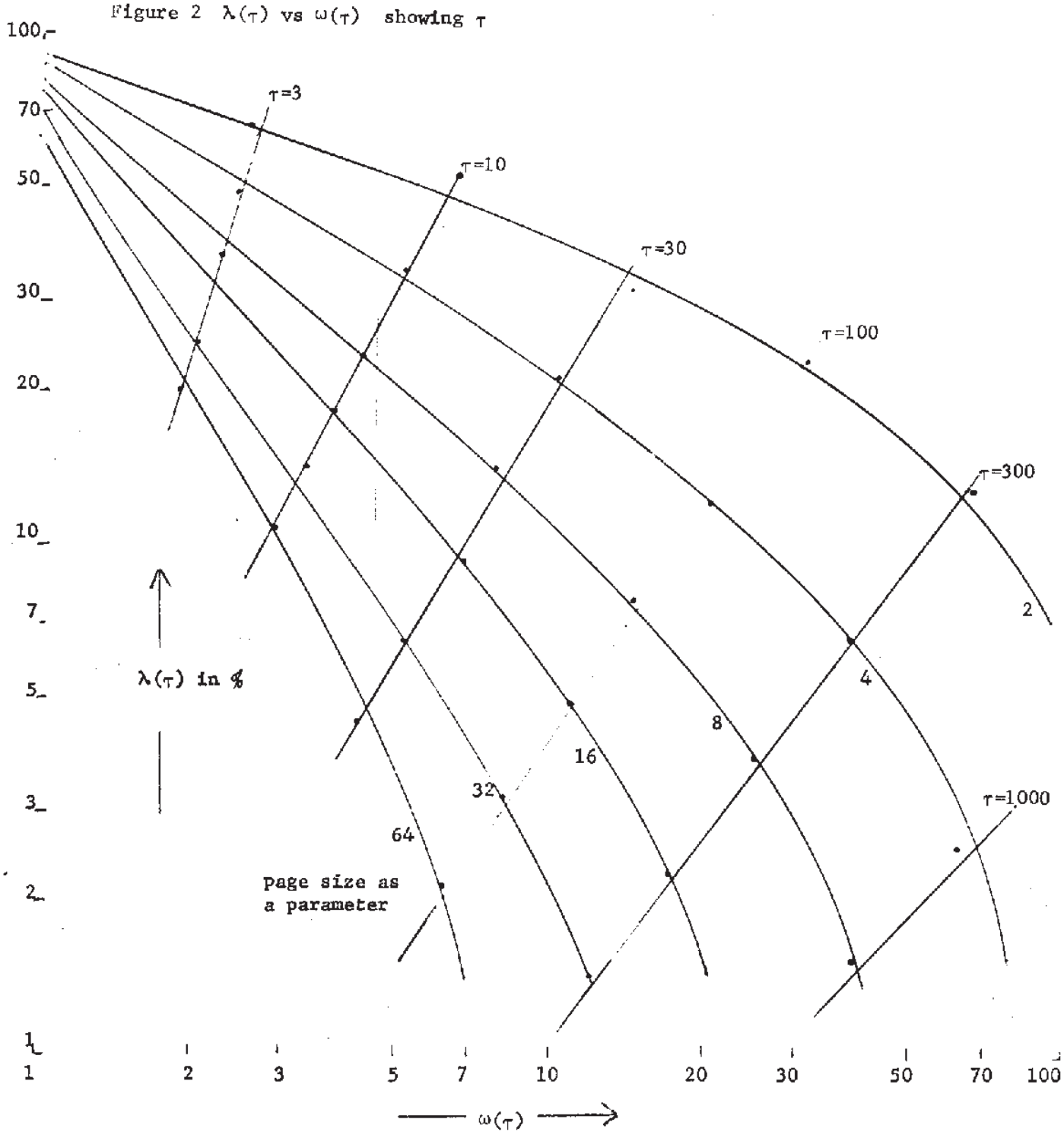


Figure 3 $\lambda(\tau)$ vs $\omega(\tau)$ showing various total memory capacities (L)

