7-1-1993

Expected Values for Cache Miss Rates for a Single Trace (Technical Summary)

Russell W. Quong
Purdue University School of Electrical Engineering

EXPECTED VALUES FOR CACHE MISS RATES FOR A SINGLE TRACE

RUSSELL W. QUONG

TR-EE 93-26
JULY 1993

SCHOOL OF ELECTRICAL ENGINEERING
PURDUE UNIVERSITY
WEST LAFAYETTE, INDIANA 47907-1285

Expected Values for Cache Miss Rates for a Single Trace
(Technical Summary)

Russell W. Quong
School of Electrical Engineering
Purdue University
W. Lafayette, IN
quong@ecn.purdue.edu

Sun Microsystems Laboratory, Inc.
Mountain View, CA

July 13, 1993

Abstract

The standard trace-driven cache simulation evaluates the miss rate of cache C on an address trace T for program P running on input data I with object-code address map M for P. We note that the measured miss rate depends significantly on the address mapping M set arbitrarily by the compiler and linker. In this paper, we remove the effect of the address-mapping on the miss rate by analyzing a symbolic trace T of basic blocks. By assuming each basic block has an equal probability of ending up anywhere in the cache, we determine the average miss rate over all possible address mappings.

We present the gap model for predicting the mean and variance of the miss rate for direct-mapped caches. Our model also predicts how an intervening trace, such as an operating system call or a task switch, will affect the miss rate. For fully-associative caches, we give the expected number of misses for different working set sizes. In particular, for a working set of size w, and a fully-associative cache of size L, the expected number of misses until the working set is resident is $\approx L\ln(L/L-w)$ for $w < L$. We present a metric for estimating the interference between two parts of a program, which is important in choosing a address mapping that minimizes cache conflicts. Finally, we present a method to compactly summarize a trace that allows accurate miss rate prediction for direct-mapped caches.

1 Introduction

Our initial motivation for this study came about by observing the design process of a memory system for a commercial workstation. To evaluate the performance of different memory systems, computer architects often use trace-driven simulation to determine cache miss rates. Architects compare predicted miss rates via cache simulators. The address traces are usually from standard benchmark suites (SPEC, Perfect, etc.) and are meant to be representative of typical computer workloads. Unfortunately, architects frequently place significant emphasis on the absolute miss rate numbers because no better method exists. It is natural to ask "Is a miss rate of x for benchmark y on memory system z good, or bad?" More importantly, how do we answer such a question?

The measured cache miss rate depends on four factors, (i) the program **P** being executed, (ii) the input data **I** for P, (iii) the type of cache used, and (iv) the specific address mapping **M** of the object code for **P**, which is determined automatically by the compiler and linker.

We are concerned with the last factor. An address mapping, **M**, assigns each basic block in **P** to a unique set of physical addresses. A mapping is itself affected by many factors including compiler optimization (such as procedure inlining), the order the object modules are linked, and the specific libraries on a system. We wondered how much the miss rate would vary if we linked program b differently?

As such, we question the accuracy of using a single address trace from 'P based on a single mapping to represent the expected behavior of **P**, even for similar input data, because of seemingly arbitrary variations in the address mapping from system to system. We have heard of extreme stories in which changing the order object modules are linked has changed the execution time by a factor of two due to caching effects. It is apparent the specific address mapping can have significant effects on cache performance. In a worse case scenario, the most frequently executed code would be mapped over one another. Although, such a scenario is unlikely, it is natural to ask how likely are bad address mappings? In particular, how much variation in miss rate are we likely to observe simply due to different address mappings?

To isolate the effect of an address mapping from a particular address trace, we view the execution of **P** on **I** as a block trace (or, simply trace) as a sequence of "named" blocks, e.g. "block_{12} of function `sqrt`". Thus, for any address mapping **M**, running program **P** on input **I** always produces the same trace. The advantage of defining traces in this way is that small perturbations to **P** or **I** produce small perturbations to the resulting trace. For example, if small changes are made to **P** over time, such as fixing bugs or adding new features, we expect the traces from the modified **P** and the original **P** to be roughly the same. The same argument holds true for input **I**. E.g. if **P** is a VLSI layout tool, we would expect similar traces when laying out similar circuits.

Given a block trace, we derive formulas for the expected number of misses averaged over all possible address mappings of that trace for direct-mapped and fully associative caches. Our method has the advantage of defining a single meaningful number for the miss rate of a given memory system for a specific benchmark. Our cache models handle the expected effect on the miss rate of intervening traces, such as process switches. Finally, our models provide a quantitative measure of the expected interference in the cache between two parts of a program.

The paper is organized as follows. Section 2 gives states our definitions and assumptions. In Section 3, we present the gap model and derive equations the mean and variance of the miss rate for direct-mapped caches. We also show how to model the effect on the miss rate of intervening traces, such as an kernel traps or kernel calls. Finally we describe how to compactly summarize a trace by giving gap counts. In Section 4, we derive the expected number of misses for a fully associative cache using random replacement for different working set sizes.

2 Definitions

A cache is described by the triple $C(L, \mathrm{LS}, \mathrm{SS})$, where L = the number of lines in the cache, LS = the line size in bytes, and SS = the number of sets or the set size. A cache has SS sets, each of which contains L/SS lines or slots. For a direct-mapped cache, SS = L; for a fully-associative cache, SS = 1. Otherwise, we have a a-associative cache, where $a = L/SS$. In this paper, all addresses and sizes will be measured in units of cache lines, so that the phrase "u unique addresses" means u unique cache lines. Two addresses collide if they map to the same cache line. We define x = the probability that two different addresses collide. For a direct-mapped cache, $x = 1/L$. An address in the cache is called resident.

We use lower case letters for program-specific values and upper case symbols for trace-specific values. We denote definitions via italics or with the $\equiv$ symbol. We assume we have a program P whose instruction segment is partitioned into a set of n (basic) blocks $B = \{b_1, \ldots b_n\}$. A *block* b is a segment of machine code such that if one instruction in I is executed, all of b must be executed. We denote the size of block b_i as s_i, where s_i is the number of cache lines occupied by b_i; note that s_i can be fractional. E.g. on a cache with 32-byte lines, a 40-byte block has size $40/32 = 5/4$.

We are given a fixed trace, $T(P, I)$, for program P running input I. A block trace or tmce T is sequence of blocks (not addresses) $B_1, B_2, \ldots, B_N$, where $B_i \in B$. Because we are interested in the state of the cache between reference, we view block B_t as being accessed or referenced immediately *after* time t. If $B_t = b_i$, then all of block b_i is contained in the cache at time $t+1$. As our analysis assumes a fixed trace T, and we omit T when convenient.

Times are denoted by the letter t. A time interval is denoted $[t, t']$ where $t \leq t'$.

The term N_i denotes the number of times block b_i is referenced in T, thus, $N = \sum_i^n N_i$. The term $t_i(k)$ represents the time of the k-th reference of block b_i. A *gap* of block b_i is the time interval between references to b_i. The first gap of b_i is $[0, t_i(1)]$, namely, the time interval from startup to the first reference to b_i[1]. There are N_i gaps of b_i. We denote the kth gap of b_i as $[\mathrm{gap}_i(k)] = [t_i(k-1)+1, t_i(k)]$. The length of $[\mathrm{gap}_i(k)]$ is $t_i(k) - t_i(k-1)$.

