An Analytical Model for Designing Memory Hierarchies

Bruce L. Jacob, *Student Member, IEEE*, Peter M. Chen, *Member, IEEE*, Seth R. Silverman, and Trevor N. Mudge, *Fellow, IEEE*

Abstract— ***Memory hierarchies have long been studied by many means: system building, trace-driven simulation, and mathematical analysis. Yet little help is available for the system designer wishing to quickly size the different levels in a memory hierarchy to a first-order approximation. In this paper, we present a simple analysis for providing this practical help and some unexpected results and intuition that come out of the analysis. By applying a specific, parameterized model of workload locality, we are able to derive a closed-form solution for the optimal size of each hierarchy level. We verify the accuracy of this solution against exhaustive simulation with two case studies: a three-level I/O storage hierarchy and a three-level processor-cache hierarchy. In all but one case, the configuration recommended by the model performs within 5% of optimal. One result of our analysis is that the first place to spend money is the cheapest (rather than the fastest) cache level, particularly with small system budgets. Another is that money spent on an n-level hierarchy is spent in a fixed proportion until another level is added.***

Keywords— ***cache, memory, and storage hierarchies; trace-driven simulations; optimization of cache configurations.***

I. Introduction

FAST memory and storage systems are vital to achieving good system performance, as CPU speeds increase faster than memory and disk speeds. Almost all systems use caching throughout the disk, memory, and processor subsystems to improve the average time to access data, but the widening gap between storage technologies makes it easy to lose significant performance through poor cache sizing. Unfortunately, little practical help exists for system designers and administrators seeking to optimize their cache hierarchies. Exhaustive simulation takes far too long, particularly as hierarchies become more complex [16]; trial and error on running systems is usually impossible; and prior mathematical analyses have stopped short of providing much-needed, intuitive insight into cache sizing [10] or have assumed the availability of memory technologies with arbitrary speeds and costs [20].

In this paper, we analyze the performance of a general, n-level memory hierarchy using a parameterized workload characterization. Our intent is to derive a simple and intuitive method for quickly generating first-order approximations of optimal hierarchy organizations. To this end, we make several simplifications and compare the results to those obtained by more accurate simulation techniques.

B. Jacob, P. Chen, and T. Mudge are with the Advanced Computer Architecture Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122. e-mail: blj@eecs.umich.edu; pmchen@eecs.umich.edu; tnm@eecs.umich.edu.

S. Silverman is with Chelmsford Systems Software Lab, Hewlett-Packard, Chelmsford, MA 01824. e-mail: seth@apollo.hp.com

The result is a simple, closed-form solution for the size of each level of the hierarchy as a function of the workload locality, the speed and cost of available technologies, and the amount of money available to spend on the system. We validate the model with trace-driven simulations of a three-level processor-cache hierarchy and a three-level storage hierarchy. The cache sizes recommended by our model perform close to the optimal performance as determined by exhaustive simulation.

With little money to spend on the hierarchy, the model recommends spending it all on the cheapest, slowest storage technology rather than the fastest. This is contrary to conventional wisdom, which focuses on satisfying as many references as possible in the fastest cache level, such as the L1 cache for processors or the file cache for storage systems. Interestingly, it *does* reflect what has happened in the PC market, where processor caches have been among the last levels of the memory hierarchy to be added. We discuss why initial money is best spent on slow technologies and hope that this paper helps to improve the intuition of those configuring caches.

The model also suggests that every dollar spent on an n-level hierarchy be done in a fixed proportion; every dollar should increase the size of every level in the hierarchy, not just one. This is described more in the discussion of the analysis (Section IV).

II. Previous Work

Countless articles have been written about memory hierarchies ([17, 18] provide excellent overviews of CPU and disk caches), generally focusing on a two-level hierarchy [9]. Most papers in recent years have used trace-driven simulation to investigate such aspects of cache performance as multiprocessor cache coherence and replacement strategies. Trace-driven studies are valuable for understanding cache behavior on specific workloads, but they are not easily applied to other workloads [17].

Unlike traces, mathematical analysis lends itself well to understanding cache behavior on general workloads, though such generality usually leads to less accurate results. Many researchers have analyzed memory hierarchies in the past. Chow showed that the optimum number of cache levels scales with the logarithm of the capacity of the cache hierarchy [3, 4]. Garcia-Molina and Rege demonstrated that it is often better to have more of a slower device than less of a faster device [7, 15]. Welch showed that the optimal speed of each level should be proportional to the amount of time spent servicing requests at that level [20].

These studies have had two shortcomings: 1) they assume the availability of memory technologies with arbitrary speeds and costs, and 2) they do not apply their analyses to a specific model of workload locality. Being able to create and use technologies on a continuum of characteristics is convenient for analysis but makes the analysis difficult for system builders to use. Failing to apply a specific model of workload locality makes it impossible to provide an easily used, closed-form solution for the optimal cache configuration [10], and so results from these papers have contained dependencies on the cache configuration—the number of levels, or the sizes and hit rates of the levels.

We provide three main contributions beyond those of previous analyses:

- We extend previous analyses by applying our general solution to a specific model of workload locality. We are thus able to provide a closed-form solution for the optimal sizes of each workload level as a function of two locality parameters and the device speeds and costs.
- We discuss what the resulting equations mean intuitively to system designers in terms of their general locality and available device characteristics.
- We verify our model's accuracy against a detailed simulation of two memory hierarchies: 1) a storage hierarchy consisting of RAM, disk, and tape, and 2) a three-level processor-cache hierarchy consisting of an on-chip cache (L1), an off-chip cache (L2), and main memory. The performance of the cache configuration recommended by our model is almost always within 5% of the best performance obtained from exhaustive simulation.

III. Analysis

In this section, we derive an analytic solution for the size of each level in a cache hierarchy. The analysis starts with a pre-specified set of technologies, though the resulting equations may be used easily to choose an optimal subset of technologies from the universe of technologies.

A. The System Model

A.1 Notation

In the following analysis, a hierarchy will consist of n cache levels, numbered 1 through n, with the backing store considered to be level $n+1$. Fig 1 shows a typical hierarchy. Note that the hierarchy to be analyzed can start and end anywhere; it need not begin immediately beneath the CPU. For instance, our simulated I/O hierarchy does not begin at the on-chip cache level, but several layers below at main memory, and could just have easily started with the disk level.

Each technology level i is described by the following three parameters:

- t_i is the *average access time* of the technology,
- c_i is the *cost* of the technology in dollars per byte, and
- s_i is the *size* of the level, in bytes.

The only unknown variables are s_i; the values for all c_i and t_i are known constants. We assume that the choice of

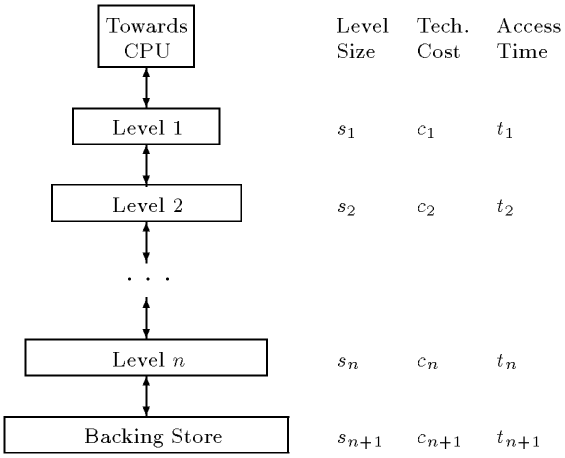


Fig. 1. A general cache hierarchy. The bottommost level (backing store) is not considered part of the hierarchy; it is the storage level being cached. The highest level shown is labeled "Towards CPU" instead of just "CPU" because the analytical hierarchy can begin at any point, not necessarily immediately below the CPU. We assume inclusion; that is, all data at each level is contained in the level immediately beneath it.

hierarchy technologies is made first. We also assume that any hierarchy will be a realistic one in that level i is always faster and more expensive than level $i+1$.

The total cost of the cache hierarchy system is the sum of the costs of cache levels 1 through n. For the sake of simplicity, we will assume a linear cost model.[1] The system budget, B, is given by:

$$B = \sum_{i=1}^{n} c_i s_i \tag{1}$$

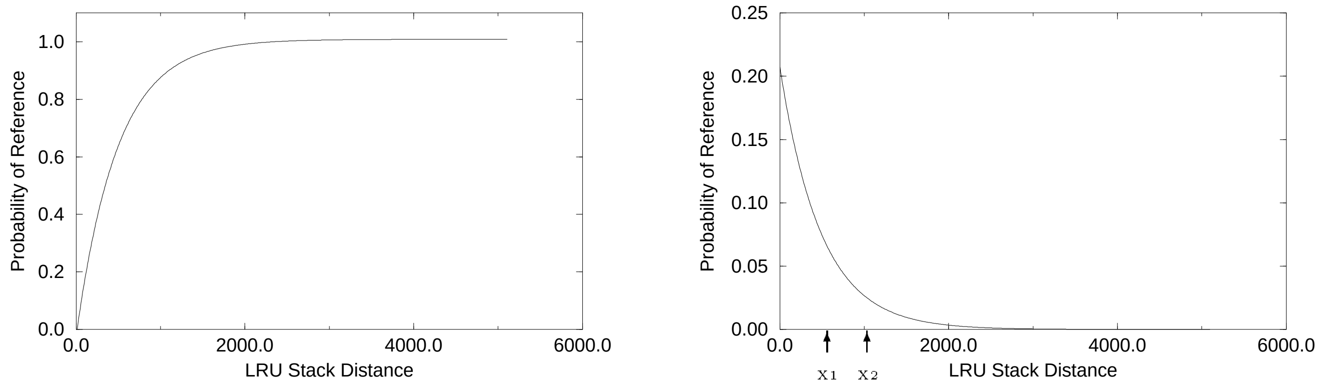
A.2 Stack Distance Curves

Our analysis for cache performance depends on a mathematical description of workload locality and models a fully associative cache and a hierarchy that maintains inclusion. The effects of these assumptions are discussed at the end of this section. To compute the probability of a reference hitting at a cache level, we use *stack distance curves*, measurements taken directly from address streams [5].

