Trace Compaction Using Cache Filtering With Blocking

Anant Agarwal

Technical Report No. CSL-TR-88-347

December 1987

This technical report was supported by Defense Advanced Research Projects Agency under Contract No. MDA903-83-C-0335.

Trace Compaction Using Cache Filtering with Blocking

Anant Agarwal

Technical Report No. CSL-TR-88-347

December 1987

Computer Systems Laboratory
Department of Electrical Engineering
Stanford University
Stanford, CA 94305

Abstract

Trace-driven simulation is a popular method of estimating the performance of cache memories, translation lookaside buffers, and paging schemes. Because the cost of trace-driven simulation is directly proportional to trace length, reducing the number of references in the trace significantly impacts simulation time. This paper concentrates on trace-driven simulation for cache analysis. A technique called cache filtermg with blocking is presented that compresses traces by exploiting both the temporal and spatial locality in the trace. Experiment@ results show that this scheme can reduce trace length by nearly two orders of magnitude while introducing less than 15% error in cache miss rate estimates.

Key Words and Phrases: trace-driven simulation, cache performance, temporal locality, spatial locality, block filler, cache filler, and trace compaction.

Contents

1. Introduction 3

2. Cache Filter 5

3. Block Filter 6

3.1 Implementation of the Cache and Block Filters 10

3.2 Miss Rate Estimation 11

4. Results 12

4.1 Sensitivity Analysis 12

5. Conclusions 15

References 16

1 Introduction

A trace driven simulation (TDS) study involves evaluating a model of a cache using a set of addresses generated by a program as the input. TDS is popular for cache studies because it is flexible, easy to use, and does not require special hardware. The chief problem with this method is its computation expense. Large caches and multiprogramming workloads exacerbate the problem because much larger traces, often over a million references, are necessary to obtain reliable cache performance estimates [1]. Since the cost of trace-driven simulation is directly related to the to the size of the input data, reducing the number of references has a tremendous impact on simulation time.

There are two ways of reducing simulation time. Trace sampling methods [1] use several discrete samples of program execution instead of a single, long, continuous trace without significant loss of accuracy, but at a much reduced cost. Still, the trace samples are hundreds of thousands of references long, and if each sample could be compressed further, the benefits would be proportionally greater. Trace compaction schemes can achieve just this. Trace compaction has the added advantage that it eases the trace storage space requirements.

The key idea behind trace compaction is that not all references carry the same amount of useful information. By retaining only those references that contribute some new information, traces can be stripped down in size by an order of magnitude or more. For example, program references display the property of temporal locality [2] by repeatedly reusing references from a working set over a given period of time; discarding repeat references in this time interval can retain most information necessary for cache studies.

Two earlier studies used this property of programs in their trace compaction efforts. Smith [3] described two methods of reducing trace length. The first scheme, called ***stack*** deletion, discards all references that are hits to the top D levels of the LRU stack of data. Because the hit rate to the top stack levels is very high, and because any reasonable memory management scheme will succeed in retaining these frequently used references in main memory, little if any information is lost by discarding these repeated references for memory management studies. The second scheme ***is*** the ***snapshot method,*** which records the set of references in memory at regular time intervals. The rationale for this scheme is that the set of references in use does not change rapidly with time. So, recording the set every T references will not result in significant information loss. Unfortunately, Smith's scheme is not flexible because its assumes that the page size (or block size) is fixed throughout the analyses. Furthermore, the schemes are primarily suited for paging studies and no data is provided on the suitability of the schemes to cache studies.

Puzak [4] proposed a scheme called ***trace stripping*** with particular application to cache studies. A direct-mapped cache (called a cache filter) is simulated and a trace of the references that miss is generated. This method has the appealing property that compaction does not introduce any error in cache simulations if the number of sets in the given cache is not less than the number of sets in the cache filter, provided the block size is kept constant. As in Smith's techniques, the constant block size requirement limits the scheme's flexibility. Although it is mentioned that the stripped trace can be used to derive rough miss rate estimates for block sizes larger than the cache filter block size, smaller block sizes are not considered. This is a problem because when the number of sets of the final cache organizations cannot be less than the corresponding cache filter parameter, a small direct-mapped cache must be used as the filter for maximum flexibility, which results in poor compaction. Consequently, one uses large block sizes in the cache filter to improve the compaction, but this precludes the simulation of smaller block sizes. It is also not possible to derive sub-block statistics. (A sub-block is the portion of a block that is fetched into the cache on a miss.)