We define $U[t', t''] = \{b_i | B_{tt} = b_i, t' < tt < t''\}$ = the set of unique blocks accessed between time t' and t'', not including time t' or t''. We denote the size of $U[t, t']$ as $|U[t, t']|$, so that $|U[t, t']|$ is the number of unique addresses referenced between time t and t'. We define the size of gap $[\mathrm{gap}_i(k)]$ as $u(i, k) \equiv$ the number of unique references in the kth gap of b_i. Table 1 summarizes the definitions used in this paper.

As an example, let P consist of basic blocks $B = \{b_1, b_2, \ldots, b_5\}$, where $s_i = i$. Let trace $T = b_2, b_4, b_1, b_2, b_5, b_1, b_4, b_2$. We have $B_1 = b_2$, $B_2 = b_4$, $B_3 = b_1$. The gaps of b_2 are $[\mathrm{gap}_2(1)] = [0, 1]$, $[\mathrm{gap}_2(2)] = [2, 4]$, and $[\mathrm{gap}_2(3)] = [5, 8]$. We have $U[2, 7] = \{b_1, b_2, b_5\}$, $|U[2, 7]| = 1 + 2 + 5 = 8$, and $u(2, 3) = |U[5, 8]| = 5 + 1 + 4 = 10$.

For any random variable (r.v.) Z, $E[Z] \equiv \bar{Z}$ = the expected value of Z, and $\mathrm{Var}[Z] \equiv$ the variance of $Z = E[(Z - \bar{Z})^2]$. If $Z \in \{0, 1\}$ (a Bernoulli trial), then $\mathrm{Var}[Z] = \bar{Z}(1 - \bar{Z})$. We use

[1] This definition allows us to catch cache misses at startup and also simplifies later formulas because there are N_i gaps rather than $N_i - 1$ which would have resulted if we ignored the first gap.

Program value		Trace value	
b_i	block i of program P	B_t	block accessed just after time t
n	number of blocks in P	N	number of blocks in trace T
		N_i	# times b_i is accessed in T
s_i	size of block i (in cache lines)	$u(i,\mathrm{k})$	# unique addr in k-th gap of b_i (gap size)
[t, t']	time interval from t to t', t $\leq t'$	$g(i,\mathrm{k})$	# addr of k-th gap of b_i (gap length)
		$S(i,\mathrm{t})$	# of resident addresses of b_i at time t
		$X_i(k)$	# of misses blamed on b_i in its kth gap
		$t_i(k)$	time of k-th reference to block b_i
L	size of the cache (in lines)		
x	probability two addresses collide		

Table 1: Definitions

random variables named S = 1 to represent blocks surviving in the cache, and X = 1 to represent cache misses.

We make the following assumptions in our analysis.

1. For $1 \leq i \leq$ n, $s_i \ll |L|$ — each block b_i is much smaller than the cache, e.g. each block is no larger than 1/20 of the cache size.
2. $|C| \gg 1$ — the number of cache lines is much greater than 1, e.g. the cache has at least 50 lines.
3. $|P| \gg |C|$ — the size of program P is much larger than the cache.
4. Block b; is equally likely to start at any line in the cache. This assumption follows from 3, by considering all possible address maps of P. (This assumption allows for physically impossible mappings such as having every block start at line 1 in the cache. It can be proven that the physically impossible mappings are improbable enough to have an insignificant effect on the final answer. Throughout this paper, we shall assume x the probability that any two addresses A_1 and A_2 collide is $1/L$. In practice, address mappings are continuous, laying out blocks one after another, so that

$$x = \frac{|P| - L}{|P| * L} = \frac{1}{L} - \frac{1}{|P|} \approx \frac{1}{L}, \qquad \text{when} |P| \gg |L|$$

   To apply this correction, use the exact value for x wherever the term $1/L$ is found.
5. Addresses in the same block do not collide with each other except for a fully associative cache using random replacement.

In practice, these assumptions are usually valid. Note, if blocks are basic blocks in machine code, the assumptions on block size are quite reasonable.

3 A Gap Model of Direct Mapped Caches

3.1 The expected miss rate

We blame each cache miss on the block currently being accessed, which then replaces a resident address. We calculate the expected number of cache misses in trace T by adding the expected number of misses for each block b_i over all blocks. To determine the number of misses blamed on block b_i, we consider each of the gaps of b_i. The intuitive reason for using gaps is that if $B_t = b_i$, at $t+1$ all of b_i must be in the cache, so that the previous status of b_i is irrelevant for future times. We define the random variable

$$S(i,t) \equiv \text{the number of addresses of } b_i \text{ in the cache at time } t.$$

$S(i,t)$ denotes the number of "survivors" of b_i since its last reference. To determine $S(i,t)$ assume $[t,t']$ is a gap of block b_i and blocks b_j, b_k, b_j are accessed at time $t+1, t+2, t+3$, respectively as shown below.

time:	t		$t+1$		$t+2$		$t+3$		...	t'	
block:		b_i		b_j		b_k		b_j	...		b_i

Accessing block b_j replaces s_j lines in the cache. Over all possible address maps, each line of b_j has a $(1 - s_j/L)$ chance of surviving. (This analysis remains true even if part of b_j was already in the cache.) The expected number of cache lines for b_i that survive past the reference to b_j is $\mathrm{E}[S(i,t+2)] = s_i(1 - s_j/L)$. Similarly, after referencing b_k at $t+2$ we have $\mathrm{E}[S(i,t+3)] = s_i(1 - s_j/L)(1 - s_k/L)$.

However, $S(i,t+4) = S(i,t+3)$, because at $t+3$ we reference b_j which has already been referenced earlier in this gap. This and subsequent references to b_j in this gap cannot replace any b_i lines in the cache, because the first reference to b_j would have already done so. Thus, only the first access to each block in the gap matters, and we need only consider the set of unique blocks accessed in $[t,t']$, namely $U[t,t']$. As a result, the expected value of $S(i,t')$ is

$$\mathrm{E}[S(i,t')] = s_i \prod_{\ell} (1 - s_\ell/L), \qquad b_\ell \in U[t,t']. \tag{1}$$

In practice, we will probably need to add a correction factor because blocks that follow one another in the trace are often "neighboring" blocks in program (e.g. spatial locality). Thus, a significant number of conflicts from the gap model simply won't occur in practice. The reason is that the compiler and linker place neighboring blocks at adjacent addresses, with the result that they will not collide. Our correction consists of labelling pairs of blocks (b_i, b_j) *exempt* from colliding with each other. For example, we might declare all blocks in the same procedure exempt from collisions with one another, which assumes the cache is larger than any procedure. With exempt pairs, we modify Equation 1 so that the $(1 - s_\ell/L)$ term exists only for non-exempt blocks b_ℓ of b_i.