The stack distance curves describe how many unique bytes of data separate two references to the same item. For instance, consider a reference stream $A_1C_1B_1B_2C_2A_2B_3$, where A_n refers to the nth time that datum A is referenced. Then there are two unique data touched between A_1 and A_2, which are C and B, and the *stack distance* between A_1 and A_2 is 3. There is one unique datum touched between C_1 and C_2, which is B, and the stack distance between C_1 and C_2 is 2; there are no data touched between B_1 and B_2, and the stack distance is 1; there are two data touched between B_2 and B_3 (C and A), so the stack distance is 3.

After normalizing, we can plot this distribution of stack distances as a cumulative probability function and a prob-

[1] A non-linear model can be used but would tend to obscure the discussion, and as demonstrated in Section V, the linear model gives good results.



(a) Cumulative probability function

(b) Probability density function

Fig. 2. The stack distance curves. These functions describe the degree of locality in a workload. They show the distribution of how many unique bytes the workload touches between references to the same item. The cumulative probability function is a plot of hit rate versus cache size, for an LRU-managed cache. We use these graphs to compute the probability that a workload's reference hits in a given cache size.

ability density function. The cumulative probability function shows, at each x value, how many references were made at a stack distance of x or less; how many references were separated from the last reference to the same data by less than x unique pieces of data (Fig 2a). This relates well to an LRU-managed, fully-associative cache; a cache of size 1 will catch all references of stack distance 1 (those references with 0 intervening data), a cache of size 2 will catch all references at stack distance 1 and 2, and so on. The cumulative probability function $P(x)$ indicates what proportion of accesses are to data at a stack distance of x or smaller. An LRU-managed, fully-associative cache of size x would therefore have a hit ratio of $P(x)$. The probability density function is the derivative of the cumulative probability function; $p(x)$ describes the frequency of references at exactly stack distance x (Fig 2b).

Fig 2 plots the expected shape of these curves. We primarily use the density function. As we expect most workloads to have some locality, Fig 2b graphs more pairs of accesses being separated by few data than by more data. The area under the density function between X1 and X2 describes the probability of an access being separated from its last reference by more than X1 and less than X2 pieces of unique data. With fully associative caches, this is exactly the probability of a reference missing in a cache of size X1 and hitting in the next cache level of size X2. With caches that are not fully associative, conflict misses can occur, where the size of the cache is large enough to capture a reference stream but the placement policy causes additional misses. For the same reason, caches that are not fully associative may yield hits on data that is old but, due to the placement policy, has not yet been displaced.

Fig 3 illustrates the difference between fully and non-fully associative caches. With a fully associative cache, we can draw an exact line on the probability density curve separating cache hits from cache misses based on capacity and inter-reference distance. With a non-fully associative cache, we may only specify a probability distribution of hits on the probability density curve. References closer to the y-axis are more likely to be hits than references farther from the y-axis.

This paper conducts a first-order analysis using fully associative caches. Section V verifies that this analysis is also accurate for non-fully associative caches.

A.3 Average Time per Reference

Chow and Welch use as a performance measure the average time per memory reference, and model it with the following equation:

$$T = \sum_i P_i t_i$$

Each t_i is the time to access level i in the hierarchy, and each P_i is the probability that level i will be accessed. The hierarchy maintains inclusion and the probabilities do not necessarily sum to one; the topmost level is accessed on every single reference (hit or miss), so P_1 is 1.

$$1 = P_1 \geq P_2 \geq P_3 \geq \cdots \geq P_n$$

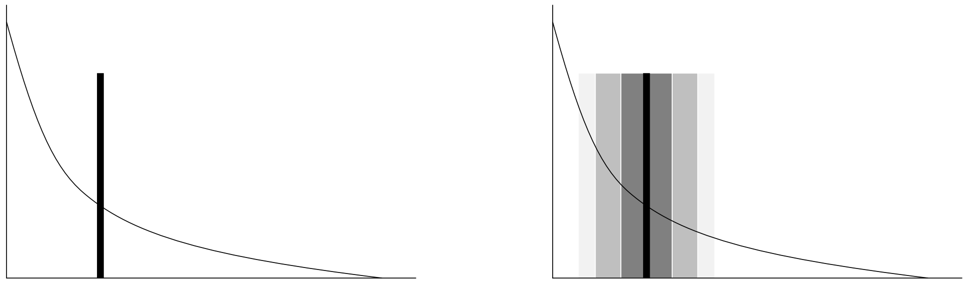
In our analysis, the stack distance curves are used to compute P_i. The probability of accessing level i is equal to the probability that the reference will miss in all the levels above it:

$$\int_{s_{i-1}}^{\infty} p(x)dx$$

where $p(x)$ is the probability density function. The average system time spent per reference accessing level i is thus the time to reference level i scaled by this probability:

$$t_i \int_{s_{i-1}}^{\infty} p(x)dx$$

and the total system time spent per reference is the sum of the times across all levels in the hierarchy:



(a) Fully associative (b) Direct-mapped

Fig. 3. Fully associative cache model vs. direct-mapped cache model. In the fully associative model, it is clear what cache accesses are hits and what accesses are misses. The size of the cache is the clear dividing line separating the two. The percentage of hits is therefore the area under the curve—a simple integral over the probability density function. The direct-mapped case introduces an element of chance, as a particular line in the cache could be thrown out on the next cache reference, or it could last in the cache for an unusually long period of time, all depending on the particular address reference stream. It is much more difficult to draw a solid line between the hits and misses; the line is blurred, as illustrated in (b). However, it can be modeled probabilistically, where we know with high probability that the references at the extremes (those near in time and those distant in time) will be hits and misses, respectively. However, in the middle ranges, the chances of guessing correctly grow worse, depicted by darkening shades of grey.

$$T = t_1 + t_2 \int_{s_1}^{\infty} p(x)dx + \cdots + t_{n+1} \int_{s_n}^{\infty} p(x)dx \qquad (2)$$

The size of the bottom storage level $n+1$ does not appear in the equation, since this level is assumed to contain all data, so s_{n+1} is for all intents infinite. The time to reference this level does appear, scaled by the miss rate of the lowest cache level. As we expect, backing store is only referenced on misses to the lowest cache level.

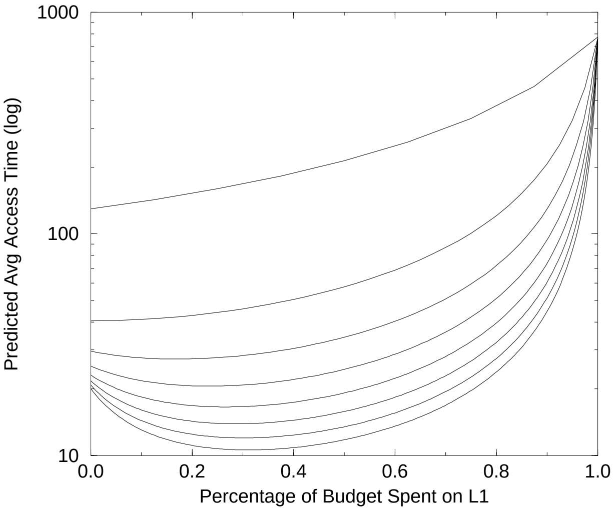


Fig. 4. Behavior and sensitivity of the access time function. This is an example of the access time for a two-level hierarchy, depending on what proportion of money is spent on which level in the hierarchy. The curves represent constant budget values.

Fig 4 illustrates the behavior of the access time function T as affected by the hierarchy organization. The graph represents a two-level hierarchy and the x-axis shows the percentage of the budget spent on the top level of the hierarchy. Towards the left represents more money spent on the L2 cache, toward the right represents more money spent on the L1 cache. The curves depict constant budget values.

B. Hierarchy Optimization

Our goal is to specify the memory hierarchy with the fastest average access time (T). Specifically, we solve for the size of each hierarchy level (s_i), given the access time (t_i) and unit cost (c_i) of each technology; parameters describing workload locality; and the total system budget. Our solution proceeds through the following steps: 1) use Lagrange multipliers [2] to get a general solution without constraining the sizes to be non-negative; 2) apply a specific, parameterized model of workload locality to derive a closed-form solution, again allowing negative sizes; 3) refine the solution to account for the additional constraint that all sizes be non-negative.

B.1 Calculus of Variations

First, we put the cost function B into an appropriate form and obtain the constraint function g:

$$g = (c_1 s_1 + c_2 s_2 + \cdots + c_n s_n) - B = 0 \qquad (3)$$

T is a function of the n variables $s_1, \ldots, s_n$, as is the cost function B and its associated cost constraint g. At the point where T is minimized, we know that:

$$\nabla T = \lambda \nabla g$$

where λ is the Lagrange multiplier. Combining the gradients of Eqs 2 and 3, we get

$$-t_{i+1} p(s_i) = \lambda c_i, \text{ for } 1 \leq i \leq n$$

This form gives us an interesting ratio which we will call the cost-performance ratio Ψ_{ij}:

$$\Psi_{ij} \equiv \frac{p(s_i)}{p(s_j)} = \frac{c_i t_{j+1}}{c_j t_{i+1}} \qquad (4)$$

The behavior of Ψ_{ij} is described later; for now, it is only necessary to note that $\Psi_{ii} = 1$. Eq 1 (total system cost) and Eq 4 yield n independent equations, so solving for the n variables $s_1, \ldots, s_n$ is straightforward as long as $p(x)$ is invertible.