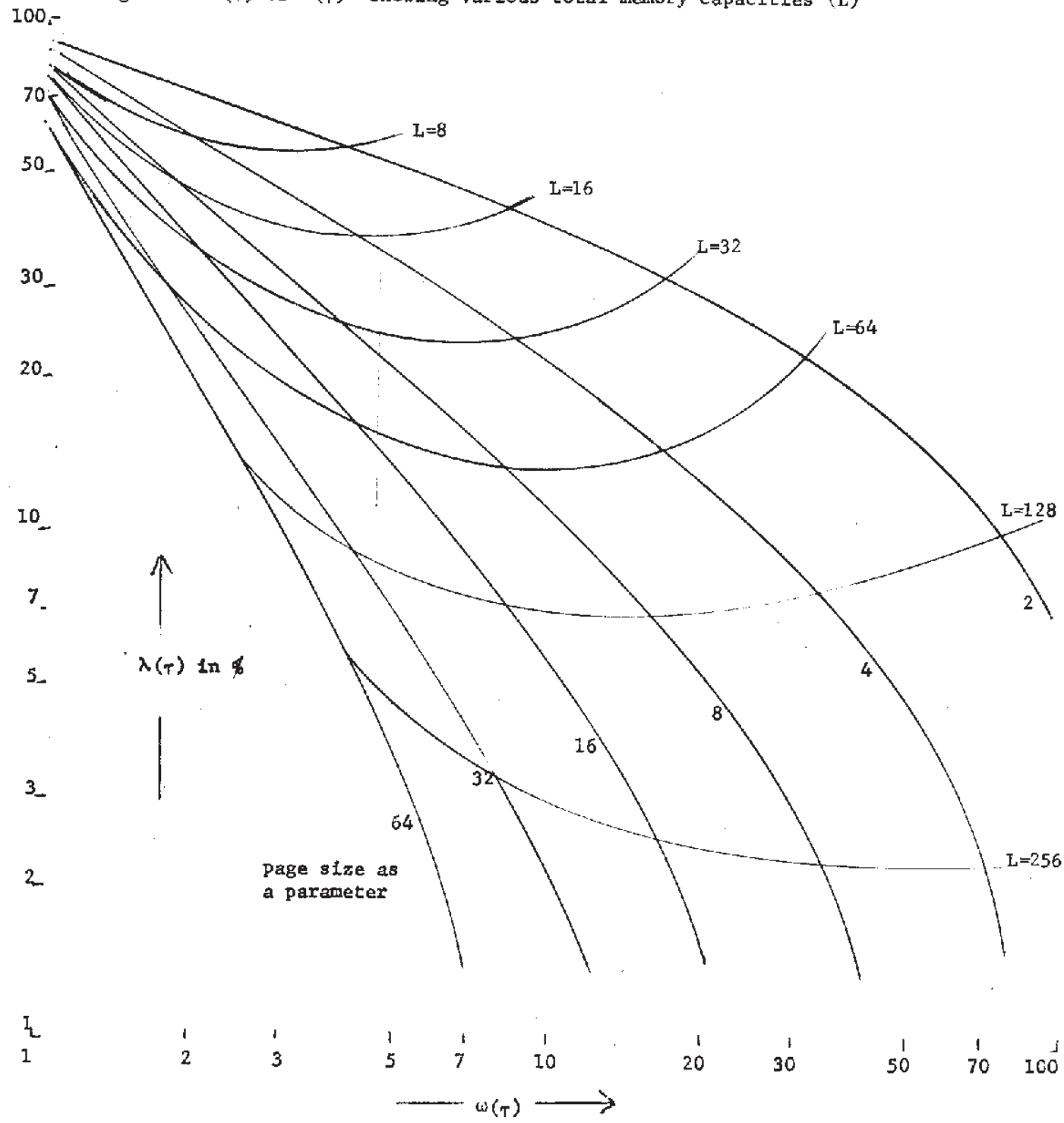
