Operating Systems

C. Weissman Editor

The Distribution of a Program in Primary and Fast Buffer Storage

Erol Gelenbe
Institut de Recherche d'Informatique et d'Automatique

A virtual memory computer system with a fast buffer (cache) memory between primary memory and the central processing unit is considered. The optimal distribution of a program between the buffer and primary memory is studied using the program's lifetime function. Expressions for the distribution of a program which maximizes the useful fraction of the cost-time integral of primary and fast buffer storage are obtained for swapping and nonswapping buffer management policies.

Key Words and Phrases: cache, virtual memory, lifetime function, cost-time integral, fast buffer

CR Categories: 4.3, 6.2, 6.3

This work was partially supported by a Rackham Research Grant from the University of Michigan. Author's present address: IRIA, Domaine de Voluceau, 78 Rocquencourt, France.

1. Introduction

Economical and technological considerations dictate that there be a wide difference in operating cycle time between the central processing units and the primary storage facility of present day computer systems. While the former is in the order of 100 ns, the latter is in the μs range so that full use of the potential cpu throughout cannot be made with this organization. In recent years the inclusion of a small high-speed memory to serve as a buffer between the cpu and primary memory has become technologically and economically feasible [1], and has been shown to allow the system throughput to approach that of one in which primary memory speed is commensurate with cpu speed. The purpose of this paper is to study an aspect of the design of such a buffer memory, namely its size, in the context of a virtual memory system.

Virtual memory gives the programmer the illusion of programming with no constraints on the amount of primary memory space available to him by automatically managing several levels of storage [2]. This approach is particularly useful in multiprogrammed or time-shared systems since here several programs reside in the system simultaneously, with subsequent multiplexing of the cpu and the input-output units among them. It is common with such systems that all portions of a program cannot be maintained in primary memory so that "faults" to secondary storage occur when information not in primary memory is referred to. The average time between these faults depends on the amount of primary storage space available to a program, on the manner in which the portions of a program residing in primary memory are chosen, and on other factors. Based on empirical evidence, Belady and Kuehner [3] have characterized the dependence of the mean time between faults upon the storage space available to a program by means of their lifetime function, which is used in this paper to describe program behavior.

2. Preliminaries

In this section we shall briefly review the fast buffer (cache) memory operation as described in [1], and provide an outline of the basic ideas related to the Belady-Kuehner lifetime function [3].

In IBM 360/85 implementation [1], both the fast buffer (FB) and primary memory (PM) are partitioned into equal-sized sectors so that at a given instant of time each FB sector is assigned to contain the contents of some PM sector. When a program generates a reference to a sector not in the FB, the FB sector which has been least recently used is assigned to the PM sector just referred to. The whole contents of the PM sector are not transferred all at once into the FB sector, however. Each sector consists of 64 words only four of which (a block) are transferred, with the word which was just referred to being transferred first and being loaded simultaneously into the cpu. The remaining blocks of the sector are loaded into the FB sector set aside for them as demand arises.

The subject of our study is a similar organization within a virtual memory machine [2].

The lifetime function $e(s)$ for a program computing with s amount of storage (which may be less than the size of the program) was introduced by Belady and Kuehner [3] to characterize the mean time between faults to secondary storage for programs running on a virtual memory machine. Based on empirical evidence, they indicate that for a useful range $0 \leqslant s \leqslant R$, the lifetime function is given by $e(s) = a'a''s^k$ where a' is a program property, a'' relates the cycle time of the storage medium from which the program is being executed to real time, and $1.5 \leqslant k \leqslant 2.5$. This expression is not based on the assumption of any particular memory management policy, and it is stated [3] that changing this policy will not affect the form of $e(s)$ drastically. By memory management policy here we will mean two things: the replacement policy (e.g. least recently used, last-in-first-out, etc.) [2]; and the size of the blocks being replaced (the page size). Throughout this paper it will be assumed that the same replacement policy[1] is used for the FB and PM, and that for the page and block sizes in question (we will use the term "block" for the unit of storage transfer between FB and PM, and "page" for the transfers between PM and the secondary storage medium) the lifetime function is invariant. Our central assumption is that the FB and PM space available to a program is such that the above expression for the lifetime function is valid. We will write $a = a'a''$, with a'' being the value obtained for the FB.