B.2 Modeling Program Behavior

We wish to replace $p(x)$ in Eq 4 with a specific function. Smith, Stone and others have noted that a cache's miss rate can be modeled as a one-term polynomial function of its size, of the form

$$\beta x^{\alpha}$$

where α and β are constants with α less than zero [17, 19]. This follows from the 30% Rule[2]. It is also consistent with our workload traces in Section V. Thus we assume polynomial forms for $P(x)$ and $p(x)$ in this paper; we have also used an exponential form with similar results [8].

It is easiest to start at the cumulative probability graph $P(x)$. As mentioned before, $P(x)$ is related to a cache's hit ratio—an LRU-managed cache of size x would have a hit rate of $P(x)$, given the input stream that generated $P(x)$. It is necessary that $P(x)$ be 0 at 0 and 1 at infinity, and ideally would have a form similar to

$$1 - \beta x^{\alpha}$$

For simplicity and convenience we make the following changes:

- we would like the exponent to be positive, so we move x^{α} to the denominator,
- the function blows up at 0, so we replace x with $x+1$,
- we want the derivative $p(x)$ to have a simple exponent, so we replace α with $\alpha - 1$, and
- the function defined would *not* have the value 0 at 0, and also would not be unitless, so we scale the value of x directly by β.

This gives us the following forms for $P(x)$ and its differential $p(x)$.

$$P(x) = 1 - \frac{1}{(\frac{x}{\beta}+1)^{\alpha-1}}, p(x) = \frac{C}{(x+\beta)^{\alpha}} \tag{5}$$

where the constant C is $(\beta^{\alpha-1})(\alpha-1)$. Workload locality improves with decreasing β or increasing α.

Given this form for $p(x)$, we can now find the sizes for the optimal hierarchy. First, we combine Eqs 4 and 5 and arrive at the following (ignoring negative roots because all values are positive):

$$\frac{s_j+\beta}{s_i+\beta} = \Psi_{ij}^{1/\alpha} \tag{6}$$

Eq 6 yields $n-1$ independent equations with n unknowns $s_1, \ldots, s_n$:

$$\frac{s_2+\beta}{s_1+\beta} = \Psi_{1,2}^{1/\alpha}, \frac{s_3+\beta}{s_1+\beta} = \Psi_{1,3}^{1/\alpha}, \cdots, \frac{s_n+\beta}{s_1+\beta} = \Psi_{1,n}^{1/\alpha}$$

which yields:

$$\frac{c_2 s_2 + c_2\beta}{s_1+\beta} = c_2\Psi_{1,2}^{1/\alpha}, \frac{c_3 s_3 + c_3\beta}{s_1+\beta} = c_3\Psi_{1,3}^{1/\alpha}, \cdots,$$

$$\frac{c_n s_n + c_n\beta}{s_1+\beta} = c_n\Psi_{1,n}^{1/\alpha}$$

and since this is valid over all i and j we have $n-1$ independent equations of the form:

$$\sum_{j\neq i} \frac{c_j s_j + c_j\beta}{s_i+\beta} = \sum_{j\neq i} c_j \Psi_{ij}^{1/\alpha}$$

Together with Eq 1 (total system cost), we have n equations and n unknowns and can solve for each s_i in the following manner (using s_1 as an example):

$$\frac{\sum_{i=2}^{n} c_i s_i + c_i\beta}{s_1+\beta} = \sum_{i=2}^{n} c_i \Psi_{1,i}^{1/\alpha}$$

$$\frac{B - c_1 s_1 + \sum_{i=2}^{n} c_i\beta}{\sum_{i=2}^{n} c_i \Psi_{1,i}^{1/\alpha}} = s_1 + \beta$$

$$s_1 = \left(\frac{\sum_{i=2}^{n} c_i \Psi_{1,i}^{1/\alpha}}{c_1 + \sum_{i=2}^{n} c_i \Psi_{1,i}^{1/\alpha}}\right)\left(\frac{B + \sum_{i=2}^{n} c_i\beta}{\sum_{i=2}^{n} c_i \Psi_{1,i}^{1/\alpha}} - \beta\right)$$

$$s_1 = \frac{B + \sum_{i=2}^{n} c_i\beta(1-\Psi_{1,i}^{1/\alpha})}{c_1 + \sum_{i=2}^{n} c_i \Psi_{1,i}^{1/\alpha}}$$

and, since $\Psi_{ii} = 1$:

$$s_1 = \frac{B + \sum_{i=1}^{n} c_i\beta(1-\Psi_{1,i}^{1/\alpha})}{\sum_{i=1}^{n} c_i \Psi_{1,i}^{1/\alpha}}$$

A similar sequence of steps can be performed for each s_i in turn, yielding a general form for s_i:

$$s_i = \frac{B + \sum_{j=1}^{n} c_j\beta(1-\Psi_{ij}^{1/\alpha})}{\sum_{j=1}^{n} c_j \Psi_{ij}^{1/\alpha}} \tag{7}$$

Note that all values are constants except for B, the system budget; Eq 7 says the size of each hierarchy level increases linearly with the amount of money to spend on the system. The costs and access times (c_i and t_i) are constants derived from the chosen technologies. Note that the denominator is different for each level in the hierarchy; therefore the rate of increase is different for each level. The y-intercept is also different for every level in the hierarchy. The shape of the curves is shown in Fig 5.

[2] The 30% Rule, first suggested by Smith [17], is the rule of thumb that every doubling of a cache's size should reduce the cache misses by 30%. Solving the recurrence relation

$$0.7 f(x) = f(2x)$$

yields a polynomial of the form

$$f(x) = \beta x^{\alpha}$$

where α and β are constants, α is negative, and β is a scaling term.

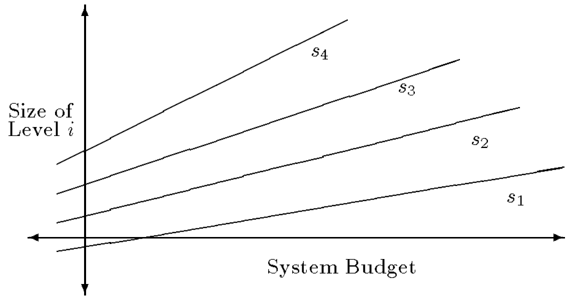


Fig. 5. The graph of Equation 7. Note that every solution is linear and that some solutions can have negative values, or non-zero values at budget zero. This is an artifact of using Lagrange multipliers and assuming the variables $s_1, \ldots, s_n$ can take on any values, even negative ones. This is fixed in the section *Undoing the Effects of Negative Solutions.*

B.3 Undoing the Effects of Negative Solutions

Eq 7 can yield negative values for s_i, particularly for small system budgets. As it is obviously impossible to have negative amounts of memory, a level with negative size should actually have zero size and not appear in an optimal hierarchy. This leads to the concept of a crossover budget for a hierarchy level i, called χ_i. Only when the budget is greater than χ_i does the optimal system include level i; for cheaper systems, money is best spent on other hierarchy levels. When a hierarchy level is not part of the system, it is not part of the system cost or the average access time.

To find the crossover budgets, we note that the highest level in the hierarchy has an optimal size less than zero at small budgets. We calculate the budgets where the size is negative and remove this level from consideration at these budgets. The resulting equations are identical to the original but with a few subscripts changed. This process is repeated down the hierarchy to obtain the final equations. The process is described by the following:

1. Each s_i is a linear function of B with positive slope since every cost and access time is positive and so every $\Psi_{ij} > 0$.
2. Simple inspection of the relative costs and access times of the technologies shows that $\Psi_{ij} > 1$ when $i < j$, and, since Ψ_{ji} is the inverse of Ψ_{ij}, $\Psi_{ij} < 1$ when $i > j$. Therefore, at least when $i = 1$ (when looking at the topmost cache level),
$$1 - \Psi_{ij}^{1/\alpha} < 0 \text{ for all } j,$$
and so the constant term (y-intercept) of the linear equation (Eq 7) is negative; this means the optimal size of level $i = 1$ at budget $B = 0$ is negative. When $i = n$, the y-intercept is positive for a similar reason; the optimal size of level $i = n$ at budget $B = 0$ is positive. This means that the analytic optimal solution has traded cache level 1 for more of level n.
3. A priori, we cannot tell whether any other s_i has a positive or negative y-intercept, so we look further at s_1. Eq 7 leads to positive solutions for level i when
$$B > \sum_{j=1}^{n} c_j \beta(\Psi_{ij}^{1/\alpha} - 1) \qquad (8)$$
This then is the point where s_1 becomes non-negative; the crossover point of s_1 is at budget value
$$B = \sum_{j=1}^{n} c_j \beta(\Psi_{ij}^{1/\alpha} - 1)$$
4. For system budgets less than this value, we remove level 1 from consideration. Without level 1, we appropriate the budget across the $n-1$ remaining levels, hierarchy levels 2 through n. This leads to a new equation similar to Eq 2 but with an integral from s_2 to infinity, as well as a new equation similar to Eq 1 but starting the sum at level 2. We can solve these new $n-1$ equations for $n-1$ unknowns in the same manner as before (but without reference to level 1). This means the equations for T and B change to become functions of one fewer variables, so that level 1 of the hierarchy affects neither the average access time nor the budget. We now obtain a general solution for the size of the ith level in this reduced hierarchy:
$$s_i = \frac{B + \sum_{j=2}^{n} c_j \beta(1 - \Psi_{ij}^{1/\alpha})}{\sum_{j=2}^{n} c_j \Psi_{ij}^{1/\alpha}}$$
We have obtained a form identical to the original equation, except for the indices of the summations, which now sum from level 2 to level n. This marks the end of one iteration.
5. Applying the same procedure with the new set of equations, we see that the size of s_2 is negative for budget values
$$B > \sum_{j=2}^{n} c_j \beta(\Psi_{ij}^{1/\alpha} - 1)$$
We can now remove level 2 from the analysis and obtain a new equation for s_i.

