J. E. MacDonald
K. L. Sigworth

Storage Hierarchy Optimization Procedure

Abstract: The goal of storage hierarchies is to combine several storage technologies in such a way as to approach the performance of the fastest component technology and the cost of the least expensive one. This paper presents optimization techniques for a storage hierarchy subject to quantity-sensitive component costs. It is assumed that a finite (and probably small) set of technologies is available. Each technology is characterized by an access time and two cost parameters. We assume that statistical summaries of address sequences are available. We present solutions to four problems of increasing complexity: 1) minimization of access time for a fixed cost and preassigned page sizes; 2) optimization of a generalized price-performance function under preassigned page sizes; 3) minimization of access time for a fixed cost when page sizes are allowed to vary; 4) optimization of a generalized price-performance function when page sizes are allowed to vary.

Introduction

In the best of worlds, the storage facility of a computer system would be large, fast, and cheap. In the real world this triplet of characteristics has not yet been attained. Although many significant strides have been made, it has not been possible to devise a single technology that meets all three goals.

As a consequence, much attention has been devoted to methods of exploiting the statistical properties of the sequence of storage addresses which characterize the communication between a processor and a storage during the execution of a computer program [1, 2]. Parallel efforts have concentrated on efficient techniques for gathering and presenting statistical summaries of typical address sequences [3, 4]. In this paper, we assume the existence of these statistical summaries and describe how they may be used to optimize the design of the storage hierarchy.

Suppose that we have available B different storage technologies. The simplest hardware design for use in exploiting the statistics of the address sequence is a B-level hierarchy, where Level 1 is the smallest, fastest, and most expensive, and Level B is the largest, slowest, and least expensive. A request for β_0 contiguous bytes is directed from the processor to Level 1. If Level 1 contains the desired β_0 bytes, they are transferred to the processor, and the Level 1 directory is updated to reflect the transaction. If Level 1 does not contain the desired β_0 bytes, the request is directed to Level 2. If Level 2 contains the desired β_0 bytes, a block of size β_1 containing those β_0 bytes is transferred from Level 2 to Level 1, and the appropriate β_0 bytes are simultaneously transferred to the processor. The directories at Level 1 and Level 2 are updated to reflect this transaction. If Level 1 has become filled with blocks of β_1 bytes, it will be necessary to evict a block of β_1 bytes from Level 1 to make room for the block coming to it from Level 2. This eviction must also be posted to the Level 1 directory. We assume that the evicted block need not be pushed back to Level 2 since a copy is presumed to exist at Level 2. The block to be evicted is determined by the replacement algorithm. If Level 2 does not contain the desired bytes, but Level 3 does, then β_2 bytes containing the desired β_0 bytes are transferred to Level 2; of these, β_1 bytes are transferred to Level 1, and, of these, β_0 bytes are transferred to the processor. Directories at Levels 1, 2, and 3 are updated concurrent with this transaction. If Level 3 does not contain the required β_0 bytes, the process continues until the required β_0 bytes are found at some level. We assume that the required bytes are available at Level B with probability one. For $i = 1, 2, \cdots, B-1$, let α_i be the number of blocks of size β_i bytes which are resident at Level i. If we denote by s_i the size of Level i in bytes, we obtain

$$s_i = \alpha_i \beta_i. \tag{1}$$

Obviously we are assuming a discrete block organization at each level such that α_i and β_i are integers and $\beta_i > \beta_{i-1}$, where α_i = number of blocks at Level i and β_i = size in bytes of an individual block at Level i.

The size of Level B, s_B, deserves some comment. The minimum size of s_B will be

$$\min s_B = \beta_{B-1}\, n_{B-1}, \tag{2}$$

where n_{B-1} = number of *distinct* blocks of size β_{B-1} referenced by the address sequence. Any size of s_B larger than min s_B will add to the cost without improving performance. We assume that n_{B-1} is known. The value of n_{B-1} is referred to as "program size in β_{B-1}-sized blocks."

We assume that the statistical data are known in the form of the following cumulative distribution function for $i = 1, 2, \cdots, B-1$:

$$G_{\beta_i}(\alpha_i) = \text{Probability that the next required address is } \textit{available} \text{ in Level } i, \text{ when it contains } \alpha_i \text{ blocks of } \beta_i \text{ bytes.} \quad (3)$$

The simplest performance-oriented figure of merit for a storage hierarchy is the average access time, given by

$$E = \sum_{i=1}^{B} v_i \Pr(R_i), \quad (4)$$

where E is the average access time (seconds), v_i is the time required for processor to access data from Level i (seconds), and $\Pr(R_i)$ is the probability that the next reference will be made to Level i.