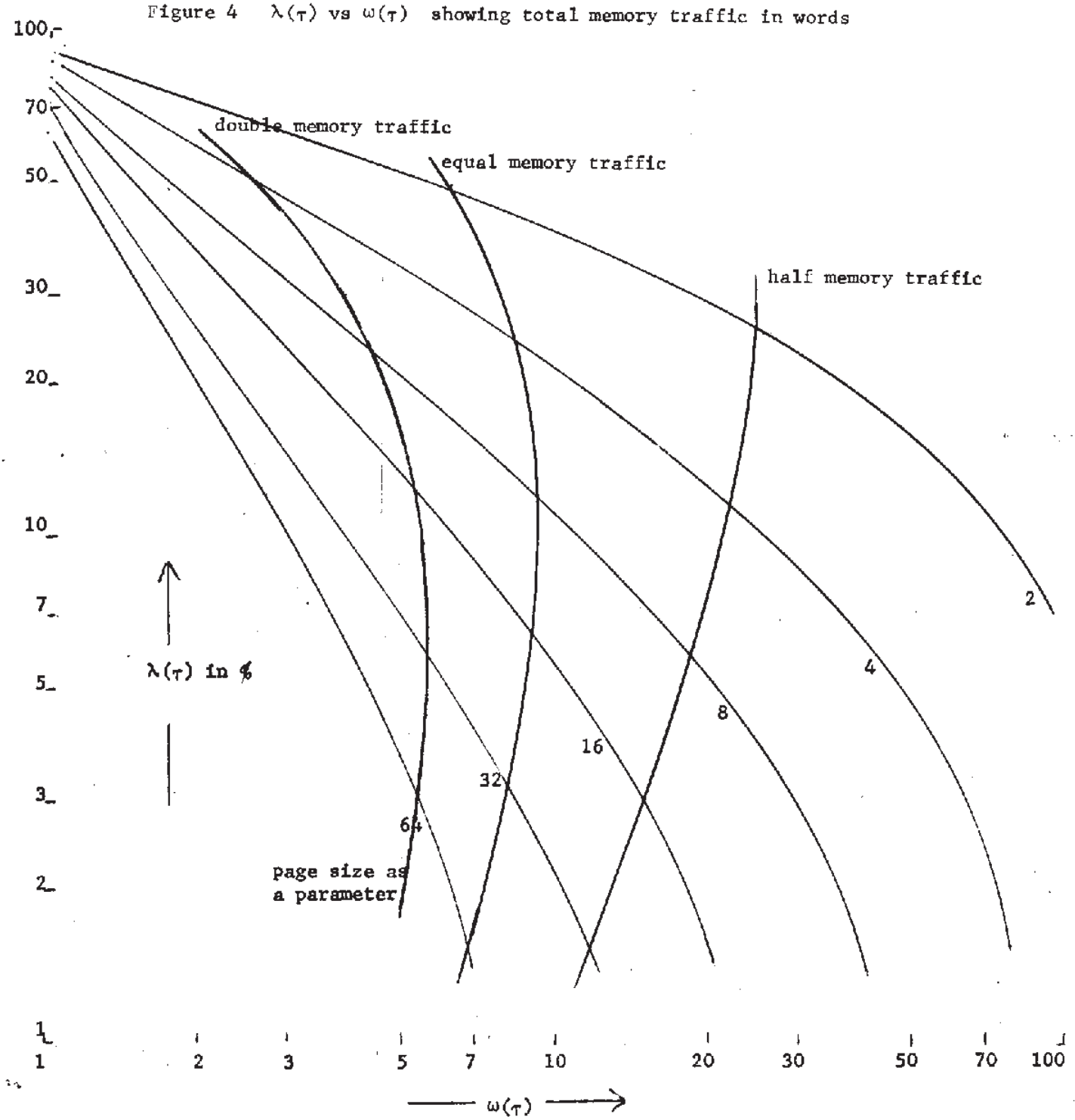