The reason why earlier schemes failed to achieve significant compaction while allowing flexibility in cache studies was that compaction was achieved primarily by exploiting the ***temporal*** locality of program blocks, and not providing a model for the ***spatial*** locality. Spatial locality is the property by which programs tend to reuse references that are in a close spatial neighborhood in a given time interval. As we demonstrate later, obtaining results for smaller block sizes requires explicit knowledge of the compaction components due to both temporal ***and*** spatial locality.

Our technique uses separate models for temporal and spatial locality to yield increased compaction and allows caches with arbitrary parameters to be simulated. The key assumption is that the properties of the spatially proximal references are correlated just as the properties of references in a temporal locality are. (We will provide supporting empirical evidence.) In our scheme a trace **is** compressed in two steps. First, a cache filter compresses the trace by eliminating a large proportion of the references in each temporal window, a block filter then compacts the trace by discarding a large fraction of the references in each spatial neighborhood. While our experimental data concerns caches, the techniques can be used for TLB and paging studies also.

The ensuing discussion begins with some definitions and a brief review of trace compaction using a cache **filter.** This is followed by an analysis of how a block filter can be used to compact traces further. This section describes the **rationale behind** our blocking scheme, the implementation of the block **filter** and miss rate estimation. Section 4 presents our results and an analysis of the sensitivity of the miss rate estimates on the block **filter** and cache **filter** parameters.

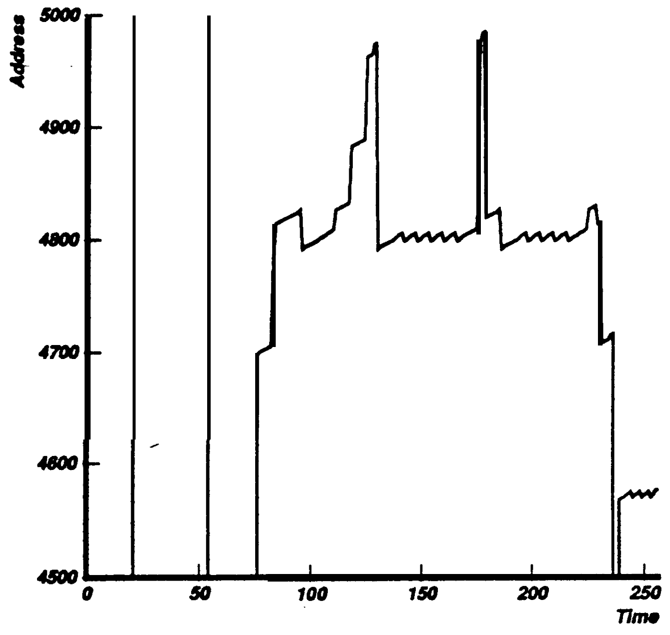

Figure 1: Addressing pattern in a trace segment.

2 Cache Filter

Consider the segment of *a* raw instruction-only address trace shown in Figure 1. The segment represents 250 references, moat in the address range 4500 to 5000. This trace segment displays a large amount of temporal and spatial locality. A diagonal line segment represents a stream of sequential references, called a ***run,*** epitomizing the spatial locality in the program. A run is defined as a maximal stream of sequential references. The saw-toothed segment made up of repeated runs depicts the temporal locality in the program.

Let us first consider a cache **filter.** We define ***compaction*** **ratio** of a cache filter (c_f) to be the number of references in the compacted trace (T_f) divided by the number of references in the original trace *(T),* or $c_f = T_f/T$. A cache filter eliminates redundant references from the trace by recording only those references that miss in the cache. Puzak [4] proves the following theorem to show that a cache **filter** retains complete information for cache studies: the filtered trace generated using a direct-mapped cache with 2' sets preserves the number of misses over all caches with $2^{s'}$ sets, for **all** $s' \geq s$, provided the block size remains constant. The proof uses the property that a reference that hits in **a** cache with 2^s sets is guaranteed to hit in a cache with 2" sets, if $s' \geq s$. In this scheme, if the miss rate of a cache simulated against the compacted trace is m_f, the actual miss rate of the cache is $m = c_f m_f$.