There exist many different replacement algorithms [3-5]. Of these, the Least Recently Used (LRU) seems to represent a good engineering compromise between performance and cost. In addition, LRU has two characteristics that are particularly applicable to this paper; 1) LRU belongs to a class of algorithms called stack algorithms [4], meaning that the $B-1$ functions implied by (3) can be obtained by a single scan of an address sequence whose addresses imply a block size of β_0 bytes, and 2) LRU applied to Levels 1 and 2 of a three level storage hierarchy has the following properties [6]

$$\begin{aligned} \Pr(R_1) &= G_{\beta_1}(\alpha_1) \\ \Pr(R_2) &= G_{\beta_2}(\alpha_2) - G_{\beta_1}(\alpha_1) \\ \Pr(R_3) &= 1 - G_{\beta_2}(\alpha_2). \end{aligned} \quad (5)$$

Slutz and Traiger [7] extended this result to B-level hierarchies and found

$$\begin{aligned} \Pr(R_1) &= G_{\beta_1}(\alpha_1) \\ \Pr(R_i) &= G_{\beta_i}(\alpha_i) - G_{\beta_{i-1}}(\alpha_{i-1}) \text{ for } i = 2, 3, \cdots, B-1 \\ \Pr(R_B) &= 1 - G_{\beta_{B-1}}(\alpha_{B-1}). \end{aligned} \quad (6)$$

For (5) and (6) to hold, it first appeared that a necessary condition is that the address stream from the processor should be used to update each of the $B-1$ directories in a B-level hierarchy for each address. Gecsei [8] has shown that this condition is not necessary for (5) and (6) under LRU. Instead Gecsei has described a method of distributed hierarchy management using LRU for which (5) and (6) still hold. We implicitly assume Gecsei's distributed management technique in any implementation. All we require mathematically, however, is the validity of (6).

Cost considerations

We assume the use of a different technology at each of the B levels, and, loosely speaking, the faster the level the higher the cost. The actual situation, however, is somewhat more complex. Any candidate technology has the property that the average cost per bit is a monotone non-increasing function of the quantity of bits produced. Inspection of several such functions suggests that they are sufficiently well represented by

$$(C_i - c_i) Q_i = t_i, \quad (7)$$

where C_i is the average cost per bit of technology i (dollars/bit), Q_i is the quantity produced of technology i (bits), c_i is a constant for technology i (dollars/bit), and t_i is the other constant for technology i (dollars). Equation (7) can be rewritten in the following form

$$T_i = t_i + c_i Q_i, \quad (8)$$

where T_i is the total cost of technology i (in dollars). It then becomes apparent that the constants t_i and c_i can be given a physical interpretation. The constant t_i is a "start-up" cost, and the constant c_i is an incremental cost per bit after the decision is made to use technology i and invest the start-up cost t_i.

Let P_i be the fixed costs per unit produced at Level i for power, directories, etc. (dollars), N be the number of units of the storage hierarchy to be produced, J be the discretionary cost per unit (dollars), $\mathscr{C}$ be the total cost (dollars), and l be the number of bits per byte. Then we find

$$\mathscr{C} = \sum_{i=1}^{B} t_i + N \sum_{i=1}^{B} (P_i + l c_i \alpha_i \beta_i). \quad (9)$$

Let

$$J = (\mathscr{C} - \sum_{i=1}^{B} t_i - N \sum_{i=1}^{B} P_i - N l c_B s_B) / N. \quad (10)$$

Then

$$J = \sum_{i=1}^{B-1} l c_i \alpha_i \beta_i. \quad (11)$$

It can be seen that the right hand side of (10) begins with a total cost $\mathscr{C}$, decrements that cost by the fixed costs, and divides the discretionary cost so obtained by N to yield the discretionary cost per unit. This is confirmed by (11) where we assume l, c_i, β_i, and J to be already chosen and the α_i to be unknown. Letting

$$k_i = l c_i \quad (12)$$

we may rewrite (11) as a cost constraint equation

$$H = 0 = N \sum_{i=1}^{B-1} k_i \alpha_i \beta_i - NJ, \quad (13)$$

where now the α_i must be chosen to make H identically equal to zero.

Minimization of access time for a fixed cost

The central problem of this section is to choose α_1 through α_{B-1} in such a way as to minimize E (the average access time) while obeying the cost constraint (13). We approach this problem by using the method of Lagrange multipliers. Let

$$F = E + \lambda H, \tag{14}$$

where λ is the Lagrange multiplier. We then express E as a function of the α_i by making use of (4) and (6) so that

$$E = v_1 G_{\beta 1}(\alpha_1) + \sum_{i=2}^{B-1} v_i [G_{\beta_i}(\alpha_i) - G_{\beta_{i-1}}(\alpha_{i-1})] + v_B [1 - G_{\beta_{B-1}}(\alpha_{B-1})]. \tag{15}$$

Equation (15) can be rewritten in a more convenient form as

$$E = \sum_{i=1}^{B-1} (v_i - v_{i+1}) G_{\beta_i}(\alpha_i) + v_B, \tag{16}$$

where $v_i - v_{i+1} < 0$.