3. High-Speed Buffer Size as a Function of Primary Storage Space

Suppose that P is the amount of primary storage space occupied by some program, and let B be the size of fast buffer space it occupies. If $e(s)$ is the average time between faults for the program for storage space s (the lifetime function), assuming s refers to allocated buffer space, we see that on the average a reference to primary memory is generated each $e(B)$ seconds during computation of the program. When such a reference occurs, assume that a fixed time u elapses to transfer a (fixed sized) block of information from primary to fast buffer memory. Evidently it will sometimes happen that the information referred to is not in primary memory, in which case a fixed time y is spent in transferring a (fixed sized) page from secondary to primary memory, and an additional time u is spent in bringing a required portion of the page into the buffer. For a total computing time between faults to secondary storage of $e(P)$, $e(P)/e(B)$ faults to primary storage will occur. This statement requires clarification. If the size of the fast buffer had been P, $e(P)$ would be the uninterrupted computing time in the buffer. With the present organization, however, this amount of computation time is interrupted $e(P)/e(B)$ times. Note that the contents of the buffer pertaining to the program in question are also contained in primary memory, and that this discussion is valid if the buffer and primary memory are managed using the same replacement algorithm.

The cost of maintaining the program in primary and fast buffer memory is

$$\xi = P + cB \tag{1}$$

in terms of the unit cost of primary memory, where c is the unit cost of buffer memory per unit cost of primary memory. In a total cost-time integral of

$$\xi(e(P) + (e(P)/e(B))u + y) \tag{2}$$

only $\xi e(P)$ is expended in useful computation. The ratio

$$\eta = \frac{e(P)}{e(P) + (e(P)/e(B))u + y} \tag{3}$$

is then a measure of utilization of the cost-time integral, assuming that the program does not have to wait for the attention of the cpu, and that the time the program spends in acquiring its space P is short compared to its total residence time in the system. Belady and Kuehner [3] have indicated that for a useful range of values of s, starting at $s = 0$, $e(s) = a\,s^k$ represents the lifetime function fairly accurately, a and k being constants, where a is program and system dependent and $k \approx 2$. For a detailed discussion of this claim the reader is referred to [3].

η is a measure of how efficiently storage is being utilized by a program. Therefore let ξ, the buffer and primary storage cost for the program, be kept constant, and let us examine how $\alpha = P/B$ affects η. Using $e(s) = a\,s^k$ we now rewrite

$$\eta = aP^k/(aP^k + u\alpha^k + y), \tag{4}$$

and since

$$P = \xi\alpha/(c + \alpha), \tag{5}$$

we remain with

$$\eta = \frac{a\xi^k\alpha^k}{a\xi^k\alpha^k + u(c + \alpha)^k\alpha^k + y(c + \alpha)^k} \tag{6}$$

Keeping ξ constant and solving $d\eta/d\alpha = 0$ we see that η is maximized for α given by

$$\alpha_0 = [(y/u)c]^{1/(k+1)} \tag{7}$$

since

$$d^2\eta/d\alpha^2\,|_{\alpha=\alpha_0} < 0.$$

[1] If the contents of the FB are also held in PM, it is necessary that the replacement policy does not violate this as would be the case with a random policy.

That α_0 should grow with y/u and with c is not counter-intuitive. It is somewhat surprising, however, that it is independent of a since it represents some program properties as well as the speed of computation [3]. Another interesting aspect is the variation of α_0 with k. A larger k is indicative of greater program locality, or better memory management, or both; with programs performing better in these respects, one sees that more space should be allocated to the fast buffer with constant cost ξ. Finally, it is interesting to note that α_0 is independent of ξ, and can be deduced solely from properties of the system and the program.

The lifetime function, as defined and measured by Belady and Kuehner [3] and used in this paper, makes no distinction between the execution of instructions which modify program data and those which do not, or merely involve "read" accesses to memory. The implicit assumption throughout this section has been that instructions which modify program data are executed only in the fast buffer (as all other instructions) so that whenever a block in the fast buffer is to be replaced, it is copied back into primary memory if it has been modified. If special hardware facilities exist, it is possible to carry out this transfer from the FB into PM simultaneously with the transfer of a block from PM to the FB.

In the IBM 360/85, on the other hand, each instruction modifying data must be executed both in the FB and in PM. This evidently slows down the operation of the system. In the next paragraph we examine the optimum choice of α in the case that "write" instructions are executed in PM as well as the FB.