When this process is repeated down through every level in the hierarchy, we find that the cross-over budget for each level is given by
$$\chi_i = \sum_{j=1}^{n} c_j \beta(\Psi_{ij}^{1/\alpha} - 1) \qquad (9)$$
where we define χ_0 to be infinity for notational brevity. In general, we find that $\chi_1 > \chi_2 > \cdots > \chi_n$, breaking up the system budget domain into a convenient collection of intervals. The values for a given s_i (call it $s_{i,k}$) are only valid on the interval $[\chi_k, \chi_{k-1}]$:

- between $B = \chi_1$ and $B = \infty$, all levels in the hierarchy appear,
- between $B = \chi_2$ and $B = \chi_1$, levels $2..n$ appear,
- between $B = \chi_3$ and $B = \chi_2$, levels $3..n$ appear, etc.

The value of s_i on the interval $[\chi_k, \chi_{k-1}]$ is then given by:

$$s_{i,k} = \begin{cases} 0 & \text{when } i < k \\ \frac{B+\sum_{j=k}^{n} c_j \beta (1-\Psi_{ij}^{1/\alpha})}{\sum_{j=k}^{n} c_j \Psi_{ij}^{1/\alpha}} & \text{when } i \geq k \end{cases} \quad (10)$$

Remember that B is the system budget, and the only independent variable in the equation. Eqs 9 and 10 are depicted graphically in Fig 6, picturing examples of 3- and 4-level hierarchies. The costs and access times for the technologies in the hierarchy are constants and need only be "realistic" values: costs should monotonically decrease and access times should monotonically increase as one moves down the hierarchy (to a larger i). The figures are applicable across all choices of technologies for the memory hierarchy using realistic values for costs and access times.

Given a budget and a workload characterization, and told to find the appropriate cache organization, one would first find the crossover values for the levels in the hierarchy, using Eq 9. This would indicate which cache levels should be present in the hierarchy at that budget, and what value of k to use in the next step. The next step is to use Eq 10 to find the sizes of each level at the budget value, on the interval indicated by the value of k. Alternatively, one could find the cache sizes for every realistic budget value from zero up. Here, one would only need to use Eq 10, and use all values of k, from 0 to n.

IV. Discussion

We have found a closed-form solution for the size of each level in a general memory hierarchy, given device parameters (cost and speed), available system budget, and a measure of the workload's temporal locality[3]. The solution is given by Eqs 9 and 10.

A. The Bottom Line

The solution indicates how one should spend one's money. The first dollar should go to the lowest level in the hierarchy. As money is added to the system, the size of this level should increase, until it becomes cost-effective to purchase some of the next level up. From that point on, every dollar spent on the system should be divided between the two levels in a fixed proportion, with more *bytes* being added to the lower level than the higher level. This does not necessarily mean that more *money* is spent on the lower level. Every dollar is split this way until it becomes cost-effective to add another hierarchy level on top, and from that point on every dollar is split three ways, with more bytes being added to the lower levels than the higher levels, until it becomes cost-effective to add another level on top. Since real technologies do not come in arbitrary sizes, hierarchy levels will increase as step functions approximating the slopes of straight lines.

B. A More In-Depth Look at the Analysis

The closed-form solution has several implications.

First, note that the crossover budget for level i is always larger than the crossover budget for level $i+1$ and that the crossover budget for level n is 0. This means that the optimum hierarchy with a small budget (between 0 and χ_{n-1}) consists solely of the slowest, cheapest cache level; all other levels do not exist. This is counter-intuitive—we (including the authors) normally think of adding the fastest cache level first in an attempt to speed up the average access without concern for the worst-case access. Our solution shows that this intuition is incorrect—the slower, cheaper cache level can capture more of the misses to backing store, and it is far more valuable to prevent references from having to be satisfied by a tape drive with a 15-second access time than to optimize the access time of hits higher up in the hierarchy. Once the slowest cache level is large enough to divert a large fraction of the misses to backing store (at system budget χ_{n-1}), we then start increasing the next higher cache level ($n-1$) along with the lowest level.

The crossover budget for a given level i depends on workload and device parameters (Eq 9):

- χ_i decreases with better temporal locality, that is, smaller values of β or larger values of α. As expected, better temporal locality favors adding higher cache levels sooner.
- χ_i decreases as the devices for the higher cache levels improve in cost and speed. Ψ_{ij} decreases with lower cost and faster times (described in detail later); both these technology improvements decrease the crossover budget.

To summarize our first conclusion, *money spent on a given level is money wasted if the level below it is not large enough*. If the lower level is not large enough, it allows too many performance-crippling accesses to the backing store.

Second, we see that, within each region $[\chi_i, \chi_{i-1}]$, the size of each level increases linearly with system budget. That is, within each region, additional dollars are spent according to a fixed proportion.

Third, note that the slope of s_i between any two crossover budgets is higher for larger values of i, because Ψ_{ij} decreases with i and is in the denominator of the slope in Eq 10. Thus, even when one allocates money to increase the size of a fast cache level, one still should increase the size of the lower cache level even faster[4]. The rate at which level i increases depends on workload and device parameters (Eq 10):

- The difference in slopes between higher and lower cache levels decreases with better temporal locality, that is, with larger values of α. With high locality, cache levels increase in size at nearly the same rate.
- The slope for a cache level increases as the devices for the cache level improve. Ψ_{ij} decreases with lower cost and faster times[5]; both these technology improve-

[3] The workload's locality is represented by two numbers which can be obtained very easily through numerical analysis, such as curve-fitting. One can fit a polynomial of the form $P(x)$ in Eq 5 to a roughly estimated hit rate curve to find α and β, without having to do a stack distance measurement of a real workload. Two points will suffice, for example the cache sizes where one would expect to catch 50% and 95% of the reference stream.

[4] Its size increases faster; its cost may or may not increase faster.

[5] Since t_i does not appear in Ψ_{ij} it is a bit more accurate to say that Ψ_{ij} decreases as the cost of level i decreases and as the access times of all other technologies increase relative to that of level i.

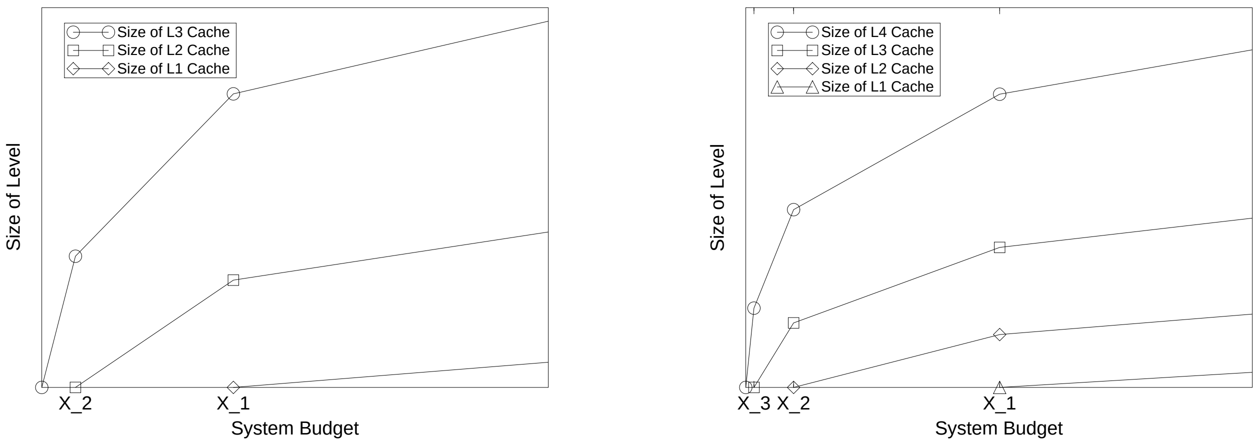


Fig. 6. An example of solutions for two larger hierarchies. A three-level hierarchy is shown on the left; a four-level hierarchy is shown on the right. The X_1,X_2 labels represent crossover values χ_i. The crossover budget for the lowest level in the hierarchy is always zero. The crossover budget for the next highest cache level follows, and so on. Between crossover budgets the equations are linear, and the curves simply change slopes at the crossover budgets to adjust for the cost of a new level in the hierarchy. These graphs were produced with values of c_i and t_i similar to those found in Tables I and II.

ments decrease the denominator of Eq 10 and hence increase the rate that the size of this level increases.

C. Meanings and Interactions of Ψ_{ij}, α, and β

Remember that Ψ_{ij} has the following form:

$$\Psi_{ij} = \frac{c_i t_{j+1}}{c_j t_{i+1}} = \frac{c_i / t_{i+1}}{c_j / t_{j+1}}$$

Ψ_{ij} is the ratio of cost-performances between two levels. This suggests that each level in the hierarchy can be characterized by two numbers; the cost of the technology at that level and the speed of the technology in the level beneath it. This is the effectiveness of a given cache level; it explains how good a job the level does in cost per second cut, or how many dollars per byte it costs to save a second of references to the next lower level. These ratios combine to characterize the entire hierarchy; every level gets compared against each other in the equations.