We define the random variable

$$X_i(k) \equiv s_i - S(i, t_i(k)) = \# \text{ of cache misses blamed on } b_i \text{ after its kth gap,}$$

because the number of misses is the block size minus the number of surviving lines. Summing over all the gaps of b_i, we get the expected number of misses blamed on I; over the entire trace is

$$\mathrm{E}[X_i] = \sum_{k=1}^{N_i} \mathrm{E}[X_i(k)] = \sum_{k=1}^{K} (s_i - \mathrm{E}[S(i, t_i(k)]) .$$

The total number of misses over the entire trace is the sum of the misses over all blocks, $X(T) = \sum_{i=1}^{n} X_i$. The miss rate is the number of misses divided by the total number of accesses in the trace. If b_i is accessed at time t, $B_t = b_i$, then s_i cache lines are referenced at t.

As a simplification, for L $\gg$ 1 and L $\gg s_\ell \geq 1$, we have $(1 - s_i/L) \approx (1 - 1/L)^{s_i} \approx e^{-(s_i/L)}$. Thus, applied to the kth gap of b_i, Equation 1 becomes

$$\mathrm{E}[S(i,k)] = s_i \prod_{4} (1 - s_\ell/L) \approx s_i \prod_{4} e^{-s_\ell/L} = s_i e^{-u(i,k)/L} \tag{2}$$

$$\mathrm{E}[X(i,k)] = 1 - \mathrm{E}[S(i,\mathrm{k})] = s_i \left(1 - e^{-u(i,k)/L}\right) \tag{3}$$

As an example, Equation 3 shows that if the gap is the same size as the cache, b_i has a $e^{-1} = 36.8\%$ chance of surviving. Summing over all gaps of blocks gives the expected number of misses over T.

$$\mathrm{E}[X(T)] = \sum_{i=1}^{n} s_i \left(\sum_{k=1}^{N_i} 1 - e^{-u(i,k)/L} \right) . \tag{4}$$

We grossly categorize gaps sizes as either small (survival is likely), medium (survival depends on the specific address map) or large (survival unlikely). For small gap sizes u less than $L/4$, the chance of surviving is $\approx (1 - u/L)$. For gaps larger than $3L$, b_i has less than a 5% chance of surviving, indicating b_i is almost always knocked out. Thus, for any address mapping, blocks will survive small gaps and will not survive large gaps. Alternatively phrased, for a specific address mapping, the main factor of the actual cache performance is the miss rate on mid-sized gaps, e.g. those with sizes between $L/30$–$3L$.

Note that our model provides quantitative insight on how much each block accounts for cache misses. Unlike other miss rate models, Equation 4 is a weighted sum of exponentials. With different weights, Equation 4 accounts for a wide variety of miss-rate versus caches size behaviors.

3.2 Data Caches

The preceding analysis assumes only instruction address traces. We now discuss data-only caches and then mixed caches. When considering data references, some of our previous assumptions no longer apply. We make the following modifications to our model. Data consists of either scalars or arrays. In general, a trace consists of a series of blocks and array references $B_1, B_2, \ldots, B_i, \ldots, B_N$, where B_i is either a basic block, a scalar, or an index from an array, such as "index 42 from array 17." As before, we assume the sizes of all blocks, scalars, and arrays are known.

Each scalar is treated as a block, except that scalars have fractional sizes. E.g. on a cache with 32-byte lines, a 4-byte integer scalar has size 4/32 = 1/8. Allowing fractional sizes is a better model for blocks/scalars that are likely to be in the same cache line, because in Equation 1, we are interested in the number of unique cache lines used. Rounding sizes up to the nearest integer is a better model for blocks that are unlikely to occupy the same line, such as when text and data collide in a mixed cache. As a contrived example, if eight scalars fit in a cache line, and L = 128, what is the likelihood that a scalar will survive through a gap that references 128 other scalars? We believe spatial locality implies that if two scalars are accessed close to each other in time (such as local variables for a given function), they are likely to be close to each other in the address map, perhaps in the same line.

Arrays however, do not fit perfectly into the block model. Like basic blocks, arrays occupy contiguous addresses, and thus, array addresses do not collide with one another (unless the array is larger than the cache). Unlike blocks, however, (i) arrays can be quite large relative to the cache size, and (ii) arrays need not be referenced in their entirety. We model small arrays as blocks that need not be referenced in their entirety.

We categorize arrays as either small or large. If an array A[] is small, say $|A| \leq L/5$, we treat each entry of A as a separate scalar. Otherwise, if an array $A[]$ is large, we cannot use our previous approximation. We define $\bar{S}(i, \mathrm{A}[], \mathrm{T}) \equiv$ the expected number of addresses of block b_i that survive T unique references to A[]. We have $\bar{S}(i, \mathrm{A}[], \mathrm{T}) = (1 - rs_A/L)$, where s_A is the size (in cache lines) of an array element. If the array is accessed sequentially, i.e. via indices 5, 6, 7, 8 ..., then s_A is the fractional size of each array element. Thus, if an array is one-third of the size of the cache and it is accessed in its entirety, then rs_A = L/3 and the probability of an address surviving is 2/3, which is what we would expect.

Thus, we amend the Equation 1 on the expected number of lines of b_i that survive through time t' to be

$$\mathrm{E}[S(i,t')] = s_i \prod_{\ell}(1 - s_\ell/L) \prod_{\kappa}(1 - r_\kappa s_{A_\kappa}/L), \qquad b_\ell \in U[t,t'], A_\kappa[] \in U[t,t'], \tag{5}$$

where each b_ℓ is a block or scalar, and each A_κ is a large array of with elements of size s_{A_κ} to which there are r_κ unique references during [t, t'].

3.3 Mixed instruction and data caches

Although not obvious, the equations for instruction-only caches remain reasonably accurate for traces with both instruction and data addresses. We can still use the approximations from Equation 2 because if (for a large array) rs_A is large relative to the cache size (say, $\geq L/5$), the gap size must also be large, as instruction references must outnumber array references. The inaccuracy of the approximation $(1 - s/L) \approx e^{-s/L}$ only matters when L/5 $\leq rs_A \leq$ L. If $rs_A \gg$ L, the term $e^{-u(i,k)/L} \approx 0$ which remains accurate. If rs_A is small relative to L, our previous approximation is obviously valid. Thus, for the kth gap of b_i, Equations 2 and 3 remain

$$\mathrm{E}[S(i,k)] = s_i e^{-u(i,k)/L} \tag{6}$$

$$\mathrm{E}[X(i,\mathrm{k})] = 1 - \mathrm{E}[S(i,\mathrm{k})] = s_i \left(1 - e^{-u(i,k)/L}\right) \tag{7}$$

where $u(i,\mathrm{k})$ now counts all unique addresses including array references.

3.4 The interference between blocks

We can now quantitatively define the interference between two blocks blocks b_i and b_j. We define two random variables $Y(i,\mathrm{j},\mathrm{k})$ and $Y(i,\mathrm{j},\mathrm{T})$ = the number of cache misses blamed on b_i when referencing b_j *(i)* in the k-th gap of b_i and (ii) over the entire trace T, respectively. As before, we sum $Y(i,j,\mathrm{k})$ for each gap of b_i to get $Y(i,\mathrm{j},\mathrm{T})$. If $[t,t']$ is the k-th gap of b_i and t_j is the first reference of b_j in the gap, then $\mathrm{E}[Y(i,\mathrm{j},\mathrm{k})] = S(i,t_j)(1 - s_j p) \approx s_i(1 - e^{-|U[t,t_j+1]|/L})$. The interference between b_i and b_j is $\mathrm{Z}(\mathrm{i}:\mathrm{j}) = Y(i,\mathrm{j},\mathrm{T}) + Y(j,\mathrm{i},\mathrm{T})$, which gives the desired symmetric property $\mathrm{Z}(\mathrm{i}:\mathrm{j}) = \mathrm{Z}(\mathrm{j}:\mathrm{i})$.