Let r write instructions per unit time be executed in the FB, and suppose that the execution of a write instruction requires w' units of time in the FB and w units of time in PM. It is evident that $r \cdot e(B)$ is the number of write instructions executed on the average in the FB before a reference is made to a block which is not in the FB but is in PM. Clearly, $r \cdot e(B) \cdot w'$ is a fraction of $e(B)$ and represents the part of $e(B)$ expended in writes. If writes have to be executed into PM as well as the buffer, this can be done in two ways: either the write in PM and the FB are executed sequentially (one after the other), so that the actual time between references to a block not in the FB but in PM is $e(B) \cdot (1 + wr)$ with a useful fraction of the cost-time integral we call η_1; or the writes are executed in parallel so that the corresponding length of time is $e(B) \cdot (1 + (w - w')r)$ with the reasonable assumption that $w > w'$ leading to a useful fraction of the cost-time integral η_2.

It is seen that [2]

$$\eta_1 = \frac{e(P) \cdot (1 + w \cdot r)}{e(P) + (e(P)/e(B))(u + wr \cdot e(B)) + y} \tag{8}$$

and

$$\eta_2 = \frac{e(P) \cdot (1 + w \cdot r)}{e(P) + (e(P)/e(B))(u + (w - w')r \cdot e(B))y}. \tag{9}$$

For both (8) and (9), the value of α maximizing η_1, η_2, is a solution of $ak\xi^k\alpha^{k-1}(c + \alpha)^{k-1}(cy - u\alpha^{k+1}) = 0$. The solution of the equation is given by

$$\alpha = 0, \; -c, \; [cy/u]^{1/k+1}.$$

The first solution sets (8) and (9) to zero, while the second is unrealizable. Therefore the value of α maximizing (8) and (9) is given by

$$\alpha_0 = [cy/u]^{1/k+1} \quad \text{for which}$$

$$d^2\eta_1/d\alpha^2 \mid \alpha = \alpha_0 < 0, \quad \text{and} \quad d^2\eta_2/d\alpha^2 \mid \alpha = \alpha_0 < 0.$$

Thus we see that the effect of write operations reduces the useful fraction of the cost-time integral but leaves α_0 unchanged.

4. Independent Fast Buffer and Primary Memory Contents

In the previous sections the assumption had been that all program sectors in the FB were also contained in PM. This approach is inefficient in its utilization of memory space but frees the system from the necessity of transferring blocks from the FB to PM when blocks not in the FB are referenced, if writes are executed in both FB and PM.

This section is devoted to the analysis of a management policy which maintains disjoint portions of a program in the FB and PM so that total memory space is more efficiently used. Such a technique is not without its costs. At each reference to a block of information which is in PM, but not in the FB, it is necessary to remove one of the blocks in the FB and to copy it into PM to make a place for the incoming block. This requires special hardware, as well as a buffering area in PM. An additional requirement is some form of address translation hardware similar to that found in paged virtual memory systems—since program address references will have to be translated into appropriate PM or FB physical addresses. The more efficient utilization of space, however, may be well worth this additional cost. The useful fraction of the cost-time integral now is

$$\eta_3 = \frac{e(P + B)}{e(P + B) + (v + u) \cdot (e(P + B)/e(B)) + y} \tag{10}$$

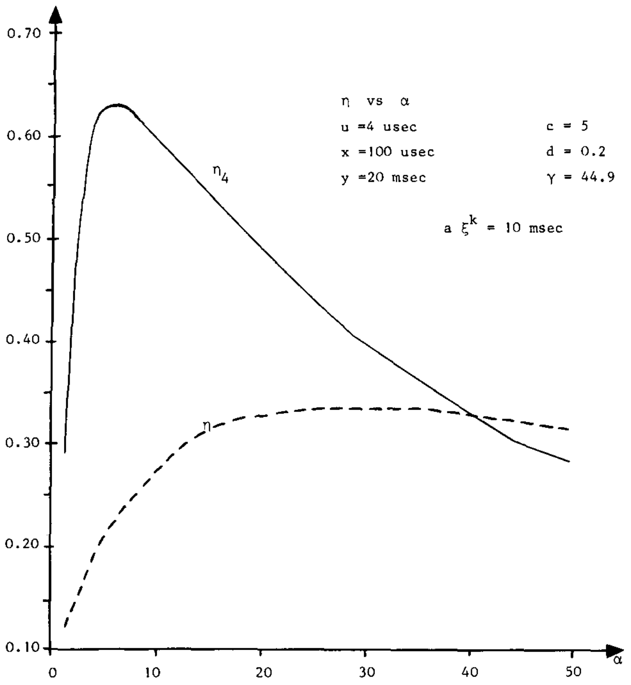
where v is the additional time necessary to transfer a block from FB to PM when a block in PM but not in FB is referenced. Note that v may be zero if the transfer into FB is carried out simultaneously with the transfer out. Using (5) and the expression for $e(s)$, we obtain an expression whose derivative is given by

$$d\eta_3/d\alpha = (c + \alpha)^{k-1} \cdot k \frac{a\xi^k}{X^2} (1 + \alpha)^{k-1} \cdot (y(c - 1) - (1 + \alpha)^{k+1}(v + u)) \tag{11}$$

where X is the denominator of (10) and as usual

[2] Note that the multiplicative factor $(1 + w.r)$ in (8) and (9) represents the slowdown due to write operations and does not affect the optimum value of α.