The temporal locality of a workload is characterized by the two variables α and β. The variables α and Ψ_{ij} always appear together ($1/\alpha$ is always in the exponent of Ψ_{ij}). The cost-performance ratio Ψ_{ij} indicates how good a job one level does at reducing access time compared to another level, and scales how large each of the levels should be in relation to one another. The α term tempers the effect of the cost-performance ratio. For example, when locality is good, α is large (the shape of the locality curve is steep), so $1/\alpha$ is small, and the effect of the cost-performance ratio in differentiating the levels is small. The result is that crossover budgets will be closer to the y-axis; it will make more sense to include the upper cache levels at smaller budgets. The size of different levels will increase at similar rates. In the traditional characterization of a memory hierarchy as a pyramid, the difference in sizes between one level and the next will be much less than if the locality were poor; when locality is good, it will be a narrow and tall pyramid.

When locality is poor, the shape of the differential curve will be less steep and α will be closer to 1. The effect of Ψ_{ij} will be more pronounced: the size of different levels will increase at different rates, and the crossover budgets will be much further out—when crossover budgets do occur, the sizes of the existing levels will be much larger than in the case where locality is high. The result will be a much broader hierarchy than in the previous case; it will take more money to add on the higher levels, and the base levels will be much larger when the higher levels *do* get added. Just as a workload with good locality results in a tall and thin hierarchy, a workload with poor locality results in a short and wide hierarchy.

The β term is a scaling factor; its units are the same as s_i (be it bytes, kilobytes, megabytes, etc.) and its effect is to scale where the crossover budgets occur on the x-axis by scaling the y-intercepts. When the workload spans an enormous amount of data and a convenient unit for graphing is chosen to be MBytes, β will tend to be large, pushing the crossover budgets further out. When the workload spans a smaller range and a smaller unit for β is chosen, β will be smaller, drawing the crossover budgets in.

D. Using the Model to Choose a Subset of Technologies

The model specifies the optimal size of each level with a given set of technologies. By finding the crossover budgets, the model also determines when higher levels in the hierarchy should not exist. However, the model does not automatically determine if technologies in the middle of the hierarchy should be removed. For instance, consider a hierarchy of RAM, disk, and tape, where the disk is almost as expensive as RAM and almost as slow as tape. The model will suggest the best way to arrange the three levels, given an operating budget. The model does not attempt to find any weak links in the hierarchy, except for levels from the top down. If the model decides that the RAM should be part of the hierarchy, the disk will also be kept. In this example, the model may suggest a configuration where the disk level is only slightly larger than the RAM level, indirectly showing that the disk technology is useless in the hierarchy.

Since the model takes only a moment to recommend a configuration, we can easily use it to choose a subset of devices from a larger pool of technologies. This is similar to Przybylski's dynamic programming approach to hierarchy optimization [12], but it is much simpler because we can quickly search through all possible subsets. This process will find the best organization of the best subset of technologies at a given budget point.

V. Verification

The analysis in Section III makes the following simplifications:

- The polynomial stack distance curves do not perfectly model a real workload. In particular, a cache would need to be infinitely large to achieve a 100% hit rate with a polynomial stack distance curve, while only a finite size is needed to achieve this with a real workload. At large budget values, our solutions recommend endlessly increasing the size of all levels; the optimal design would cease increasing the size of a level once it contained all data in the trace.
- The model assumes a fully associative cache model at all levels of the hierarchy.
- The model ignores the effects of a block size, including a certain amount of prefetching and cache pollution.
- The model does not distinguish between read and write behavior. All accesses are treated as reads—they incur delay when issued rather than when forced out of the cache for consistency or cache overflow.
- The model ignores compulsory misses. This affects the performance predicted by Eq 2. However, it has no effect on the optimal hierarchy design, since cache levels of all sizes will miss these references (assuming no prefetching).
- The performance of each technology is characterized by a single access time that does not change as the size of the cache grows.
- The cost function for each technology is strictly proportional to size; there is no extra cost for the first byte.

These simplifications make Eq 2 less accurate at predicting hierarchy performance, possibly affecting the optimal hierarchy recommended by Eqs 9 and 10. The goal of this section is to verify that the general hierarchy configurations recommended by Eqs 9 and 10 do indeed achieve close to optimal performance. To this end, we compare the model's ability to recommend specific sizes for general hierarchies against simulations of real technologies. We performed exhaustive, detailed simulation of two hierarchies, using two different traces from real applications, and specifications from real technologies. The first simulation is of an AFS server with memory, disk, and optical disk; it uses a month of network file requests as the trace. The second is of a memory hierarchy with on-chip cache, off-chip cache, and main memory; it uses an application-level, virtual address trace as the workload.

A. Simulator Description

Our simulator connects together device modules such as CPU caches, main memory, disk drives, optical disks, and cartridge tape robots. The device modules simulate the various caches and keep track of usage statistics. At each level, if the item requested is not present it is requested from the next level down. The request time at each level is the time to first access plus the amount of data to transfer divided by the transfer rate.

In the storage hierarchy simulations, all caches were modeled as write-through with fetch-on-write [17]. In the processor-cache hierarchy simulations, the caches were modeled as writeback with fetch-on-write. None of the simulated caches are fully associative. The I/O caches are set-associative; the RAM file cache is 256-way set-associative, and the disk cache is 1024-way set-associative. The off-chip cache is 4-way set associative, and the on-chip cache is modeled as direct-mapped.

We simulate different ways to allocate money, where the quantum of money was \$256 for the storage hierarchy simulation and \$64 for the processor-cache hierarchy simulation.

B. Storage Hierarchy Simulations

We simulated a storage hierarchy similar to that of Plan 9 [11, 13], where the file system lives entirely on an optical disk jukebox and is cached by DRAM and magnetic disk. Table I describes the specifications used in the simulator and analytical calculations for the constant values of the various c_i and t_i (specifications taken from [6, 14]).

B.1 Workload

The data that was used for the workload in the I/O hierarchy simulations was collected by a logging AFS server [1]. The server sees all requests not serviced from the client's local AFS disk cache[6]. We used one month's worth of trace data from a single file server; April 14, 1992 through May 8, 1992. The full traces represent over 20 million records of several different types of AFS server requests. We extracted references to the relevant commands *fetchdata*, *storedata*, *removefile*, *createfile*, *removedir*, and *makedir*. The rest of the commands do not read and write data; for example, AFS uses *fetchstatus* to synchronize the local cache with the server.

The traces were analyzed for the measurement of their temporal locality, producing the stack distance curves in Fig 7. By using a standard sum-of-squares technique to fit Eq 5 to the data, we found that $\alpha = 2.91$ and $\beta = 439.68$.

B.2 Optimal Configurations

Fig 8a shows the optimal sizes of the hierarchy levels as predicted by our analysis (Eqs 9 and 10). The crossover budget for the disk level is zero since it is the lowest cache

[6] This client cache was found to capture only a few MB of data for this server.

TABLE I
Summary of specs used for storage hierarchy simulation

Technology	Simulated Capacities	Block Size	Cost per MB	Startup Time	Bandwidth	Time to Access 1 Block
DRAM	8MB-64MB	8 KBytes	$32.00 ($256 buys 8MB)	0 ms	160 MB/sec	0.05 ms
Magnetic Disk	512MB-6GB	16 KBytes	$0.50 ($256 buys 0.5GB)	15 ms	10 MB/sec	16 ms
Optical Jukebox	unlimited	64 KBytes	$0.001 ($256 buys 256 GB)	1 sec	1 MB/sec	1063 ms

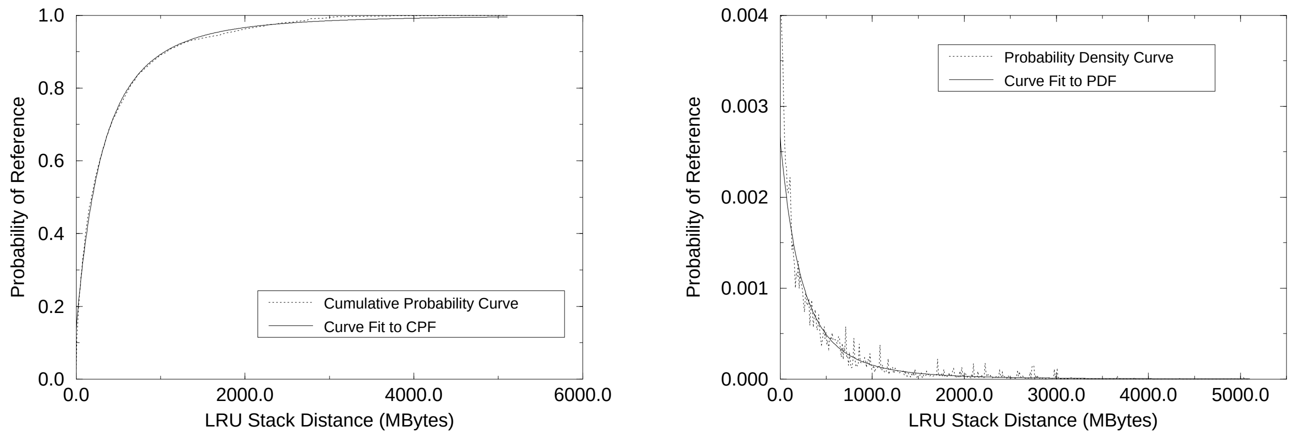


Fig. 7. Stack distance curves for AFS traces. The cumulative probability curve is shown on the left, the probability density curve on the right. Collected data is shown with dotted lines; the curves fit to the data are shown in solid lines.

level in the hierarchy; the crossover budget for DRAM is around $3800, implying that with less than $3800 to spend on cache levels (disk and RAM), all the money should be allocated to disk.