As a generalization of Equation 4, we introduce the interference function

$$\mathcal{Z}(L,T) = \sum_{i=1}^{n} s_i \left(\sum_{k=1}^{N_i} 1 - e^{-u(i,k)/L}\right). \tag{8}$$

which describes, in theory, the expected miss rate of T for a direct-mapped cache of any size.

3.5 Intervening traces

When analyzing trace T, we define an intervening trace T' as a continuous sequence of blocks during T that have no addresses in common with T. Thus, we can view (i) operating system calls, (ii) operating system interrupts and *(iii)* process switches due to multi-tasking, all as intervening traces. For simplicity, we assume the intervening trace T' runs to completion without being interrupted itself. The notation $\mathrm{T} < T' >$ indicates that T' was an intervening trace sometime during T.

If T' has U' unique addresses, it increases the size of all pending gaps by U'. An intervening trace interrupts L gaps. Let G = the sum of all gap lengths. If no information is known a priori as to when the intervening trace will occur in T, e.g. a hardware timer interrupt, the probability of interrupting any gap g is proportional to the length of g, Pr[interrupting gap g] = (length of g) $* L/G$. For the kth gap of b_i, we define the r.v.

$V_i(k,T') \equiv X_i(k,T < T' >) - X_i(k,T)$ = # extra cache misses in kth gap of b_i due to T'. Then the expected value of $V_i(k)$ is

$$\mathrm{E}[V_i(k)] = (g(i,k)L/G)\, s_i (e^{(u(i,k)+U')/L} - e^{u(i,k)/L}) \tag{9}$$

The expected number of additional misses due to T' is the sum of $\mathrm{E}[V_i(k)]$ over all gaps.

3.6 An approximate lower bound of misses

Because Equation 4 represents the average number of cache misses for T over all address mappings, there must exist "good" mappings that result in fewer misses. How well might a good address mapping perform? Unfortunately, determining the miss rate of the optimal mapping is an NP-hard problem. However, using the gap model we can estimate a lower bound.

As done in [McF89], we use the optimal cache-line replacement strategy, OPT, to derive a lower bound. On a cache miss, OPT replaces the line that will be accessed furthest into the future. Although OPT is impossible to implement in practice (as it requires knowledge of the future and a fully associative cache to boot), the miss rate of OPT forms a convenient comparison point. We can overestimate the number of misses for OPT, by discarding all terms in Equation 4 for gaps of size less than L, as seen by following lemma.

Lemma 1 *If block b_i has a gap g of size u, using the OPT replacement strategy, b_i survives through g if $s_i + u \leq L$.*

Proof: Let U be the set of addresses referenced in gap g. Assume g starts at time t. Rank (with labels 1, 2, 3, ..., L) each of the addresses resident at t based on the time of the next access, with lower ranks for sooner accesses. Let r be the number of addresses in U are already resident and $x = u - r$ of the addresses are not resident at t. At the end of g, OPT will have replaced the x addresses of largest rank; thus, the $L - x$ addresses with lowest rank must survive through g. The last address of b_i (the address of b_i with highest rank) has rank $r + s_i$. From the lemma, $s_i + u = s_i + x + r \leq L$. Thus, $s_i + r \leq L - x$, and b_i has low enough rank to survive through g. □

We note a block can survive gaps with size greater than L, if OPT replaces parts of U as the gap progresses. As an example, assume L = 2, and all blocks have size of one. In the following trace, block b_1 survives the entire trace including two large gaps.

$$b_1, b_2, b_3, b_1, b_4, b_5, b_6, b_7, b_1, b_2, b_3, b_1.$$

3.7 The image of a trace

In this section, we describe methods to compactly summarize the gaps in a trace. Our idea is to categorize each gap as either "short" or "long" depending on whether the size of the gap (number of unique addresses in the gap) is smaller-than, equal-to or larger-than the cache size, L.

Equation 4 shows that the ratio of the gap size to L determines the miss rate. For example, if the gap size is 5L, then the block probably will not survive the gap as $e^{-5} < .7\%$. Thus, for each block, if we know how many gaps are of each size, we can calculate Equation 4 precisely. However, this information potentially requires keeping information about many gaps sizes.

Instead, we adjust all gap sizes by rounding them to the nearest power of 2, and we count the number of gaps of each adjusted sizes. For a trace of length N, there are $\log_2 N$ adjusted gap sizes. Thus, we need $\log_2 N$ integers for each block, or $n \log_2 N$ integers in total to summarize a trace. As we alluded earlier, we can group large gaps with sizes $\geq 5L$ together. As it is seems unlikely we

will see caches with more than 2 x 10^5 cache lines (not bytes) in the near future, in practice, we can lump together all gaps of size greater than 10^6. Thus, we need only $\log_2 10^6 \approx 30$ adjusted gap sizes, meaning we need only $(30n)$ integers to summarize the trace. For example, if $N = 10^9$ and $n = 10^4$, we need $30 \times 10^4 = 3 \times 10^5$ integers, a space reduction by a factor of 3×10^3. Furthermore, if many blocks have gap sizes with zero counts, we can save more space by listing just the non-zero entries for each block (analogous to an adjacency list for representing sparse graphs). This method could reduce the space needed by another factor of 2–1000 times.

If handling intervening traces is important, we need to store both the length and size of each gap. As before we can adjust each gap length by rounding to the nearest power of 2, so that we need $n(\log_2 N)^2$ integers. As before, if many entries are zero, we can list just the non-zero entries.

If is not important to know which blocks are causing the misses, we can lump all blocks together and simply count adjusted gap sizes weighted by block sizes. That is, if block b_i has a gap with adjusted size 2^g, then we add s_i to the gap count for size 2^g. In this manner, only $\log_2 N$ integers are needed. As this is at most 40 integers in practice, for better accuracy, we might round gap sizes to the nearest power of $\sqrt{2}$ or even $\sqrt[4]{2}$, which would double or quadruple the space requirement. To capture how the miss rates changes as the trace progresses, we can split T into shorter pieces and summarize the pieces individually.

Finally, we note that our gap model expands upon the LRU stack model of an address trace [Spi77] [RS77]. In their model, they record the probability that the next address in the trace will be the mth most recent address. This corresponds to a gap of size m in our model. However, our derivation points out the expected effect on the miss rate of such an occurrence.

3.8 Bounds on the variance of $X(T)$

Determining the variance of $X(T)$ directly is difficult because the terms $X_i(k)$ are not independent. Although we can derive an analytic formula for $\mathrm{Var}[X(T)]$, directly evaluating this formula is computationally intractable as it involves many, many terms (typically, $\gg 10^8$), forcing us to settle for bounding the value of $\mathrm{Var}[X(T)]$.