In order to find the values of α_1 through α_{B-1} we solve the system of equations

$$\begin{cases} H = 0 = N \sum_{i=1}^{B-1} (k_i \alpha_i \beta_i) - NJ \\ \dfrac{\partial F}{\partial \alpha_i} = 0 = (v_i - v_{i+1}) \dfrac{\partial G_{\beta_i}(\alpha_i)}{\partial \alpha_i} + \lambda N k_i \beta_i. \end{cases} \tag{17}$$

The minimum value of E occurs when the solution vector is substituted into (16). The solution of this system of B equations in B unknowns is rendered relatively easy by the observation that the second line of (17) involves only two of the B unknowns, namely α_i and λ. Thus, if we assume a value for λ, we can readily compute, in turn, α_i for i equal to $1, 2, \cdots, B-1$, by analytic or numerical computation depending on the numerical form assumed for $G_{\beta i}(\alpha_i)$. The resultant values of the α_i can then be substituted in (13) to calculate the value of J that corresponds to the assumed value of λ. It can be seen from (17) that, if we make the eminently reasonable assumption that $\partial G_{\beta i}(\alpha_i) / \partial \alpha_i$ is a monotone decreasing function of α_i, then each of the resultant α_i will be a monotone decreasing function of λ. It then follows from (11) that J is a monotone decreasing function of λ. Since $\mathscr{C}$ may be seen from (10) to be a monotone increasing function of J, it follows that $\mathscr{C}$ is a monotone decreasing function of λ.

The last conclusion is important. It means that we choose λ, determine the optimum α_i, calculate the minimum value of E, and then compute the value of $\mathscr{C}$ corresponding to our choice of λ. We know immediately from the monotonicity of $\mathscr{C}$ with λ whether we should increase or decrease λ in order to realize any particular $\mathscr{C}$.

Optimization of a generalized price performance function

In the previous section we have shown that each α_i appropriate to be used in the minimization of E is a monotone decreasing function of λ. Equation (16), however, shows that E is a monotone decreasing function of each of the α_i. It follows that $\min(E)$, the minimum value of E for a given λ, is a monotone increasing function of λ. Recalling that $\mathscr{C}$ is a monotone decreasing function of λ, we determine that $\min(E)$ is a monotone decreasing function of $\mathscr{C}$ for any fixed set of technologies. This last result also has intuitive appeal, since it indicates that the $\min(E)$ can be improved (made smaller) by increasing the hierarchy cost.

The following lemma is proved in Appendix A

$$\{0 < \mu\} \Rightarrow \{\min(E^\mu) = [\min(E)]^\mu\}. \tag{18}$$

This lemma is useful to us in framing a generalized price-performance criterion which utilizes directly the results of the previous section. The criterion is the function

$$P = E^\mu \mathscr{C}, \text{ where } \mu > 0. \tag{19}$$

Because E^μ is a monotone increasing function of E, we see that small values of P imply small values of both E and $\mathscr{C}$. We state, therefore, that the design that minimizes P is the optimum design.

If we assume "performance" to be proportional to the reciprocal of E and further assume μ to be unity, then P becomes the criterion frequently called the "price-to-performance ratio."

The purpose of the factor μ in (19) is to give the designer of a storage hierarchy a voice in the relative importance of E and $\mathscr{C}$ to his design. For example, if the storage were aimed at a market in which it was common to stretch the state-of-the-art, then a value of μ greater than unity would be appropriate. On the other hand, if cost were the primary market factor, a value of μ less than one would reflect this concern.

A proof of the following theorem is given in Appendix B

$$\min(P) = \min \{[\min(E)]^\mu \mathscr{C}\}. \tag{20}$$

From what has been said, it is clear that we can construct a smooth curve of $\min(E)$ vs $\mathscr{C}$. Then, from this curve and a choice of μ, it is easy to construct a smooth curve of $[\min(E)]^\mu$ vs $\mathscr{C}$. Indeed, this curve will exhibit $[\min(E)]^\mu$ as a monotone decreasing function of $\mathscr{C}$. From the lemma above, this is equivalent to a curve of $\min(E^\mu)$ vs $\mathscr{C}$. The theorem, therefore, states that the point on this last curve whose coordinates exhibit the smallest product is the optimal design point for the minimization of P.

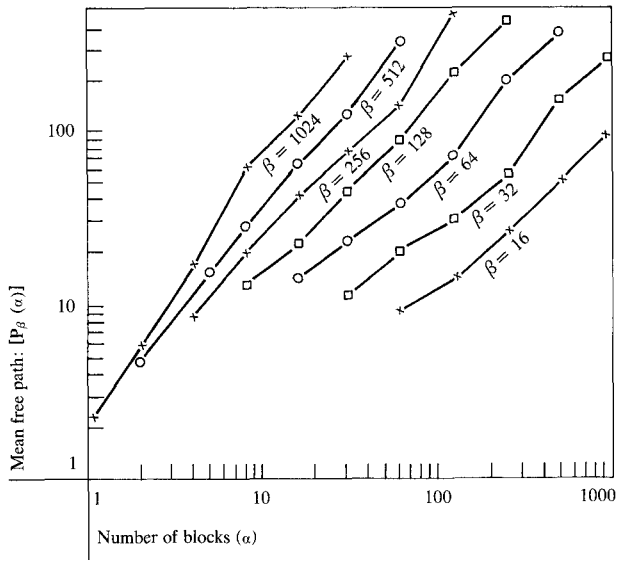

Figure 1 Graphs of some typical values of mean free path $[P_\beta(\alpha)]$ vs α with β as a parameter.