Fig. 1.



$\alpha = P/B$. It is seen that the value of α maximizing η_3 is given by

$$\alpha_0' = \left[\frac{(c-1)y}{v+u}\right]^{1/k+1} - 1 \qquad (12)$$

5. Extension to Other Hierarchies and Conclusions

The previous sections have been devoted to the presentation of a simple and useful analytic method for the design of a storage hierarchy consisting of a cache memory and primary memory backed by a secondary storage device. It is evident that this method can be extended to design more complicated hierarchical structures. For more precise results, it is also possible to use more accurate expressions for the lifetime function. Of course, it is to be expected that closed form expressions (like the one obtained for α_0 above) will not always be obtainable so that one will have to turn to numerical methods.

In what follows, we will examine a more complicated hierarchy than the one studied above, and it will be shown that this more complicated hierarchy will yield a larger η, if properly designed, than the FB-PM-Secondary Memory hierarchy studied above, for the same cost ξ.

Consider a hierarchy consisting of an FB, a PM, a slow core (SC) memory, and a secondary storage (drum or disk). Suppose that all of the information contained in the FB is also in PM. Information in PM is disjoint of that in SC so that when a reference is made to a sector in SC but not in PM (and therefore not in the FB) the sector is transferred from SC to PM and some sector in PM is transferred into SC to make place for an incoming sector. Of course, special hardware facilities and address translation hardware is assumed to be available. The replacement algorithm (Least-Recently-Used or First-In-First-Out, etc.) as well as sector size is assumed to be the same at all levels.

Let B, P, E, be the sizes of FB, PM, SC, respectively. We will use c to denote the unit cost of FB space per unit cost of PM space; similarly, d is the unit cost of SC space per unit cost of PM space. Thus, the total cost of the hierarchy is $\xi = P + cB + dE$ per unit PM cost. We also use $\alpha = (P/B)$ and $\gamma = (E/B)$. Thus, the useful fraction η_4 of the cost-time integral, if write operations into PM are neglected, is given by

$$\eta_4 = \frac{e(P+E)}{e(P+E) + (e(P+E)/e(P))x + (e(P)/e(B))\,u + y}, \qquad (13)$$

where u is the time to transfer a block from PM to the FB, x is the time to transfer a block from SC into PM and the FB (and to transfer out a block from PM into SC), and y is the time necessary to transfer a block from the rotating (drum or disk) secondary memory into SC. Equation (13) is rewritten as

$$\eta_4 = \frac{a\xi^k(\alpha+\gamma)^k}{a\xi^k(\alpha+\gamma)^k + (1+\gamma/\alpha)^k H^k x + \alpha^k H^k u + yH^k}, \qquad (14)$$

where $H = \alpha + c + d\gamma$. Some analysis shows us that the values of α and γ maximizing η_4 are the solutions of two simultaneous nonlinear equations for which closed form solutions cannot be determined. Therefore, in Figure 1, η_4 has been plotted against α for a constant value of γ which maximizes γ_4 for the various parameter values used in the example.

The optimum value of γ is also determined numerically. The plotted curve shows that for this case η_4 is highly sensitive to variations in α near its maximum. In the same figure, η of eq. (3) has been plotted against α for the same parameter values and, in particular, the same cost ξ.

One sees that, for the same cost, the more elaborate hierarchy yields a much better utilization of the cost-time integral, though this cost does not include that for special hardware to implement the hierarchies (i.e. only the memory space cost is included). In both curves, the effect of read operations into PM has been neglected.

Received July 1971; revised July 1972

References

1. Liptay, J.S. Structural aspects of the System/360 Model 85 II, The Cache. *IBM Syst. J. 7*, 1 (1968), 15–21.
2. Denning P.J. Virtual memory. *Computing Surveys 2*, 3 (Sept. 1970), 151–189.
3. Belady, L.A., and Kuehner, C.J. Dynamic space sharing in computer systems. *Comm. ACM 12*, 5 (May 1969), 282–288.