Before getting involved in the mathematics, recall the goal of this section is to estimate how much the miss rate varies. We shall use the variance as a yardstick. However, as we can only estimate the variance, our methods are approximate at best. As such, we shall sacrifice accuracy and rigor for simplicity and intuition when possible. For any r.v. $X = X_1 + X_2 + \ldots + X_n$, we have

$$\begin{aligned}
\mathrm{Var}[X] &= \mathrm{E}[(X - \bar{X})^2] \\
&= \sum_{i=1}^{n} \mathrm{Var}[X_i] + 2\mathrm{E}[\sum_{i=2}^{n} \sum_{j=1}^{i-1} (X_i - \bar{X}_i)(X_j - \bar{X}_j)]. && (10) \\
&= \sum_{i=1}^{n} \mathrm{Var}[X_i] + 2\mathrm{E}[\sum_{i=2}^{n} \sum_{j=1}^{i-1} (X_i X_j - \bar{X}_i \bar{X}_j)]. && (11) \\
&= \text{sum of variances} + \text{cross terms}
\end{aligned}$$

$$\mathrm{Var}[X] > \sum_{i=1}^{n} \mathrm{Var}[X_i], \qquad \text{cross terms} \geq 0 \tag{12}$$

If the X_i terms are pairwise independent (namely, $\mathrm{E}[X_i X_j] = \mathrm{E}[X_i]\mathrm{E}[X_j]$) then the double sum of cross terms is zero. In our case, X_i and X_j represent the number of cache misses when accessing blocks b_i and b_j at times t; and t_j, respectively. Unfortunately, these cross terms are dependent and there are $O(n^2)$ of them. Directly calculating each of these terms is somewhat involved. Consequently, we shall instead derive upper and lower bounds of $\mathrm{Var}[X]$.

The cross terms X;, X_j are positively correlated if $\mathrm{E}[(X_i - \bar{X}_i)(X_j - \bar{X}_j)] > 0$ or alternatively, $\mathrm{E}[X_i X_j] > \mathrm{E}[X_i]\mathrm{E}[X_j]$. In this case, Equation *12* shows we can underestimate $\mathrm{Var}[X]$ by summing the variances of the individual X_i. Informally, if knowing that X_i is greater than its average value means that we expect X_j to be greater than its average value, then we shall consider X_i and X_j to be positively correlated. We believe the sum of cross terms is positive so that discarding them gives a lower bound.

Both loop iterations are gaps of b;. If we know that b_j or b_k collide with b_i then the $X_i(k_i)$ and $X_i(k_i')$ terms will both have larger than average values, showing they are dependent. We can also, view the same references in terms of gaps of b_j. Here, we see b; and b_k in the b_j gaps. Thus, the corresponding $X_j(k_j)$ and $X_j(k_j')$ terms are related. Finally, the $X_i(k_i)$ and $X_j(k_j)$ terms are dependent because if b_i and b_j collide, then both terms will tend to be larger. (All four terms are dependent, in fact.) Overall, we expect the terms for the same block to be related for this reason and many of the terms for different blocks to be dependent if references to these blocks are interleaved as in major loop calls.

We may discard terms for gaps with $\geq 10L$ unique addresses because it is unlikely (prob $< 5 \times 10^{-5}$) a block can survive the gap. The term $\mathrm{E}[(X - \bar{X})]$ will be quite small.

We can underestimate $\mathrm{Var}[X_i(k)]$ by considering each address in a block, independently. We define the r.v. $X_{i[j]}(k) = 1$ if and only the jth address in b_i causes a miss at the end of the kth gap. Using Equation 3 with $s_i = 1$, gives the value of $X_{i[j]}(k) = 1$. We know $X_{i[j]}(k)$ and $X_{i[j']}(k)$ are positively correlated, because if the jth address of b_i is replaced during the gap, it is likely the j'th address was also replaced. Thus,

$$\mathrm{Var}[X_i(k)] > \sum_{j=1}^{s_i} \mathrm{Var}[X_{i[j]}(k)] = s_i \left(1 - e^{-u(i,k)}\right) e^{-u(i,k)},$$

which plugged into Equation *12* gives

$$\mathrm{Var}[X(T)] > \sum_{i=1}^{n} s_i \sum_{k=1}^{N_i} \left[\left(1 - e^{-u(i,k)/L}\right) e^{-u(i,k)/L}\right].$$

We can get a crude estimate of the variance by considering the dominant loops in the trace, as a significant fraction of the execution trace will consist of iterations from a few major loops. Loop iterations will appear as consecutive (nearly) identical gaps. In practice, caches are much larger than a loop body, so that we expect the loop to survive intact from iteration to iteration. Thus,

much of the variance will come from the case where two (or more) blocks collide in a frequently executed loop. E.g. the gap model predicts no collisions but some occur. In the following, we assume that dominant loops have been identified by some means.

time:	t	$t+1$	$t+2$	$t+3$	...	t'
block:	b_i	b_j	b_k	b_l	...	b_i

Consider a loop that is a gap of b; which contains b_j once, as shown above. (If a gap contains more than one occurrence of b_j, we consider only the first.) We restrict ourselves to the case where b_i and b_j collide. Let X; and X_j represent the number of misses on the second loop iteration when accessing b; and b_j. We shall evaluate the cross term in Equation 11, $X_iX_j - \bar{X}_i\bar{X}_j$. For simplicity, we assume the gap size is less than $L/10$ so that we can ignore the product of the means because $\bar{X}_i\bar{X}_j < e^{1/10}e^{1/10} < .01 \approx 0$. Without loss of generality, if $s_i \geq s_j$, there are three cases for b_i and b_j: no collision, partial collision, and full collision where b_i completely "covers" b_j.

$$
\begin{aligned}
\mathrm{E}[X_iX_j] &= \frac{1}{\mathrm{L}}\left[(L - s_i - s_j + 1)(0 * 0) + 2\sum_{\mathrm{k=1}}^{s_j-1}(\mathrm{k} * \mathrm{k}) + (s_i - s_j + 1)(s_j * s_j)\right] \\
&= \text{no collision} + \text{partial collision} + \text{full collision} \\
&= \frac{1}{L}\left[s_j(s_j - 1) + (s_i - s_j + 1)(s_j^2)\right] \qquad (13) \\
&\approx \frac{(s_i - s_j + 2)s_j^2}{L}, \qquad s_j \geq 5
\end{aligned}
$$

The above cross term applies to each X_i and independently to each X_j from each loop iteration. Thus, if there are m iterations of a particular loop, there are m^2 identical X_i X_j terms for each i and j due to the cross product of m X_i terms and m X_j terms. We calculate similar terms for all pairs of variables involved in the loop. Finally, we apply these terms over all major loops of the program. Formally, let ϕ represent a loop from T that makes $|\phi|$ iterations. Let i, $\mathrm{j} \in \phi$ mean that blocks b_i and b_j are referenced in each iteration of ϕ. Then using Equation 13, the sum of the loop cross terms is

$$
\frac{1}{L}\sum_{\phi\in T}|\phi|\sum_{i,j\in\phi}\left[s_j(s_j - 1) + (s_i - s_j + 1)(s_j^2)\right] \approx \frac{1}{L}\sum_{\phi\in T}|\phi|\sum_{i,j\in\phi}\left[(s_i - s_j + 2)(s_j^2)\right]
$$

4 A fully associative cache

We now consider the expected number of misses for a fully associative cache using random replacement. With this scheme, a random line is replaced on a cache miss. The use of random replacement simplifies our analysis because block collisions are independent of the address mapping. A fully associative cache dynamically adapts to the trace, so that recently accessed lines are likely to be in the cache, independent of the trace. We shall use a working-set model [Den68] not the gap model. Previous studies [Smi78] have shown that fully associative caches have significantly lower

miss rates than direct-mapped caches. The results of this section also apply to a-way set-associative caches for a ≥ 4. In practice, a-way set-associative caches, for a ≥ 4 have similar performance to fully associative caches. However, full associativity is significantly easier to analyze than a-way set-associativity.