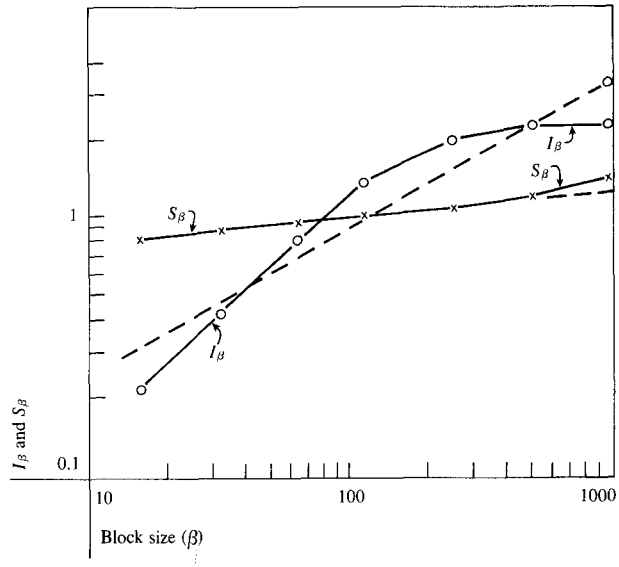

Figure 2 Some typical values of I_β and S_β vs block size (β). The dotted lines are the best straightline fits.

Optimization of block sizes

In the preceding sections it has been assumed that the $(B-1)$ block sizes of a B-level hierarchy are to be chosen arbitrarily by the designer before he employs our techniques. In the current section we present an extension of our methods which optimize both the β_i and the α_i simultaneously. The practical significance of the current section is to offer guidance to the designer who must choose values of β_i.

Let us return to the setting of the problem of minimization of access time for a fixed cost. We pointed out that the solution of the system of equations given by (17) is rendered relatively easy by observing that each of the $(B-1)$ equations of the form

$$\frac{\partial F}{\partial \alpha_i} = 0 = (v_i - v_{i+1}) \frac{\partial G_{\beta_i}(\alpha_i)}{\partial \alpha_i} + \lambda N k_i \beta_i \tag{21}$$

involves only two unknowns, namely λ and α_i.

In order to include the β_i in the optimization process, it is only necessary to take partial derivatives of F with respect to β_i and α_i. From Eqs. (11), (13), and (14) we see that F is a function of α_i and β_i. The result is

$$\frac{\partial F}{\partial \beta_i} = 0 = (v_i - v_{i+1}) \frac{\partial G_{\beta_i}(\alpha_i)}{\partial \beta_i} + \lambda N k_i \alpha_i. \tag{22}$$

We observe that for any given i, (21) and (22) represent two equations in the three unknowns λ, α_i, and β_i. If we assume a value of λ, then α_i and β_i may be readily determined by numerical computation on (21) and (22). This technique is particularly effective when (21) and (22) represent monotone functions. We next refer to some fairly extensive data for which this is the case.

Empirical equations for address sequence statistics

All of the foregoing will have practical application if we have at hand a continuous function representation of $G_{\beta_i}(\alpha_i)$ vs α_i with β_i as a parameter. In order that the method of the previous section be applicable, it is necessary that a continuous function representation of $G_{\beta_i}(\alpha_i)$ vs β_i with α_i as a parameter be available.

To this end, some empirical studies by the authors have shown that an adequate representation of $G_{\beta_i}(\alpha_i)$ can be provided by the following approximations.

$$P_\beta(\alpha) = \frac{1}{M_\beta(\alpha)} = \frac{1}{1 - G_\beta(\alpha)} = I_\beta \alpha^{S_\beta},$$

where

$$I_\beta = I_1 \beta^i, \text{ and}$$

$$S_\beta = S_1 \beta^s. \tag{23}$$

In Eq. (23) we have deliberately omitted any (level-indicating) subscripts from α and β. This is because the statistical data are always gathered as though they pertained to a two-level storage hierarchy. It is the property described by Eq. (6) that makes these two-level data immediately applicable to a B-level design.

In (23) we have introduced two new functions directly related to $G_\beta(\alpha)$. It is common to refer to $G_\beta(\alpha)$ as a "hit ratio", to $M_\beta(\alpha)$ as a "miss ratio", and to $P_\beta(\alpha)$ as a "mean free path".

A "typical" sample of statistical data is given in Table 1. These data are essentially the same as that reported by Mattson [9]. These same data are plotted in Fig. 1. "Best" values of I_1, i, S_1, and s have been obtained by doing a "least squares" fit on the three first degree equations that result from taking logarithms of each of the

Table 1 A typical sample of $G_\beta(\alpha)$ as a function of α and β.

α \ β	16	32	64	128	256	512	1024
1.0240E3	8.9375E-1	9.1458E-1	9.2700E-1	9.2334E-1	8.8314E-1	7.9138E-1	4.9450E-1
2.0480E3	9.3010E-1	9.4789E-1	9.5628E-1	9.5694E-1	9.4889E-1	9.1555E-1	8.2190E-1
4.0960E3	9.4809E-1	9.6815E-1	9.7272E-1	9.7735E-1	9.7648E-1	9.6544E-1	9.3805E-1
8.1920E3	9.8037E-1	9.8120E-1	9.8365E-1	9.8544E-1	9.8663E-1	9.8651E-1	9.8333E-1
1.6384E4	9.8910E-1	9.9313E-1	9.9485E-1	9.9620E-1	9.9242E-1	9.9175E-1	9.9139E-1
3.2768E4	9.9343E-1	9.9600E-1	9.9716E-1	9.9759E-1	9.9783E-1	9.9679E-1	9.9597E-1

three equations of (23). For the sample at hand, this procedure gave the values

$$\begin{cases} I_1 = 0.0698, \\ i = 0.553, \\ S_1 = 0.598, \\ s = 0.115. \end{cases} \tag{24}$$

The individual values of I_β and S_β for various values of β as well as the straight line approximations that correspond to (23) are shown in Fig. 2.