As an illustration, Figure 2 shows the trace segment displayed earlier after filtering with a cache of block size one and 256 sets. Notice the absence of any repetitive patterns. To ensure maximum flexibility, all our filter cache studies will assume both a set size of one and a block size of one. Typical traces yield a compaction ratio between 0.2 and 0.5 after this stage. The example in Figure 2 shows a compression ratio of about 100 : 250 or 0.4.

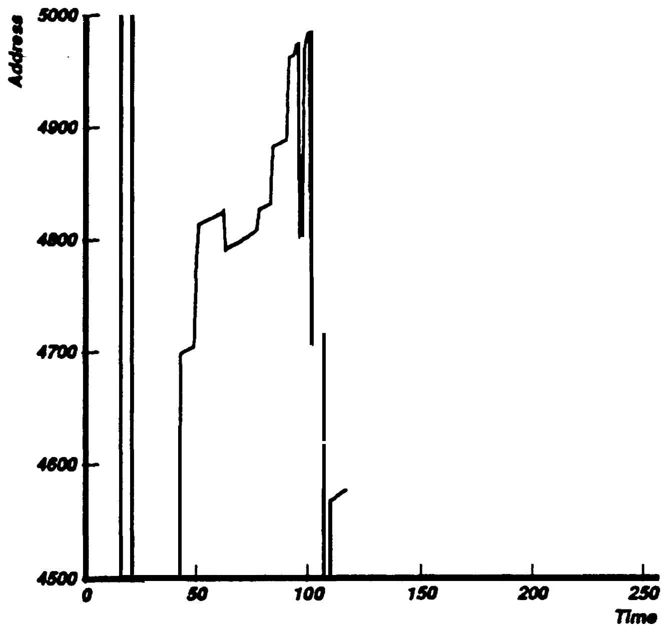

Figure 2: Addressing pattern of a trace segment after cache filtering.

3 Block Filter

Figure 2 shows that the filtered reference stream has a considerable amount of spatial locality which can be exploited to obtain further compaction. Figure 3 shows the same trace, but with only the first reference in any run represented, which reduces the overall compression ratio to 4-O : 250 or 0.16.

The blocking technique uses a representative reference from each run to predict the performance of the entire run by assuming that all the references in the run have similar properties. Blocking, which is also referred to as stratified sampling [5], comes from an established statistical technique [6], and can be explained as follows. Consider a large population of data over which the mean of some parameter is desired. Assume the population can be divided into strata (in our case the strata are runs) within which the value of the parameter of interest is more nearly constant than it is in the population as a whole. Then, blocking draws small unrelated samples from each of the strata separately to estimate the value of the given parameter. Blocking is statistically superior to taking a single sample of the same size from the entire population, or, conversely, for a given accuracy in the estimate of the required parameter value, blocking requires a smaller number of samples, which can result in marked savings in the computation cost.

As an intuitive example, consider a population that has S strata of exactly N items each. Let some parameter of interest be *constant* within each strata i. Then the population mean is equal to the mean of representative values from each strata. However, the mean of a random sample from the population might not yield the exact mean. The former method also requires much less computation than the latter for a given accuracy.

A proof of the success of blocking in predicting the population mean accurately from a sample that is a fraction of the population size is easily derived by computing the variance in

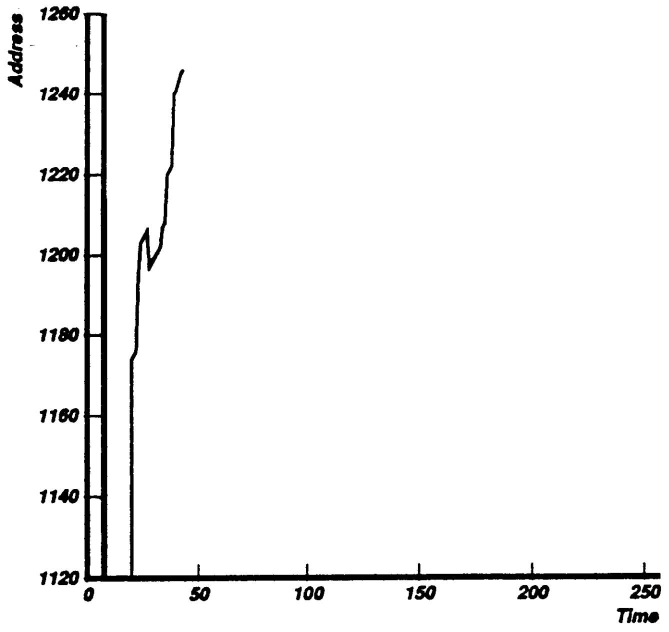

Figure 3: Addressing pattern of a trace segment after cache filtering and block filtering.

the calculated mean for blocking. Consider a population, P (e.g., the set of references in a trace), in which the mean value m of some parameter M (the miss rate) is desired. Let the population have some number of strata (runs of references) numbered onethrough s within each of which the value of M is more or less constant. Let the number of items in the i^{th} strata be N_i (run length), the sum being the total population size N (trace length).