To analyze a fully associative cache, we consider each address of P individually, because any address can collide with any other address. In this case, we partition P into n addresses (not blocks), $b_1, b_2, \ldots, b_n$, where each b_i is a cache line's worth of object code.

To gain some insight into how the behavior of a direct-mapped cache differs from that of a fully associative cache, we shall analyze the expected number of misses for a cyclic trace of w different addresses $b_1, b_2, \ldots, b_{w-1}, b_w, b_1, b_2, \ldots, b_{w-1}, b_w, b_1, \ldots$ on an initially empty cache. Here, w is meant to represent the effective working set size. We call $b_1, b_2, \ldots, b_{w-1}, b_w$ a cycle. Intuitively, if $w \leq L$, we expect all addresses will eventually reside in the cache; if $w > L$, the trace cannot fit in the cache and at least $w - L$ blocks must miss each cycle.

In a direct-mapped cache, two blocks b_i and b_j either always conflict or never conflict. Thus, the expected miss rate from these blocks over all mappings is $1/L$ for each reference. In contrast, on a fully associative cache, b_1 and b_2 will probably be placed in different cache locations initially resulting in no further collisions. However, there is a chance b_1 and b_2 might collide one or more times before both become resident.

4.1 Small working sets, $w \leq L$

We now solve for the expected number of misses when dealing with an arbitrary trace of w distinct addresses where $w \leq L$. We shall categorize the state of the cache by u, the number of unique addresses it contains. Note that u must monotonically increase, because on a cache miss, we either replace an old address leaving u unchanged, or we load the new address into an empty cache line incrementing u. Once u = w, there will be no further misses.

Let $\bar{X}(n \rightarrow n+1)$ = the expected number of misses starting with u = n until $u = n+1$. For example, $\bar{X}[0 \rightarrow 1] = 1$, as the first access misses and becomes resiclent. However, if w = L and u = w − 1, to increment u to w requires filling the one empty line rather than replacing any of the $L-1$ resident addresses on a miss. As we shall see, $\bar{X}[n \rightarrow n+1] = L/(L-n)$, so that $\bar{X}[L-1 \rightarrow L] = L$ misses are needed on average to randomly fill the last slot. Note that we ignore cache hits at this point, and it will be the case that many hits occur between misses. We have

$$\begin{aligned}
\bar{X}(n \rightarrow n+1) &= \sum_{i=1}^{\infty} i \cdot \Pr[\,\bar{X}(n \rightarrow n+1) = i\,] \\
&= \sum_{i=1}^{\infty} \Pr[\,\bar{X}(n \rightarrow n+1) \geq i\,] \\
&= \sum_{i=1}^{\infty} \left(\frac{n}{L}\right)^{i-1} = \frac{L}{L-n}. \qquad (14)
\end{aligned}$$

$a = w/L$	.1	.25	.33	.50	.66	.75	.80	.90
misses: total / compulsory	1.05	1.15	1.21	1.39	1.63	1.85	2.01	2.56

Table 2: Total misses for $w = \alpha L$, as a ratio of compulsory misses

We have $\Pr[\,\bar{X}(n \to n+1) \geq i\,] = (n/L)^{i-1}$ because we need $(i-1)$ consecutive misses that replace one of the n resident addresses in the cache. In general $\bar{X}(a \to b) = \sum_{n=a}^{b-1} \bar{X}(n \to n+1)$. Using $H_n \equiv \sum_{i=1}^{n} 1/i \approx \ln n + .577 + 1/(2n) - 1/(12n^2)$ [GKP89],

$$\begin{aligned}
\bar{X}(a \to b) &= \sum_{i=a}^{b-1} \bar{X}(i \to i+1) = \sum_{i=a}^{b-1} \frac{L}{L-i} \\
&= L\left(\frac{1}{L-a} + \frac{1}{L-a-1} + \cdots + \frac{1}{L-b+1}\right) = L(H_{L-a} - H_{L-b}) \qquad (15) \\
&\approx L\left[\ln\frac{L-a}{L-b} + \frac{(b-a)}{2(L-a)(L-b)}\right] \\
&\approx L\ln\frac{L-a}{L-b}, \qquad 10 \leq b \leq L-2 \qquad (16)
\end{aligned}$$

Thus, if $a = 0$ and $b = L$, we expect $\approx L \ln L$ misses from Equation 15. Alternatively, if $a = 0$ and $b = w = \alpha L$, each address has $(1/\alpha)(\ln(1/1-a)$ misses on average. Table 2 shows that for $w \leq 50\% L$, compulsory misses account for most of the misses, indicating that cache quickly captures the working set if the working set is smaller than the cache. In practice, caches will frequently be much larger than the working set, so that $\alpha \ll 1$. For $\alpha \ll 1$, use of the Taylor expansion $\ln(1+x) = x - x^2/2 + \ldots$, for $0 \leq x \ll 1$, shows that the ratio of total misses over compulsory misses is roughly $1 + \alpha/2(1-a)$.

The preceding analysis only considers cache misses. We now calculate the expected number of total references, including cache hits, for the working set to become resident, assuming addresses are accessed randomly from the w addresses in the working set. Let $\bar{R}(a \to b)$ = the expected number of references starting with $u = a$ until $u = b$ and let $\bar{Q}(a)$ be the expected number of references until a cache miss occurs if there are a resident addresses in the cache.

$$\begin{aligned}
\bar{Q}(a) &= \sum_{i=1}^{\infty} i \cdot \Pr[\,\bar{Q}(a) = i\,] \\
&= \sum_{i=1}^{\infty} \Pr[\,\bar{Q}(a) \geq i\,] \\
&= \sum_{i=1}^{\infty} \left(\frac{a}{w}\right)^{i-1} = \frac{w}{w-a}.
\end{aligned} \qquad (17)$$