The forms of (23) wherein the statistical properties of an address trace are summarized in a quadruple, (I_1, i, S_1, s), have been found to be of quite wide applicability. Therefore, the form of (23) will be used in the next section in order to obtain explicit equations for the optimum α_j and β_j, where $j = 1, 2, 3, \cdots, B-1$.

Explicit equations for the optimization of the α_j and β_j

In this section we assume that the statistics of the address sequence are reasonably well represented by the equation

$$P_\beta(\alpha) = \frac{1}{1 - G_\beta(\alpha)} = I_1 \beta^i \alpha^{S_1 \beta^s}. \tag{25}$$

From the relationship between $P_\beta(\alpha)$ and $G_\beta(\alpha)$ we obtain

$$\frac{\partial G_\beta(\alpha)}{\partial \alpha} = \frac{1}{P_\beta^{\,2}(\alpha)} \frac{\partial P_\beta(\alpha)}{\partial \alpha}, \tag{26}$$

and

$$\frac{\partial G_\beta(\alpha)}{\partial \beta} = \frac{1}{P_\beta^{\,2}(\alpha)} \frac{\partial P_\beta(\alpha)}{\partial \beta}. \tag{27}$$

Substitution of (26) and (27) in (21) and (22) gives

$$\frac{\partial F}{\partial \alpha_j} = 0 = (v_j - v_{j+1}) \frac{1}{P_{\beta_j}^{\,2}(\alpha_j)} \frac{\partial P_{\beta_j}(\alpha_j)}{\partial \alpha_j} + \lambda N k_j \beta_j \tag{28}$$

and

$$\frac{\partial F}{\partial \beta_j} = 0 = (v_j - v_{j+1}) \frac{1}{P_{\beta_j}^{\,2}(\alpha_j)} \frac{\partial P_{\beta_j}(\alpha_j)}{\partial \beta_j} + \lambda N \kappa_j \alpha_j. \tag{29}$$

To simplify our notation, we set

$$w_j = v_{j+1} - v_j. \tag{30}$$

Comparison of (28) and (29) yields

$$\alpha_j \frac{\partial P_{\beta_j}(\alpha_j)}{\partial \alpha_j} = \beta_j \frac{\partial P_{\beta_j}(\alpha_j)}{\partial \beta_j}. \tag{31}$$

Next we need the partial derivatives of $P_{\beta_j}(\alpha_j)$. From (25) we get

$$\alpha_j \frac{\partial P_{\beta_j}(\alpha_j)}{\partial \alpha_j} = I_1 \beta_j^{\,i} \alpha_j^{\,S_1 \beta_j^s} S_1 \beta_j^{\,s} \tag{32}$$

and

$$\beta_j \frac{\partial P_{\beta_j}(\alpha_j)}{\partial \beta_j} = I_1 \beta_j^{\,i} \alpha_j^{\,S_1 \beta_j^s} [(\ln \alpha_j) S_1 s \beta_j^{\,s} + i]. \tag{33}$$

Use of (32) and (33) in (31) and cancellation of common factors gives us

$$S_1 (\beta_j)^s = (\ln \alpha_j) S_1 s \beta_j^{\,s} + i. \tag{34}$$

When (34) is solved for $\ln \alpha_j$ the result is

$$\ln \alpha_j = s^{-1} [1 - i / S_1 \beta_j^{\,s}]. \tag{35}$$

Several properties of (35) are worthy of note. 1) The statistical parameter i has to be smaller than the statistical parameter S_β for the existence of an optimal solution. In our studies of address sequences, this condition has always been satisfied. 2) By allowing β_j to become arbitrarily large we can establish an upper bound on α_j by the relation