Figure 4 $\lambda(\tau)$ vs $\omega(\tau)$ showing total memory traffic in words



Effect of Block Size

Minimizing the missing page probability does not guarantee the lowest effective cycle time since the model does not account for the actual main memory block size b. We now calculate the effective cycle time taking memory block size into account, assuming that blocks are fetched from main memory to the buffer on demand but that space in the buffer is allocated on the basis of pages. Thus, a request for a word not on a page in the buffer will possibly result in several blocks from an old page being displaced by a single block containing the requested word, since all the blocks corresponding to the replaced page were deleted.

For the earlier case where p = b one calculates

$$C_e = (1-\lambda)(C) + (\lambda)(C + n \cdot C)$$
$$= (1 + \lambda \cdot n)$$

If, on the average k blocks are fetched from among the p/b possible on a page, then every time a word is required from a page not in the buffer $k \cdot n \cdot C$ seconds will eventaully be used to fetch it. Thus $C_e = C(1 + \lambda nk)$ where k is determined below. A working set $W_p(t_o, \tau_o)$ contains $\omega_p(t_o, \tau_o)$ pages of size p while $W_b(t_o,\tau_o)$ contains $\omega_b(t_o, \tau_o)$ pages of size b. The number of referenced blocks of size b in $W_p(t_o, \tau_o)$ is precisely $\omega_b(t_o, \tau_o)$. Averaging over the entire program, one has that the expected number of referenced blocks in a buffer memory with $W_p(\tau_o)$ pages is $\omega_b(\tau_o)$; thus, $k = \omega_b(\tau_o) / \omega_p(\tau_o)$. Referring to Figure 2, one see that for a system with 8, 8

word pages, $\tau_o = 30$ and there are 11 four word blocks actually referenced; $\omega_4(30) = 11$, or about 1.3 per block. One can call $k \cdot \lambda$ the effective missing page probability λ_e; sample curves are given in Figure 5.

Note that a system with page size equal to block size is always better than one where block size is less than the page size (for fixed total buffer memory size) since some of the blocks go essentially unused in the latter organization. However, there are various reasons for using the latter organization, primarily in an attempt at hardware optimization of the buffer memory itself, as in the I.B.M. model 85[8].

One should also note that for the data presented here and in Gibson's [6] 7094 simulation data, use of a buffer memory system with block size b makes possible higher performance than is available with b-way memory interleave. Recall that 2-way interleave means that even addresses appear in one physical memory, odd in another; one takes advantage of the fact that a second memory reference can sometimes be started before the first is completed. The buffer memory size for performance equivalent to b-way interleave can be determined if the interleave characteristics of the memories are known.

In order to verify the results predicted for a buffer memory system with block size less than page size, a simulation was run using the composite data; x's on Figure 5 indicate simulation results. Simulation