We have $\Pr[\,\bar{Q}(a) \geq i\,] = (a/w)^{i-1}$ because we need $i-1$ consecutive references to one of the a resident addresses in the cache, giving $i-1$ consecutive cache hits. Combining this result with Equation 14 gives

$$
\begin{aligned}
\bar{R}(a \to b) &= \sum_{i=a}^{b-1} \bar{R}(i \to i+1) = \sum_{i=a}^{b-1} \bar{X}(i \to i+1)\bar{Q}(i \to i+1) = \sum_{i=a}^{b-1} \left(\frac{L}{L-i}\right)\left(\frac{w}{w-i}\right) \\
&= \frac{Lw}{L-w} \sum_{i=a}^{b-1} \left(\frac{1}{w-i} - \frac{1}{L-i}\right), \qquad w \neq L \\
&= \frac{Lw}{L-w}(H_{w-a} - H_{w-b} - H_{L-a} + H_{L-b}) \qquad (18) \\
&\approx \frac{Lw}{L-w} \ln\left[\frac{(w-a)(L-b)}{(w-b)(L-a)}\right], \qquad b < w < L \qquad (19) \\
&\approx \frac{Lw}{L-w} \ln\left[\frac{(w-a)(L-w)}{(L-a)}\right], \qquad 10 \leq b = w < L. \qquad (20)
\end{aligned}
$$

In most cases, b = w as we assume the entire working set becomes resident. In particular, if a = 0 and b = w = $L-1$, there are $(L-1)^2$ references on average before the working set is resident. If w = αL, where $\alpha \ll 1$, then $\bar{R}(0 \to \mathrm{w}) \approx (w/(1-\alpha))\ln(w(1-\mathrm{a}))$.

The above analysis makes the implicit assumption that every block in the working set is accessed regularly, such as a cyclic trace, so that the entire working set eventually becomes resident. In practice, traces will rarely be so cooperative. To handle irregular access to the working set, we shall augment T with extra references.

We prove that adding blocks to a trace cannot decrease the expected number of misses. (Note, this process may decrease the miss rate, as the trace becomes longer.) We denote an insertion as insert(T, b, t) $\equiv$ add a reference to b_x to T immediately after B_t before B_{t+1}. Thus, if T = $B_1, \ldots, B_t, B_{t+1}, \ldots, B_N$, then insert($T$, b, t) = $B_1, \ldots, B_t, b_x, B_{t+1}, B_N$. As before $\bar{X}(T)$ is the expected number of misses on trace T.

Lemma 2 For any *trace* T, let T' = *insert*(T, b, t). *Then,* $\bar{X}(T) \leq \bar{X}(T')$.

Proof: There are two cases.

(i) Block b is present in the cache at time t. The extra reference to b_x is a cache hit so $\bar{X}(T) = \bar{X}(T')$.

(ii) Block b replaces block b_r at time t in the cache. We consider the misses blamed on b_x and b_r.

Misses blamed on b_r — In trace T, if b_r survives until its next reference, we get an extra miss in T' on the next reference to b. If b_r would have been replaced anyways before its next reference, the extra b_x causes no change. In both cases, we have $\bar{X}(T) \leq \bar{X}(T')$.

Misses blamed on b_x — In T, the extra b_x either prefetches b_x if b_x survives until its next reference or generates an extra miss if b_x is replaced by its next reference. In either case, $\bar{X}(T) \leq \bar{X}(T')$. □

---

[1] If $a = 0$, and $b = w = L$, we get $\bar{R}(0 \to L) = \sum_{i=0}^{L-1} L^2/(L-i)^2 \approx L^2\pi^2/6 \approx 1.6L^2$. We have used $\sum_{i=1}^{\infty} 1/i^2 = \pi^2/6$.

Theorem 1 For any *trace* T, let T_2 be the resulting truce *after an arbitrary* number of insertions on T. Then, $\bar{X}(T) \leq \bar{X}(T_2)$.

Proof: By repeated single insertions we can convert T to T_2. The theorem follows by repeated application of the lemma. □

We use this theorem to get bounds on the misses for traces with irregular reference patterns. For example, let b_ϕ represent the blocks in a loop body and consider the trace (fragment) To = $b_1, b_2, b_\phi, b_\phi, b_\phi, b_1, b_2, b_\phi, b_\phi, b_2, b_\phi, b_\phi$ in which b_1 and b_2 are accessed sporadically. At the end of T_0, it is not clear whether b_1 is likely to be resident, because the last reference to b_1 occur By inserting references to b_1 and b_2 before b_ϕ when necessary, we can convert T_0 to T_0', a sequence of cyclic traces so that Equation 15 applies. If w for T_0 is small and the cache is initially empty, we get the bounds w $\leq \bar{X}(T_0) \leq \bar{X}(T_0') = L\ln(L/L -$ w). For w $< L/10$, Table 2 shows that w $\approx L\ln(L/L -$ w), indicating our bounds are quite tight. Other trace fragments can be handled similarly.

4.2 Analyzing the trace, w $\leq$ L

To determine the expected number of misses on T, we assume the T has been partitioned by some means into sub-traces $T_1, \ldots, T_N$, such that each sub-traces has a different working set than its neighbors. E.g each time the working set changes, we get a new working set. The working set for T_t is W_t; its size (in cache lines) is w_t. We assume $w_t \leq$ L, and that after each subtrace, the entire working set W_i is resident. In this section (Section 4.2), we clenote the time immediately before subtrace T_t as time t, e.g. it is as if each subtrace requires one unit of time. The notation $b_j \in W_t$ means that address b_j is part of working set W_t.

Within each subtrace T_t, we have $\bar{X}(T_t) = \bar{X}(a_t \rightarrow w_t)$, where a_t is the expected number of addresses of W_i already resident at the start of T_t. We use the gap moclel to determine at. We define the random variables

$$S(i,\mathrm{t}) = \begin{cases} 1 & b_i \text{ in the cache at time t} \\ 0 & b_i \text{ not in the cache at time t} \end{cases},$$

and

$$S(W_i, t) \equiv \sum_{b_j \in W_i} S(j,\mathrm{t}).$$

Thus, $S(W_i, t)$ is the number of addresses of W_i that are resiclent at time t. Let $b_i \in W_t$ as shown below. The probability that address b_i will survive through subtrace T_{t+1} is $(1 - w_{t+1}/L)$, assuming none of W_{t+1} was resident before T_{t+1}. However, if αw_{t+1} addresses of W_{t+1} survived from a previous subtrace through T_t, then during T_{t+1} only $(1-\alpha)w_{t+1}$ addresses will be reloaded into the cache, and the probability of b_i surviving though T_{t+1} is $(1 - (1-\alpha)w_{t+1}/L)$.

time:	t	$t+1$	
blocks:	$\cdots b_i \cdots$		$b_{j1}\ b_{j2} \cdots b_{jm} \cdots$
subtrace:	$\mid \leftarrow T_t \rightarrow \mid$		$\mid \longleftarrow T_{t+1} \longrightarrow \mid$

Thus, the number of addresses of W_t already resident before T_t is $a_t = \bar{S}(W_t, t)$, which is defined by the recurrences

$$\begin{aligned} E[S(i,t+1)] &= 1, & b_i \in W_t \\ E[S(j,t+1)] &= E[S(j,t+1)] * \left(1 - \frac{w_t - E[S(W_t,t)]}{L}\right), & b_j \notin W_t. \end{aligned} \tag{21}$$

As an example, consider the case where two working sets, W_t and W_{t+1} overlap in their transition. We model the overlap in a separate **subtrace** that references the large working set $W_t \cup W_{t+1}$ that has been warm-started with all of W_t resident. Then during the transition, we expect $\bar{X}(w_1 \rightarrow w_1 + w_2) \approx L \ln \frac{L-w_1}{L-(w_1+w_2)}$ misses from Equation 16. After the transition, W_{t+1} is already resident and no further misses are encountered.