$$\ln \alpha_j < 1/s. \tag{36}$$

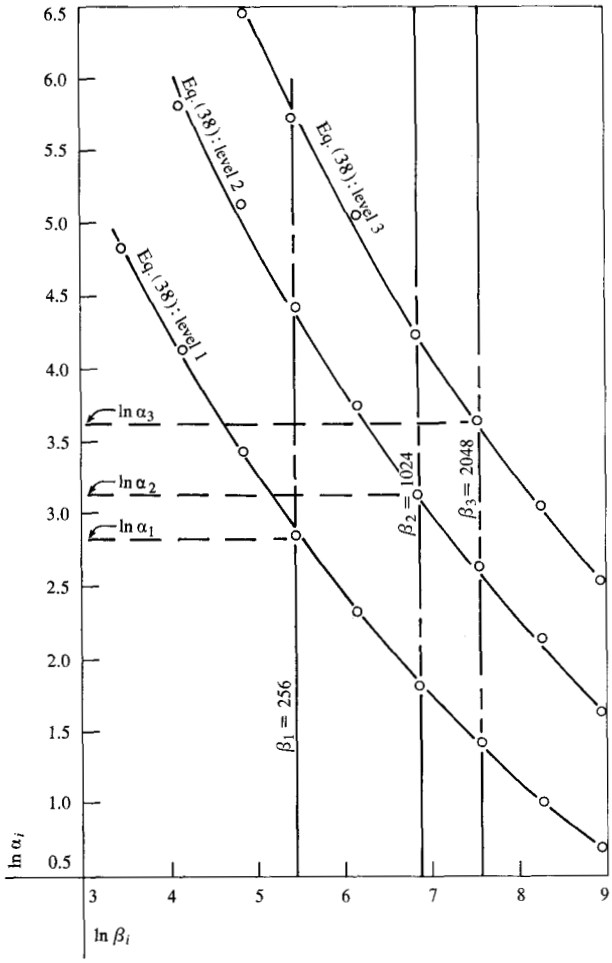

Figure 3 Graphical example of access time minimization.

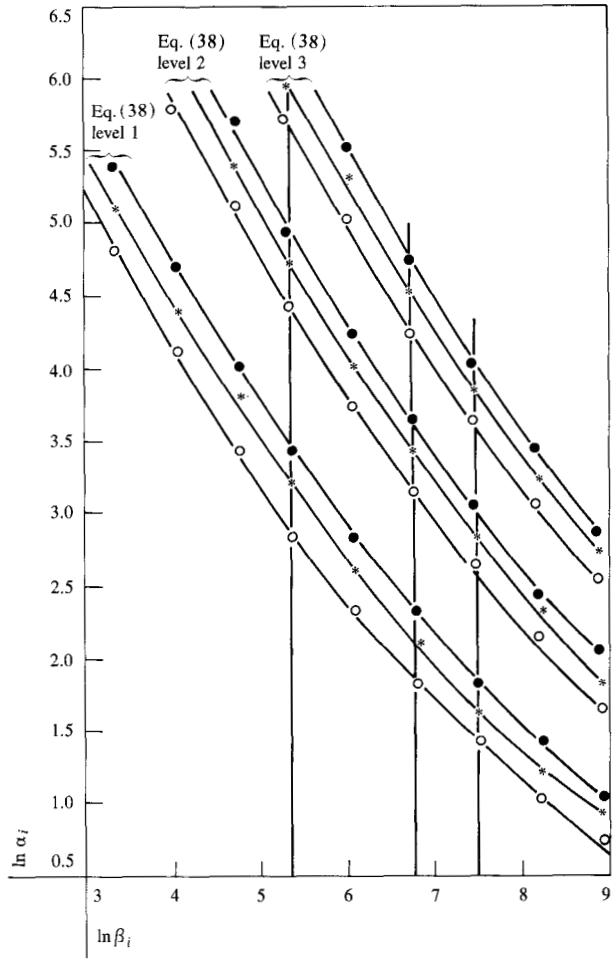

Figure 4 Graphical example of optimization. For the open-circle curves, $\lambda N = 5$; for the asterisk curves, $\lambda N = 0.05$; and for the solid-circle curves $\lambda N = 0.0001$.

3) If we regard α_j and β_j as the variables of (35), then none of the parameters of (35) depend on j. This means that (35) is applicable simultaneously to all levels of a B-level storage hierarchy. 4) We note that λ does not appear in (35). This means that (35) is applicable simultaneously to all values of the discretionary cost parameter.

At this point we recall that our original intent in this section was to solve simultaneously (28) and (29) for an assumed value of λ. Because (35) was obtained by combining (28) and (29), it may substitute for one of them in the final system of equations. This we choose to do because of the inherent simplicity and generality exhibited by (35). The remaining task is to substitute (25) into either (28) or (29) and carry out the required algebra. When (25) is substituted into (28) we find, in view of (30), that

$$w_j I_1 \beta_j^{\,i} S_1 \beta_j^{\,s} \alpha_j^{\,S_1(\beta_j)^s - 1} = \lambda N k_j \beta_j I_1^{\,2} \beta_j^{\,2i} \alpha_j^{\,2S_1(\beta_j)^s}. \qquad (37)$$

Equation (37) may be solved for α_j, This algebraic manipulation yields

$$\ln \alpha_j = \frac{\ln (w_j S_1 / \lambda N k_j I_1) + (s - i - 1) \ln \beta_j}{S_1 \beta_j^{\,s} + 1}. \qquad (38)$$

Thus optimum values of α_j and β_j may be determined by simultaneous solution of (35) and (38). Numerical procedures are entirely satisfactory for this purpose, particularly since both equations describe monotone relations between α_j and β_j. This is true because the factor $(s - i - 1)$ has always been negative for any address sequences we have investigated. That condition is sufficient, but not necessary, to make $\ln \alpha_j$ a monotone decreasing function of $\ln \beta_j$ in (38). In (35) it is clear that $\ln \alpha_j$ is a monotone increasing function of $\ln \beta_j$.