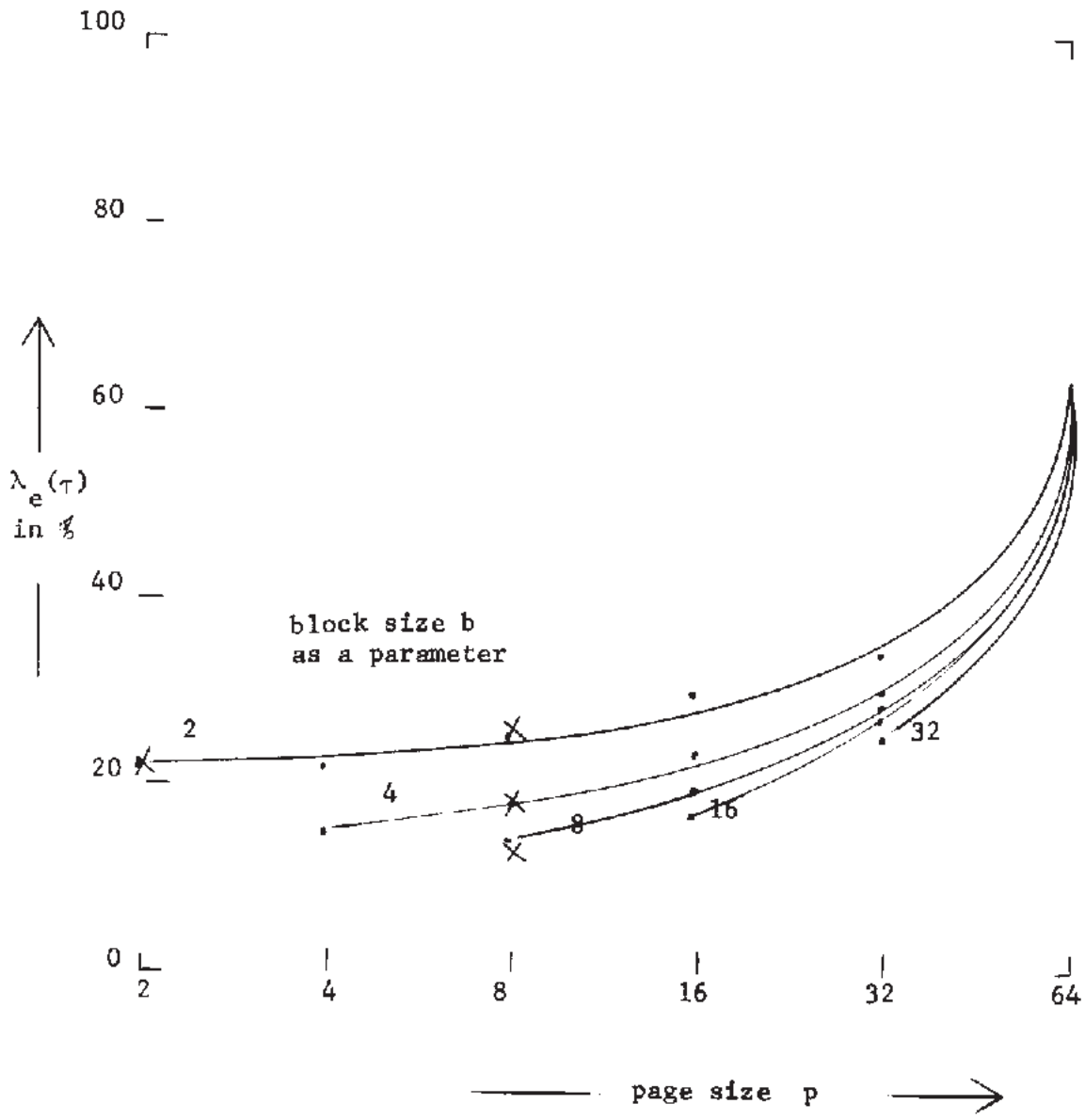


Figure 5

$\lambda_e = k \cdot \lambda$ as a function of page size for a 64 word buffer memory

x's indicate simulation results

also verified the results for other total buffer memory capacities.

Suggestions for further work

The several inadequacies in the system model ought to be remedied. These include:

(a) modeling of processor memory requests which modify memory

(b) modeling of the access time of the memory, as distinct from its cycle

(c) generalizing the model to include effects of interleaving

The working set model itself is not really an adequate tool since it forces a particular page replace algorithm upon the system, which is neither optimal nor easy to implement. Far better would be a modeling technique which permitted a choice of replacement policies. Finally, many of the quantities studied are merely averages; far better would be an analysis based on the distributions of these quantities as random variables.

Conclusions

The two major conclusions are that the working set model leads to meaningful predictions of program behavior in a buffer memory system organization and that such organizations are practical for computer systems where execution speed is of importance. Page sizes should be kept small, but an

associative memory organization is not optimum since the auxiliary words fetched in a block are useful for subsequent instructions. This area of research should prove very fruitful for computer system designers.

REFERENCES

1. Amdahl, G. M. et al., Architecture of the IBM System/360, IBM J. Res. and Dev. 8,2 (April 1964), 87-97.

2. Coffman, E. G., and Varian, L. C. , Further Experimental Data on the Behavior of Programs in a Paging Environment, Comm. ACM 11, 7 (July 1968), 471-474.

3. Conti, C. J. et al., Structural Aspects of the System 360 Model 85, IBM Sys. J. 17,1 (February 1968), 2-14.

4. Denning, P. F., The Working Set Model for Program Behavior, Comm. ACM 11, 5 (May 1968), 323-333.

5. Denning, P. J., Resource Allocation in Multiprocess Computer Systems, M.I.T. Project MAC Report MAC-TR-50 (Ph.D. Thesis), June 1968.

6. Gibson, D. H., Consideration in Block-Oriented Systems Design, AFIPS Conf. Proc. 30 (1968 SJCC), 75-80.

7. Kilburn, T. et al., One Level Storage System, IRE Trans E. C. 11, 2 (April 1962), 223-235.

8. Liptay, J. S., Structural Aspects of the System 360 Model 85, the Cache, IBM Sys, J. 17, 1 (February 1968), 15-21.