Let us derive the variance in the mean obtained by blocking. Let m_i be the value of M of a representative item in strata i. An estimate of m using the blocking scheme, namely m^*, is obtained by taking the weighted average of the selected m_i's from each strata. There being s strata, the total number of samples is also s. Thus,

$$m^* = [\frac{N_1}{N}m_1 \text{ t } \frac{N_2}{N}m_2 + \ldots + \frac{N_s}{N}m_s]$$

If the variance in the value of M in strata i is σ_i^2, and the items in successive strata are uncorrelated, the variance of m^* is approximately given by

$$Var(m^*) = [\frac{N_1^2}{N^2}\sigma_1^2 + \frac{N_2^2}{N^2}\sigma_2^2 + \ldots + \frac{N_s^2}{N^2}\sigma_s^2]$$

If the variances within strata are the same, say σ_s^2, and assuming for a moment that the strata sizes are the *same,* $Var(m^*)$ is simply σ_s^2/s. In practice σ_s^2 is small, and when further divided by the number of strata σ_s^2 becomes negligible. (We will show supporting experimental data later.)

To further display the benefits of blocking, we contrast this variance with the variance in $\overline{m}$, the estimate of m obtained from the mean of a random sample of size n. If the variance in the value of M for the whole population is σ^2, the variance in the mean of a sample of size n is approximately given by

$$Vat(A) = \sigma^2/n$$

The estimate m^* based on blocking is better than or equal to the estimate $\overline{m}$ from straightforward sampling provided $Var(m^*)$ is smaller than or equal to $Var(\overline{m})$, that is, if $\sigma_s^2 \leq \sigma^2$ for equal sample sizes $(n = s)$. Therefore, the effectiveness of blocking depends on the availability of strata with smaller variances within the strata than in the whole population.

We performed several experiments to evaluate the potential of applying blocking to trace compaction. For our studies concerning address traces, N is the length of the original trace, the parameter of interest M is the miss rate, s is the trace sample length obtained by applying the blocking technique to the original trace, and n is the number of references in a random sample from the original trace. The miss rate computations assume a cache with 16K sets, block size one, and set size one. Three benchmark traces, PS1, AL1, and TMIL1, are used. PS1 is a sample of a DEC program to compare two interconnection net lists in VLSI circuits; AL1 is a microcode address allocator; and TMIL1 is an instruction level simulator for the MIPS processor designed at Stanford University. The first two traces were obtained using an address tracing scheme called ATUM [7], and TMIL1 was obtained by tracing a VAX-11/780 using the T-bit technique. Strata sizes or run lengths in the block filtered trace are limited to a maximum of four for reasons discussed later. For example, a run of length of 11 will be partitioned into three strata of sizes 4, 4, and 3.

Figure 4 displays the correlation between the miss rates of references within a run by plotting the distribution of the difference between the mean miss rate within any run and the miss rate of a representative reference. By miss rate of a reference we mean the proportion of times the reference suffers a miss. Figures 4(a),(b), and (c) show the distribution when the representative reference is the leading reference in the run. Figure 4(d) compares the distribution when the reference is chosen from the first, middle, or last position in a run for benchmark PS1.

It is easy to see that the mean miss rate for the run and the representative miss rate are nearly identical. Minor variations, if at all,' are randomly distributed around zero, and their effect cancels out to first order. The choice of the representative also does not matter because distributions for the three positions of the selected reference are also similar (from Figure 4(d)). Quantitatively, for all benchmarks the intra-run variance in the miss rate is substantially smaller than the variance over the whole trace. For example, in PS1, σ_s^2 is 200 and σ^2 is 4500 (miss rate is in percent). Blocking with a maximum strata size of four yields a compaction of about 0.4 and a miss rate variance less than 0.002. Compare this with the variance of 0.03 in a random sample of the same size. The small variance in the miss rate estimate from the compacted trace displays the effectiveness of blocking.