Fig 8b shows the optimal sizes of the hierarchy levels as determined by the simulator. The simulator was written to take into account effects that the model ignores, and the results are noticeably different. Instead of two regions there are really three; the model predicts that the size of the topmost level in the hierarchy will remain zero until the crossover budget; clearly, the crossover budget appears earlier in the simulated results. Also, the model predicts that after the crossover budget, the slope of the RAM line will be constant; this is not the case in the simulated results. Instead, the crossover budget appears earlier and the slope is steady for the middle region, and in the general area of where the model predicts the crossover budget to occur, the slope of the RAM curve really takes off. The region between $1500 and $4000 shows that locality can be exploited by a small amount of RAM, due to an effect that the model ignores.

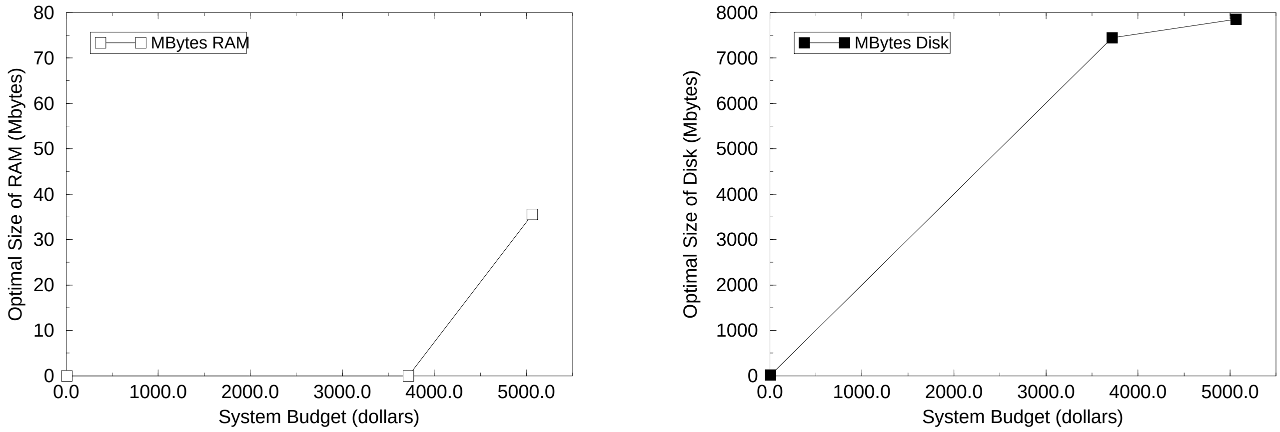
As is evident, the analysis predicts values that are similar to, but not the same as, the optimal values as determined by the simulator. However, this measurement is not enough; what is more important is performance lost by using the optimal configuration from the analysis. Fig 9 shows the performance of four configurations:

- the optimal configuration found by simulation,
- the configuration recommended by the analysis,
- an all-DRAM configuration, and
- a configuration with 512 MB of disk with the remaining funds devoted to DRAM (worst observed case).

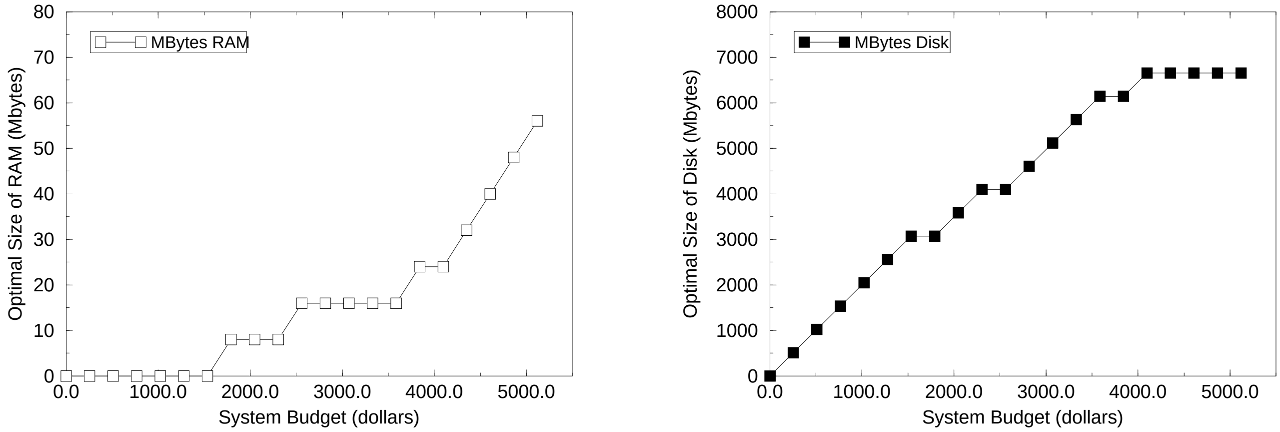
As Fig 9 shows, the predicted optimal configuration never performs more than 5% off the real optimal configuration. In contrast, two reasonable configurations (all DRAM with no disk; 512 MB disk with the remainder going to DRAM) perform as much as 50% worse. Thus, *though the configurations recommended by the analysis differs from the optimal configuration, no performance is lost by using the analytic model.* However, the time saved by not needing the simulations was substantial, as the simulations ran for many months of computer time while applying the analysis took about 30 minutes to write an appropriate Maple script, and less than a second to execute it.

C. Processor-Cache Hierarchy Simulations

We also simulated a typical three-level virtual memory hierarchy consisting of on-chip cache (L1), off-chip cache (L2), and main memory. We used *pixie*-generated traces of several programs from the SPEC92 benchmark suite: *dnasa7*, *espresso*, *hydro2d*, *mdljdp2*, *mdljsp2*, *su2cor*, and *wave5*. These programs were chosen above others because of their large cache footprints, necessary for making multi-level cache simulations run in a reasonable amount of time.



(a) Analysis of storage hierarchy



(b) Simulation of storage hierarchy

Fig. 8. Optimal configurations of the storage hierarchy. The optimal configurations, both predicted (a) and measured (b), as functions of system budget. The x-axis represents the amount of money available and the y-axis represents the optimal size of each level in the hierarchy. The optimal configurations measured by simulation are more accurate, as the simulator takes into account a number of things ignored by the model: writes, block size, access time variance, and real traces.

Table II describes the specifications used in the simulator and analytical calculations for the constant values of the various c_i and t_i.

The cost of on-chip cache needs a bit of explanation. In one sense, on-chip cache is completely free; it is a necessary part of the design and cache appears on nearly all microprocessors. On the other hand, it is infinitely expensive because you cannot arbitrarily increase its size. In an attempt to find a feasible analytical medium, we assumed that a CPU costs around \$1K, roughly half its space is devoted to cache, and a typical cache these days is around 32 KBytes. Thus the \$16K per megabyte number.

C.1 Workload

The traces were analyzed for the measurement of their temporal locality, producing the stack distance curves in Fig 10. By using a standard sum-of-squares technique to fit Eq 5 to the data, we found that $\alpha = 2.05$ and $\beta = 24.52$.

C.2 Optimal Configurations

Fig 11a shows the optimal sizes of the hierarchy levels as predicted by our analysis. The crossover budget for the L2 cache is zero since it is the lowest cache level in the hierarchy; the crossover budget for the L1 cache is around \$500. Since the curve fit is so inaccurate, we applied our analytical approach to the raw data instead of a polynomial and obtained the graphs in Fig 11b. The configurations that were optimal according to the simulations are shown in Fig 11c, with error bars demonstrating the configurations that will perform within 10% of optimal (any appropriation of system budget within the error bars would result in a performance within 10% of the simulated optimal). Note that the granularity of the analytical graph is smaller than the simulations; the simulations allocate funds across the hierarchy in \$64 quanta while the analytical scripts allocate funds in \$16 quanta.

Fig 11c shows the optimal sizes of the hierarchy levels as determined by the simulator. Again, the simulator was written to take into account effects that the model ignores,

TABLE II
SUMMARY OF SPECS USED FOR PROCESSOR-CACHE HIERARCHY SIMULATION

Technology	Simulated Capacities	Block Size	Cost per MB	Startup Time	Bandwidth	Time to Access 1 Block
On-Chip Cache	0KB-64KB	16 Bytes	\$16384 (\$64 buys 4 KB)	0 cycles	4 B/cycle	1 cycle
Off-Chip Cache	0KB-1536KB	64 Bytes	\$1024 (\$64 buys 64 KB)	4 cycles	1 B/cycle	20 cycles
DRAM	unlimited	64 Bytes	\$64 (\$64 buys 1 MB)	20 cycles	0.1 B/cycle	660 cycles