Numerical examples

In this section we assume that an address sequence has been processed by an LRU stack algorithm program to produce the statistical parameters:

$$\begin{aligned} I_1 &= 0.615 \\ i &= 0.162 \\ S_1 &= 0.218 \\ s &= 0.246. \end{aligned} \tag{39}$$

• *Example for the minimization section* Assume we want to determine the optimum values of α_1, α_2, α_3 in a four-level storage hierarchy after arbitrarily setting

$$\begin{aligned} \beta_1 &= 256 \\ \beta_2 &= 1024 \\ \beta_3 &= 2048 \\ \lambda N &= 5. \end{aligned} \tag{40}$$

The solution, obtained by use of (38) for the values just chosen, is shown graphically in Fig. 3, where we have made appropriate assumptions of the values of v_1, v_2, v_3, v_4, k_1, k_2 and k_3. The optimal values of α_1, α_2, α_3 can then be used to calculate, using (10) and (11), the discretionary cost per unit, J, and the total cost, $\mathscr{C}$, which correspond to our choice of λN equal to five.

• *Example for the optimization section* Assume we wish to optimize the generalized price performance function for the previous example. First it will be necessary to calculate the optimal α_1, α_2, and α_3 for several values of λN. This is illustrated in Fig. 4 for three such values. A value of $\min(E)$ and $\mathscr{C}$ can be computed by use of (10), (11) and (16), once we have chosen a value of N. for each choice of λN.

Next a value of μ is selected. Then the results of Fig. 4 will appear as three points on a graph such as the one in Fig. 5. From Fig. 5 the optimal operating point may be determined.

• *Example for the optimization of block sizes section* The technique described for optimizing β_j can be applied in connection with either the problem of minimizing the access time or optimizing the generalized price-performance function. In this example we apply the technique for optimizing the β_j to the problem of minimization of access time for a fixed cost.

Assume we want a four-level storage hierarchy. Once we have obtained values for v_1, v_2, v_3, v_4, k_1, k_2 and k_3 we may use (38) to plot $\ln \alpha_j$ vs $\ln \beta_j$ for various values of λN. This is done for three values of λN in Fig. 6. In addition, our solution must satisfy (35). Therefore, if we also plot $\ln \alpha_j$ vs $\ln \beta_j$ from (35), our solutions lie at the intersections of this plot with the three graphs of (38) for any fixed λN. Naturally the actual plotting of (35) and (38) is not an essential part of the solution process.

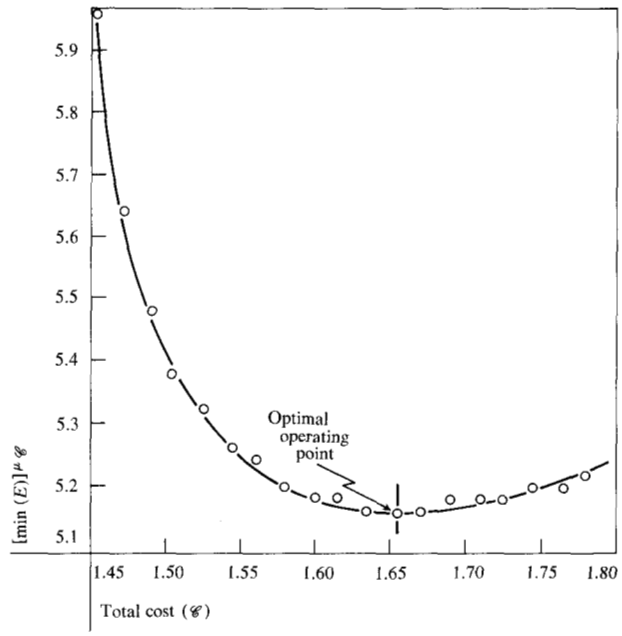

Figure 5 Graphical example of single-product optimization. Each point corresponds to some value of λN.

In fact, because of the monotonicity properties discussed previously, it is quite easy to compute (35) and (38) interatively until any desired level of precision is reached.

Summary

We have described optimization methods to be used in the design of storage hierarchies. The mathematical tool employed is the method of Lagrange multipliers. We have assumed as inputs a finite (and probably small) set of candidate technologies, each characterized by an access time and two cost parameters. The designer is asked to play an active role in the design process through his choice of certain parameters which reflect market estimates.

The block sizes to be employed may be selected in advance of the optimization if the designer so chooses. Alternatively, the techniques have been extended to permit the process to optimize block sizes if desired.

It should also be emphasized that we rely on the typicality of the statistical summaries of address traces which are available. Verification of this typicality is a potentially fruitful area for future research. The few signs and portents which are available to us are encouraging.

References

1. D. H. Gibson, "Considerations in Block-Oriented Systems Design," AFIPS Conference Proceedings, Spring Joint Computer Conference **30**, Academic Press, New York, New York, 75-80 (1967).
2. C. J. Conti, D. H. Gibson, S. H. Pitkowsky, J. S. Liptay, A. Padegs, "Structural Aspects of the System/360 Model 85," *IBM Syst. J.* **7**, 2 (1968).