Our goal is to reduce the compaction ratio s/N while minimizing the variances in the blocking mean. Because the variance within runs, σ_s^2, is very small, theoretically, we expect that s/N can be made very small without significantly sacrificing accuracy by choosing larger strata sizes. For instance, in the PS1 trace, a compression of 0.15 can be achieved if the maximum strata size is relaxed to 16. However, in practice, we cannot arbitrarily reduce the blocking sample size because our assumption of equal strata sizes is violated. By choosing just one reference from each strata, the results get biased towards small sized strata which have a higher miss rate on average. For instance, we have observed that runs of length one, typically composed of data, have a higher miss rate as compared to relatively long runs, typically formed by instructions. One solution is proportional sampling, where the number of samples chosen from each strata is proportional to the strata size [5]. We propose a simple and more practical solution to the strata size inequality problem by limiting the maximum size of any stratum. By this method, a stratum that exceeds the maximum size is split up into smaller strata, thus decreasing the bias against large strata.

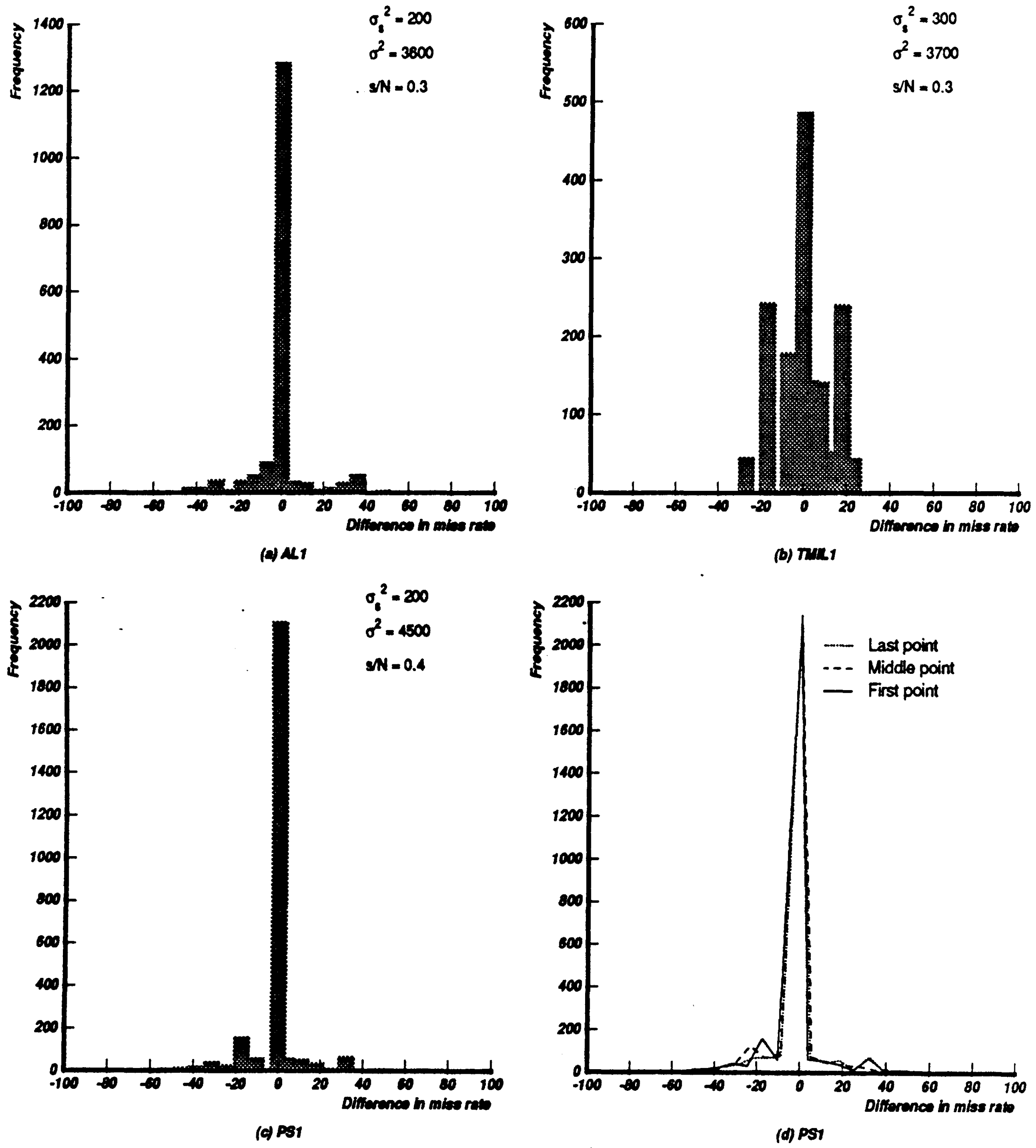

Figure 4: Distribution of the difference between the mean miss rate within a run and the miss rate of a representative reference from that run. σ_s^2 is the intra-run variance of the miss rate (percent), σ^2 is the miss-rate variance in the entire trace, and s/N is the compaction ratio.

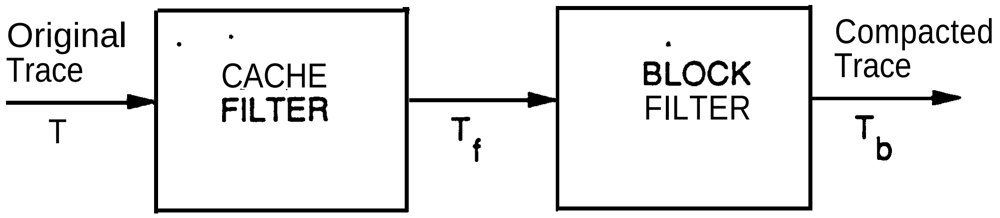

Figure 5: Trace compaction steps.

3.1 Implementation of the Cache and Block Filters

Our trace compaction set-up is depicted in Figure 5. Let us denote a cache with S sets, set-size *D*, block size *B*, and sub-block size B_s, as $C : (S, D, B, B_s)$. The original trace is first passed through a cache filter $C_f : (S_f, 1, 1, 1)$, which emits only the references that miss in the cache. The number of references is reduced from T to T_f, which gives a cache filter compaction ratio of $c_f = T_f/T$.