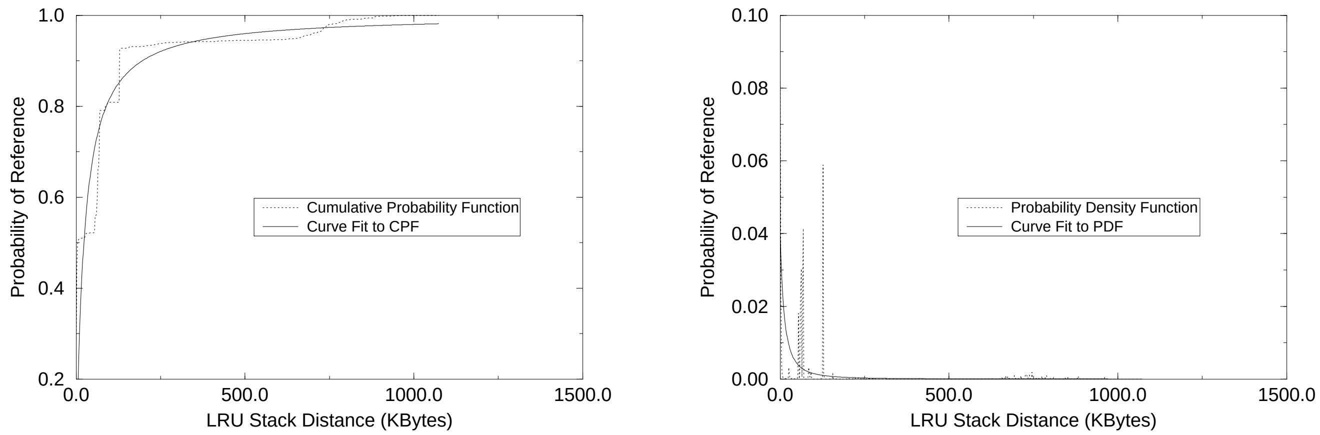


Fig. 10. Stack distance curves for SPEC traces. The cumulative probability curve is shown with its curve fit on the left, the probability density is shown on the right. Collected data is shown with dotted lines; the curves fit to the data are shown in solid lines. The region between 125KB and 600KB of the probability curve is non-zero; the scale required to show the other data obscures this.

and the results are slightly different. Compared to the results taken from the polynomial curve fit, the simulator results are much like the I/O hierarchy results; the model predicts two regions where the slopes will be constant, where the simulated results have several regions and the slopes are not constant. The L1 cache appears earlier, but its size does not really take off until after our predicted cross-over budget. For small budgets, the data suggests that L1 cache is more effective at reducing access time than L2 cache; this is consistent with the probability density curve in Fig 10. Most of the references lie within a 125KB stack distance, and very little lies in the region between 125KB and 500KB, which suggests that the L2 cache will not be truly effective until the budget allows a cache size of 500KB. This compares well with the sharp jump in the L2 cache size (and sharp decline of the L1 cache size) at the \$500 budget point.

If we compare the simulated results to the analytical results taken from the raw data, there is much more similarity; both show that L1 should be present at small budget values, and its size should take off around a budget of \$1000. However, this is less informative than the actual system performance. As before, we compare the performance of the simulated configurations to that of the optimal configurations. Fig 12 shows the performance of five configurations:

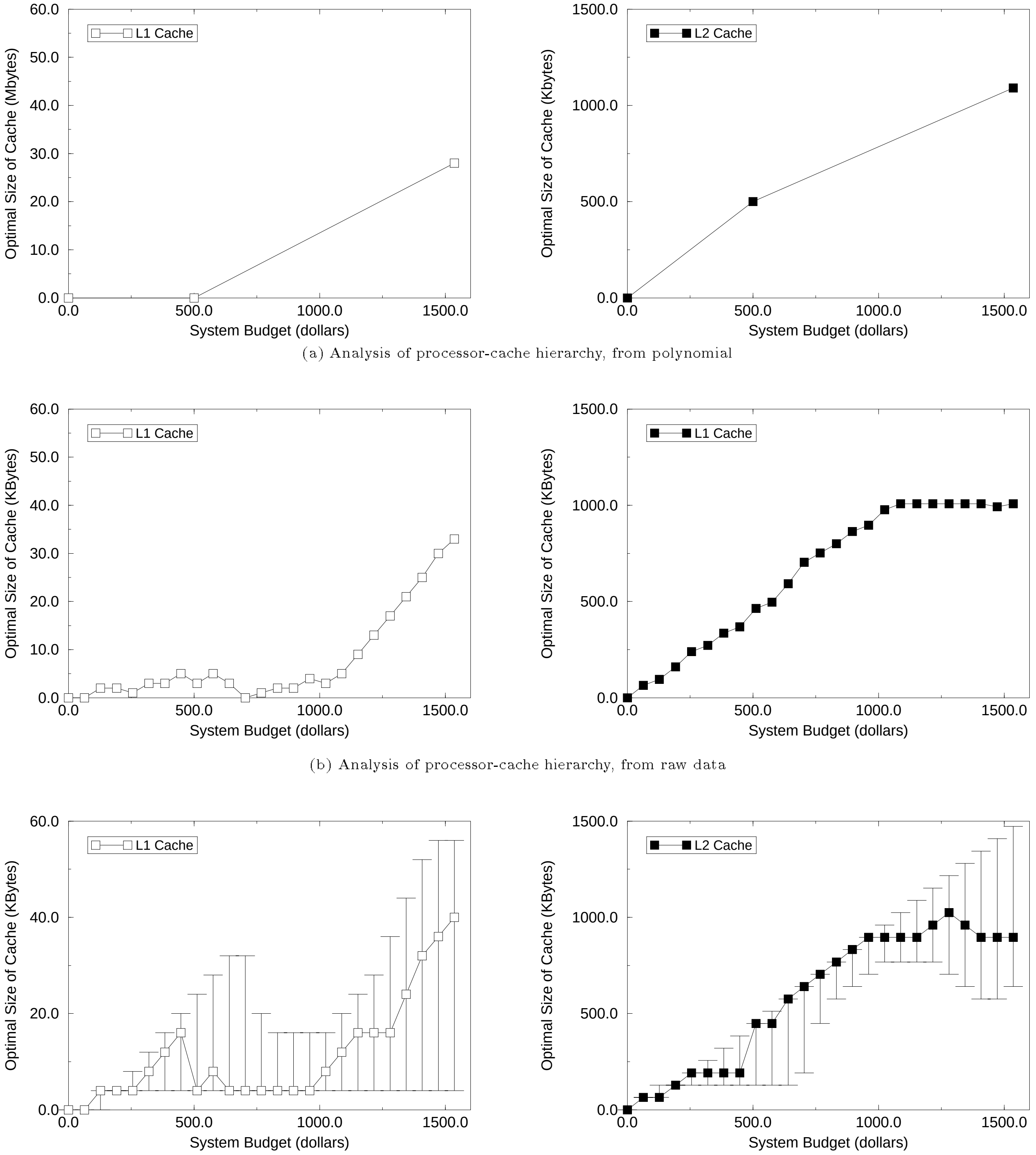
- the optimal configuration found by simulation,
- the configuration recommended by the analysis applied to the polynomial curve fit,
- the configuration recommended by the analysis applied to the raw locality data,
- an all-L2 cache configuration, and
- the worst observed case (which happens to be an all-L1 configuration).

As Fig 12 shows, the configuration predicted via the raw data performs within 5% of the real optimal configuration, except for the one place where it predicts that the L1 cache size should be 0. Here, the performance is several times worse than the simulated optimal configuration. The analytical results using the polynomial fitted curve are equal to the all-L2 configuration until the crossover point (at \$500) and from there, drop down to within 5% of the optimal curve, and remain within 5% from that point on.

In contrast, two perfectly reasonable configurations (all L1 with no L2; all L2 with no L1) perform as much as 800% worse. Again, though the configurations recommended by the analysis differ from the optimal configuration, no performance is lost.

VI. CONCLUSIONS

In this paper, we have derived a simple model for determining the optimal size of each cache level in an n-level



(a) Analysis of processor-cache hierarchy, from polynomial

(b) Analysis of processor-cache hierarchy, from raw data

(c) Simulation of processor-cache hierarchy

Fig. 11. Optimal configurations of the processor-cache hierarchy. The optimal configurations, both predicted (a, b) and measured (c), as functions of system budget. The x-axis represents the amount of money available and the y-axis represents the optimal size of each level in the hierarchy. The optimal configurations determined by simulation are more accurate, as the simulator takes into account items ignored by the model, including writes, block size, access time variance, and real traces. The data in (a) were obtained from the polynomial curve fit, while the data in (b) were obtained from the raw cumulative probability data itself. The error bars in (c) indicate the configuration ranges of L1 and L2 that will give performances within 10% of the optimal configuration.

cache hierarchy. The model is based on the access time (t_i) and unit cost (c_i) for each level, the total system budget for the cache levels (B), and a two-parameter characterization of workload locality (α and β). By using a specific

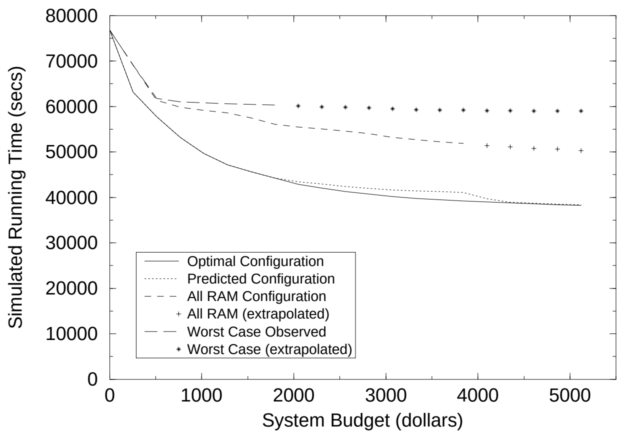


Fig. 9. Performance comparisons for the storage hierarchy. The x-axis represents system budget; the y-axis represents the average time per simulated reference. The times include compulsory misses. The hierarchy consists of two levels, RAM and magnetic disk; backing store is an optical jukebox. The comparison is between the measured optimal running times and the running times of the predicted optimal configurations. Also shown are the running times of an all-RAM system and the worst observed configuration—one with a half gigabyte cache on disk and the rest of the budget devoted entirely to RAM. The amounts of RAM in the last two configurations are not insignificant; they approached 200MB at high budget values.