4.3 Working sets larger than the cache, w > L

[Originally we tried to analyze fully-associative caches using the following analysis, however it yields erroneous results when w $\leq$ L. Fortunately, it yields reasonably accurate results for w > L. In practice, caches are becoming large enough so that w > L is increasingly unlikely.]

If w $\gg$ L (say w > $5L$), we can again apply the gap model, **because** few addresses will survive their gaps. In particular, an address will survive with probability $e^{-w/L}$, giving a miss rate of $1 - e^{-w/L}$. (A brief simulation indicates this model is **quite** accurate for w $\geq$ 3L.) The probability of a miss when accessing b_i at time t depends in part on the likelihood of b_i already residing in the cache. As before, we define the random variable

$$S(i,t) = \begin{cases} 1 & b_i \text{ in the cache at time t} \\ 0 & b_i \text{ not in the cache at time t} \end{cases}$$

If w $\gg$ L, the probability of an address surviving through a cycle until its next reference is small, and we may assume with little loss in accuracy that $S(i,t)$ is independent of $S(j,t)$ for any i and j. Then, if $B_t = b_i$, we have

$$E[S(i,t+1)] = 1 \tag{22}$$

$$\begin{aligned} E[S(j,t+1)] &= E[S(j,t)] * \left(1 - \frac{1 - E[S(i,t)]}{L}\right) \\ &= E[S(j,t)] * \left(\frac{L - 1 + E[S(i,t)]}{L}\right) \qquad i \neq j \end{aligned} \tag{23}$$

$$\tag{24}$$

In Equation 23, the term $(1 - E[S(i,t])$ is the probability that b_i is not in the cache and the term $(1 - E[S(i,t)])/L$ is the probability b_i replaces b_j in the cache. For the cyclic trace, we assume the cache reaches a stationary state in which $S(j,t)$ depends only on how long ago b_j was referenced, and we can treat all blocks identically, except they are shifted in time. We define $\bar{S}(t)$ = expected value of $S(j,t)$ where b_j was referenced t time units ago. In the cyclic trace, each address has a **gap**

w	32	36	40	44	48	52	56	60
calculatedx	0.0%	16.9%	33.7%	46.6%	56.2%	64.0%	70.0%	75.1%
measured	- %	23.6%	38.8%	50.3%	58.9%	66.0%	71.7%	76.4%
w	64	68	72	76	80	84	88	92
calculatedx	79.0%	82.3%	85.0%	87.1%	88.9%	90.7%	91.9%	93.1%
measured	79.8%	83.0%	85.5%	87.7%	89.5%	91.0%	92.4%	93.3%
w	96	100	104	108	112	116	120	124
calculated $\bar{X}$	94.0%	94.9%	95.5%	96.1%	96.7%	97.0%	97.6%	97.9%
measured	94.1%	95.0%	95.6%	96.2%	96.7%	97.1%	97.5%	97.8%

Table 3: Comparison of predicted and measured miss rates for L = 32 on a cyclic trace of w addresses.

of length w $-$ 1, so the currently referenced address has probability $\bar{S}(w-1)$ of still being resident in the cache. Using Equations 22 and 23, we can solve for $\bar{S}(w-1)$.

$$\begin{aligned}
\bar{S}(0) &= 1 \\
\bar{S}(t) &= \bar{S}(t-1) * (1 - \frac{1-\bar{S}(w-1)}{L}) \\
\bar{S}(w-1) &= \bar{S}(w-2) * (1 - \frac{1-\bar{S}(w-1)}{L}) = \bar{S}(0) * \left(1 - \frac{1-\bar{S}(w-1)}{L}\right)^{w-1} \\
&\approx e^{-(w-1)(1-\bar{S}(w-1))/L}
\end{aligned} \tag{25}$$

Taking the logarithm of Equation 25 and rearranging gives a transcendental equation which defines the expected miss rate, X, which is $1 - \bar{S}(w-1)$.

$$L \ln(1 - \bar{X}) = -(w-1)\bar{X} \tag{26}$$

After solving for X iteratively, Table 3 shows that the calculated miss rate $\bar{X}$ from Equation 26 matches well with simulation results for w $\geq 1.5L$. For Table 3, we measured the miss rate of a fully associative cache on 100 cycles of the trace $1, 2, 3, \ldots,$ w. Because random replacement is used, we can only calculate the expected miss rate. In We chose L = 32 in Table 3, because in practice, TLB's are often fully associative and TLB's are frequently of this size.

4.4 Intervening traces

Assume at some point during trace T the working set size is w and u of these addresses are resident. If an intervening trace T' occurs with u' unique addresses that all become resident, it reduces the expected number of resident addresses of T to $u(1 - u'/L)$, using the same reasoning as for Equation 1. After T' ends, we expect $\bar{X}[u(1 - u'/L) \rightarrow w]$ more misses until the working set from T become fully resident.

5 Summary and future work

We have presented formulas for the expected number of cache misses on a trace T over all possible address mappings of the underlying program P for direct-mapped and fully associative caches. For direct-mapped caches, we used a gap model; for fully associative caches, we use a working set model. As a result, our analysis (i) provides a single meaningful number representing for the miss rate of a cache for a specific benchmark, (ii) naturally models the effects of intervening traces, and (iii) is likely to be reasonably insensitive to small changes in the program or program input. The gap model provides a quantitative estimate of how blocks interefere with one another in the cache, provides a compactly trace summary and models a wide variety of cache-miss-rate versus cache-size curves.

The obvious omission in this paper is lack of experimental validation. We shall test our models for direct-mapped and associative caches empirically on the SPEC benchmarks in a future paper. Also, we were unable to develop a model for a 2-way associative caches remains an unsolved problem. We also plan to use our model to determine good address mappings during linking. For a direct-mapped cache, a good mapping must perform well for mid-sized gaps. The gap model provides this information.

6 Acknowledgements

We thank Steve Crago for his feedback on this paper.

References

[Den68] Peter J. Denning. The working set model for program behavior. Communications of the ACM, 11(5):323–333, May 1968.

[GKP89] R. Graham, D. E. Knuth, and O. Patashnik. Concrete *Mathematics*. Addison-Wesley, Reading, MA, 1989.

[McF89] Scott McFarling. Program optimization for instruction caches. In ASPLOS-111, Boston, MA, April 3–6 1989. ACM/IEEE.

[RS77] Turner Rollins and Bill Strecker. Use of the lru stack depth distribution for simulation of paging behavior. Communications of the ACM, 20(7):795–798, November 1977.

[Smi78] Alan Jay Smith. A comparative study of set associative memory mapping algorithms and their use for cache and main memory. IEEE *Transactions* on *Software* Engineering, SE-4(2):121–130, March 1978.

[Spi77] J.R. Spirn. *Program* Behavior: Models and Measurements. Operating and *Programming Systems Series*. Elsevier Scientific Publishing Co., Inc., New York, 1977.