The trace of size T_f is subsequently passed through a *block filter* characterized by two parameters called the window size (w) and filter block size *(b)*. The block filter scoops up a window of w references and sends out a reference from each spatial locality contained within the window. References are said to belong to the same spatial locality if they have the same *address div b.* (Runs that straddle block boundaries of size *b* are split up into multiple runs of maximum size b.) The above procedure is repeated until no more references remain. Because the emanated references represent the given locality, the low order $log_2 b$ bits are dropped from the output reference. We limit the search for runs within time intervals of w references because spatially local references that are temporally unrelated are unlikely to have similar properties. Furthermore, looking for sets of spatially proximal references within a given time window is also necessary to overcome the practical difficulty of isolating possibly interleaved runs in a stream of references. The block filter reduces the trace length to T_b for a blocking compression ratio $c_b = T_b/T_f$, and an overall compaction of $c_{fb} = c_f c_b$.

For example, consider a cache filtered trace with the addresses (1, 199, 2, 198, 4, 196, 6, 194, 7, 3000, 8, 9, 10) that are input to a block filter with parameters $w = 10$ and $b = 4$. The block filter outputs the references (0, 49, 1, 48, 750) from the first window, and outputs (2) from the second window, yielding $c_b = 6/13$.

Before the compacted trace is used for simulations, we need to go through one more step. Because the block filter transforms the trace by dropping the low order $log_2 b$ bits, the cache entries that were differentiated using these bits must now be coalesced into one representative entry. To achieve this, the cache under consideration C must be transformed in the following manner. Let the cache under consideration be C : (S, D, B, B_s), with the only constraint $S \geq S_f$. The transform on the cache can be derived by observing how the low order bits are used in a normal cache access. Bits $< 0 : log_2 B >$ choose a word within the block, and bits $< log_2 B : log_2(S + B) >$ index into some cache set; of these, the operations corresponding to the low order $log_2 b$ bits must be dropped. Therefore, the original cache is transformed to the cache denoted as $C^* : (S^*, D^*, B^*, B_s^*)$, or

$$C^* : \left(\frac{S}{\lceil \frac{b}{B} \rceil}, D, \lceil \frac{B}{b} \rceil, \lceil \frac{B_s}{b} \rceil\right)$$

For example, a cache C : $(16K, 1, 4, 4)$ is transformed to the cache C^* : $(16K, 1, 1, 1)$ for $b = 4$, and to the cache C^* : $(4K, 1, 1, 1)$ for $b = 16$. As can be seen from the latter case, the above transformation often reduces the amount of memory required to simulate the cache.