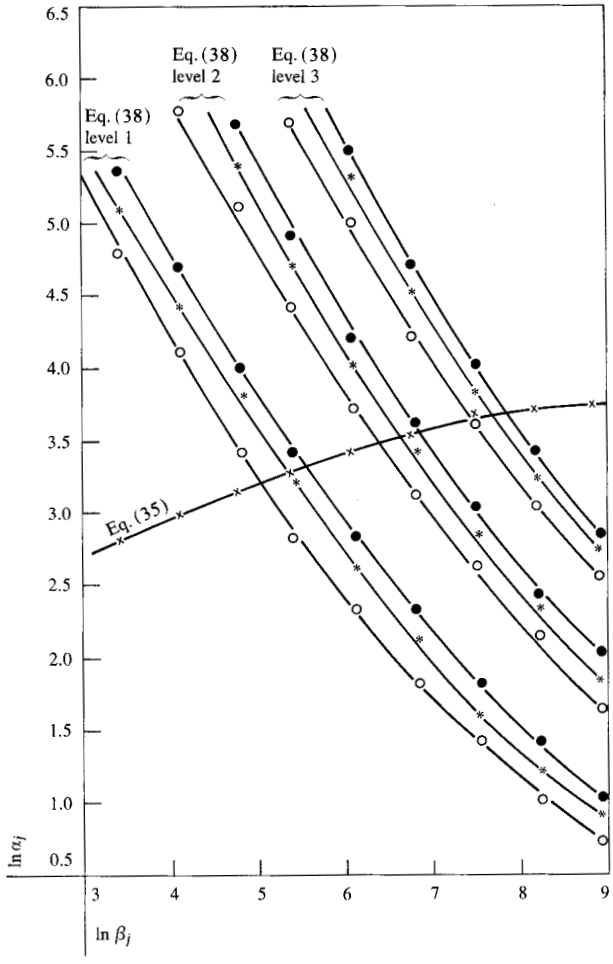

Figure 6 Graphical example of the optimization of block sizes. For the open circle curves, $\lambda N = 5$; for the asterisk curves, $\lambda N = 0.05$; and for the closed circle curves, $\lambda N = 0.0001$.

3. L. A. Belady, "A Study of Replacement Algorithms for Virtual Storage Computers," *IBM Syst. J.* **5**, 78 (1966).
4. R. L. Mattson, J. Gecsei, D. R. Slutz, I. L. Traiger, "Evaluation Techniques for Storage Hierarchies," *IBM Syst. J.* **9**, 78-117 (1970).
5. L. A. Belady and F. P. Palermo, "On-line Measurement of Paging Behavior by the Multivalued MIN Algorithm," *IBM J. Res. Develop.* **18**, 2 (1974).
6. J. E. MacDonald, "Prediction of Three-Level Storage Performance Through the Use of Two-Level Data," IBM Internal Report AR-000573-00-POK (March 19, 1968).
7. D. R. Slutz and I. L. Traiger, "Determination of Hit Ratios for a Class of Staging Hierarchies," IBM Internal Report RJ 1044, (May, 1972)
8. J. Gecsei, "Replacement Algorithms for Staging Hierarchies," *IBM J. Res. Develop.* **18**, 316 (1974).
9. R. L. Mattson, "Evaluation of Multilevel Memories," *IEEE Trans. Magnetics* **MAG-7**, 814 (1971).

Appendix A

Lemma $\{0 < \mu\} \Rightarrow \{\min(E^\mu) = [\min(E)]^\mu\}$

Proof

By definition,

$$\min(E) \leq E. \tag{A1}$$

Since the logarithm function is monotone increasing,

$$\ln[\min(E)] \leq \ln(E). \tag{A2}$$

Multiplying (A2) by a positive constant yields

$$\mu \ln[\min(E)] \leq \mu \ln(E) \tag{A3}$$

or

$$\ln\{[\min(E)]^\mu\} \leq \ln(E^\mu). \tag{A4}$$

Since the logarithm function is monotone increasing,

$$[\min(E)]^\mu \leq E^\mu. \tag{A5}$$

But (A5) is equivalent to

$$[\min(E)]^\mu = \min(E^\mu). \tag{A6}$$

Appendix B

Theorem $\min(E^\mu \mathscr{C}) = \min\{[\min(E)]^\mu \mathscr{C}\}$

Proof

For any fixed $\mathscr{C}$, consider the $\min(E)$ corresponding to that $\mathscr{C}$ and also any *other* E also realizable for that $\mathscr{C}$. From the previous lemma, we know that

$$[\min(E)]^\mu < E^\mu. \tag{B1}$$

Since $\mathscr{C}$ is positive, (B1) leads to

$$[\min(E)]^\mu \mathscr{C} < E^\mu \mathscr{C}. \tag{B2}$$

But (B2) immediately implies, as $\mathscr{C}$ is allowed to vary, that

$$\min\{[\min(E)]^\mu \mathscr{C}\} < \min(E^\mu \mathscr{C}). \tag{B3}$$

Thus from (B3) we see that no realizable pair $(E, \mathscr{C})$, where E is other than $\min(E)$ for that $\mathscr{C}$, can yield the minimum value of $(E^\mu \mathscr{C})$. But, a minimum of $E^\mu \mathscr{C}$ must exist since it is restricted to positive values. Thus, we see that

$$\min(E^\mu \mathscr{C}) = \min\{[\min(E)]^\mu \mathscr{C}\}. \tag{B4}$$

Received March 15, 1974; revised May 27, 1974

The authors are located at the IBM System Products Division Laboratory, Poughkeepsie, New York 12602.