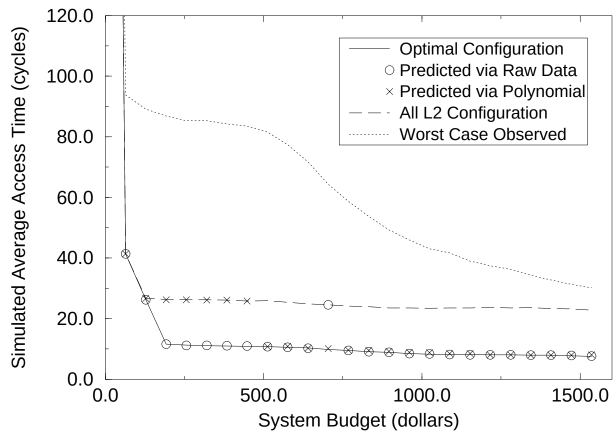


Fig. 12. Performance comparisons for the processor-cache hierarchy. The x-axis represents system budget; the y-axis represents the average time per simulated reference. Note that the times do include compulsory misses. The hierarchy consists of two levels; On-Chip cache (L1) and Off-Chip cache (L2); backing store is main memory (DRAM). The comparison is between the measured optimal running times and the running times of the predicted optimal configurations. Also shown are the running times of an all-L1 system and an all-L2 system. The worst observed configuration is exactly the all-L1 configuration.

form for the stack distance curves, we were able to derive closed-form solutions for the size of each cache level s_i on the interval $[\chi_i, \chi_{i-1}]$ to be:

$$s_{i,k} = \begin{cases} 0 & \text{when } i < k \\ \dfrac{B+\sum_{j=k}^{n} c_j \beta(1-\Psi_{ij}^{1/\alpha})}{\sum_{j=k}^{n} c_j \Psi_{ij}^{1/\alpha}} & \text{when } i \geq k \end{cases}$$

$$\chi_i = \sum_{j=i}^{n} c_j \beta(\Psi_{ij}^{1/\alpha} - 1)$$

$$\Psi_{ij} = \frac{c_i t_{j+1}}{c_j t_{i+1}}$$

These equations successfully recommended configurations with performance within 5% of optimal, as verified by exhaustive simulations of a three-level storage hierarchy and a three-level processor-cache hierarchy.

Our model led to four observations about configuring cache hierarchies. First, it is common to focus erroneously on hit time rather than miss time in designing hierarchies. In contrast, the model shows that miss time is more important until the system budget is large enough to achieve high hit ratios in the lowest cache level. This implies that the first place to spend money when designing a cache hierarchy is the cheapest level, rather than the fastest level. As specific applications of this principle, CPU cache designers should be aware that a larger off-chip cache may yield better performance than a faster, but smaller, on-chip cache. Also, when tertiary storage becomes more common, storage hierarchy designers should be aware that having enough disk will be much more important than having enough RAM.

As a corollary to our first observation, the model recommends increasing the size of the slower cache levels faster than the size of the faster cache levels, even when it makes sense to include those faster cache levels.

Third, we saw from the model that within each crossover budget interval $[\chi_i, \chi_{i-1}]$, the size of each level increases *linearly* with system budget.

Fourth, we observed that the workload locality had an interesting effect on the shape of the memory hierarchy. For workloads with good locality (large α and small β), the optimal memory hierarchy is narrow. That is, the difference in size between cache levels is small. Inversely, for workloads with poor locality (small α and large β), the optimal memory hierarchy is wide; that is, the difference in size between cache levels is large.

Acknowledgments

Enormous thanks to Peter Honeyman, Saar Blumson, and Dan Muntz of CITI, for graciously allowing us to use their AFS traces. Without their data, most of this work would not have been done.

B. L. Jacob's work was partially supported by the Advanced Research Projects Agency under ARPA/ARO Contract Numbers DAAL03-90-C-0028 and DAAH04-94-G-0327. P. M. Chen's work was partially supported by the National Science Foundation under Award Number MIP-9409229.

References

[1] S. Blumson, P. Honeyman, T. E. Ragland, and M. T. Stolarchuk. "AFS server logging." Tech. Rep. CITI-93-10, University of Michigan, November 1993.

[2] C. H. Edwards, Jr. and D. E. Penney. *Calculus and Analytic Geometry.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[3] C. K. Chow. "On optimization of storage hierarchies." *IBM Journal of Research and Development*, pp. 194-203, May 1974.

[4] —. "Determination of cache's capacity and its matching storage hierarchy." *IEEE Transactions on Computers*, vol. 25, no. 2, pp. 157-164, February 1976.

[5] E. G. Coffman and P. J. Denning. *Operating System Theory.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1973.

[6] A. L. Drapeau and R. H. Katz. "Striped tape arrays." In *Proceedings of the 1993 IEEE Symposium on Mass Storage Systems*, 1993, pp. 257-265.

[7] H. Garcia-Molina, A. Park, and L. R. Rogers. "Performance through memory." In *Proceedings of the 1987 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 1987, pp. 122-131.

[8] B. L. Jacob. "Optimization of mass storage hierarchies." Tech. Rep. CSE-TR-228-95, University of Michigan, May 1994.

[9] N. P. Jouppi and S. J. E. Wilton. "Tradeoffs in two-level on-chip caching." In *Proc. 21st International Symposium on Computer Architecture*, April 1994.

[10] J. E. MacDonald and K. L. Sigworth. "Storage hierarchy optimization procedure." *IBM Journal of Research and Development*, pp. 133-140, March 1975.

[11] R. Pike, D. Presotto, K. Thompson, and H. Trickey. "Plan 9 from Bell Labs." In *Proc. Summer 1990 UKUUG Conference*, pp. 1-9, London, July 1990.

[12] S. Przybylski. *Cache and Memory Hierarchy Design: A Performance-Directed Approach.* Morgan Kaufmann Publishers, Inc., 1990.

[13] S. Quinlan. "A cached WORM file system." *Software-Practice and Experience*, vol. 21, no. 12, pp. 1289-1299, December 1991.

[14] S. Ranade. *Mass Storage Technologies.* Meckler Publishing, Westport, CT, 1991.

[15] S. L. Rege. "Cost, performance and size trade-offs for different levels in a memory hierarchy." *IEEE Computer*, vol. 19, pp. 43-51, April 1976.

[16] A. J. Smith. "Two methods for the efficient analysis of memory address trace data." *IEEE Transactions on Software Engineering*, vol. 3, no. 1, pp. 94-101, January 1977.

[17] —. "Cache memories." *Computing Surveys*, vol. 14, no. 3, pp. 473-530, September 1982.

[18] —. "Disk cache-miss ratio analysis and design considerations." *ACM Transactions on Computer Systems*, vol. 3, no. 3, pp. 161-203, August 1985.

[19] H. S. Stone. *High Performance Computer Architecture.* Addison Wesley, 1993.

[20] T. A. Welch. "Memory hierarchy configuration analysis." *IEEE Transactions on Computers*, vol. 27, no. 5, pp. 408-413, May 1978.

Bruce Jacob received the A.B. in Mathematics from Harvard College, Cambridge, in 1988, and the M.S. in Computer Science and Engineering from the University of Michigan, Ann Arbor, in 1995. He is currently a doctoral candidate in Computer Science and Engineering at the University of Michigan, where he is conducting research in operating systems in the Advanced Computer Architecture Lab. He has worked as a distributed system architect and software engineer for the telecommunications startups Priority Call Management and Boston Technology.

His research interests include operating systems, distributed systems, computer architecture, and algorithmic composition.

Peter Chen received his B.S. in Electrical Engineering from the Pennsylvania State University in 1987 and his M.S. and Ph.D. in Computer Science from the University of California at Berkeley in 1989 and 1992. He is currently an Assistant Professor in the Department of Electrical Engineering and Computer Science at the University of Michigan at Ann Arbor. His research interests include operating systems and computer architecture and focus on improving the performance, capacity, and reliability of computer storage systems.

Seth Silverman received his B.S. in Electrical Engineering from the University of Florida in 1988 and his M.S.E. in Computer Science and Engineering from the University of Michigan at Ann Arbor in 1994. He is currently employed as a software development engineer in distributed security at Hewlett-Packard, Chelmsford Systems Software Lab. He previously worked as both systems and ASIC designer at Hewlett-Packard, Business LaserJet Division, and has served as an associate faculty member of the University of Idaho, Boise Engineering Program.

His research interests include operating systems, distributed systems and computer architecture.

Trevor Mudge (S'74, M'77, SM'84, F'94) received the B.Sc. degree in Cybernetics from the University of Reading, England, in 1969, and the M.S. and Ph.D. degrees in Computer Science from the University of Illinois, Urbana, in 1973 and 1977, respectively. At the University of Illinois he participated in the design of several high-performance computers, and conducted research in computer architecture. Since 1977, he has been on the faculty of the University of Michigan, Ann Arbor where he has taught classes on logic design, computer-aided design, computer architecture, and programming languages. He is presently a Professor of Electrical Engineering and Computer Science and the Director of the Advanced Computer Architecture Laboratory, a group of eight faculty and 70 graduate research assistants. He is author of more than 130 papers on computer architecture, programming languages, VLSI design, and computer vision, and he holds a patent in computer-aided design of VLSI circuits.

His current research includes the design and construction of a high-performance microsupercomputer, computer architecture, computer-aided design, and operating systems. In addition to his position as a faculty member, he is a consultant for several computer companies in the areas of architecture and CAD. He is also Associate Editor for ACM Computing Surveys. Trevor Mudge is a Fellow of the IEEE, a member of the ACM, the IEE, and the British Computer Society.