3.2 Miss Rate Estimation

We now describe how the miss rate of the trace sample can be derived from the miss rate of the cache filtered and block filtered trace. Let the miss rate of the cache C^* simulated against the block-filtered trace be m_b, and as before, let the cache filter compaction ratio be cf. Because the references in each spatial locality are expected to behave similarly, the miss rate can be calculated using only one representative from each locality in the compacted trace. Thus, assuming that only one reference is fetched on a miss $(B_s = 1)$, the estimated miss rate is simply

$$m^* = c_f m_b$$

For larger sub-blocks we must include the effect of prefetching, and the formula for the miss rate becomes:

$$m^* = \begin{cases} c_f c_{B_s} m_b & \text{if } B_s^* = 1 \\ c_f c_b m_b & \text{if } B_s^* > 1 \end{cases}$$

The factors c_b and c_{B_s} are the block filter compactions with parameter b and B_s respectively. To explain why the miss rate is of the given form, consider the following two cases.

1. When B_s^* is one $(B_s \leq b)$, the effect of fetching B_s references on a miss must be included and c_{B_s} represents the fractional miss rate decrease.

2. When B_s^* is greater than one $(B_s > b)$, the sub-block size B_s is composed of two factors: B_s^* and b. The prefetch benefits due to B_s^* are already included in the miss rate m_b, and the remaining fraction of prefetch benefits due to b is represented by the factor c_b.

A noteworthy feature of our scheme is that explicitly splitting the filter cache and block filter compressions allows estimating the miss rates of caches with $B < b$, possibly with sub-block placement, which was not possible in earlier schemes. The drawback is that caches with $B_s < b$ require the computation of an additional compaction ratio c_{B_s}. However, the negative effect of this extra computation is mitigated because the trace that is used to derive the compaction ratio is itself compressed by a factor of c_f, which makes the extra computation much smaller than the computation needed to generate the cache filtered trace.

Cache C	c_f	c_b	c_{fb}	m_b	Est. m^*	Act. m	Error%
(16K,1,1,1)	0.42	0.45	0.190	12.17	5.10	5.00	1.94
(1K,1,16,16)	0.42	0.45	0.190	6.65	1.26	1.35	-6.00
($\frac{1}{2}$K,1,32,32)	0.42	0.45	0.190	6.02	1.15	1.24	-7.91
(4K,1,4,4)	0.42	0.45	0.190	12.17	2.31	2.37	-2.40
(4K,2,4,4)	0.42	0.45	0.190	3.95	0.75	0.75	-0.24
(2K,4,4,4)	0.42	0.45	0.190	3.65	0.69	0.70	-0.12

Table 1: Compaction statistics for trace PS1. The filter cache is $C_f : (256, 1, 1, 1)$ and the block filter has parameters $w = 128$ and $b = 4$.

Cache C	c_f	c_b	c_{fb}	m_b	Est. m^*	Act. m	Error%
(16K,1,1,1)	0.26	0.25	0.066	17.33	4.57	5.00	-8.64
(1K,1,16,16)	0.26	0.25	0.066	17.33	1.14	1.35	-14.90
($\frac{1}{2}$K,1,32,32)	0.26	0.25	0.066	15.95	1.05	1.24	-15.30
(4K,1,4,4)	0.26	0.25	0.066	17.33	2.11	2.37	-11.03
(4K,2,4,4)	0.26	0.25	0.066	6.25	0.76	0.75	1.14
(2K,4,4,4)	0.26	0.25	0.066	4.81	0.58	0.70	-15.67

Table 2: Compaction statistics for trace PS1. The filter cache is $C_f : (1K, 1, 1, 1)$ and the block filter has parameters $w = 128$ and $b = 16$.

4 Results

Tables 1 and 2 show the results of compacting the trace PS1 for two different levels of compaction and six caches with widely different parameters. For a compaction ratio of 0.19, the estimated miss rates are close to actual miss rates for all the cache organizations. For the higher compaction of 0.066 the error is predictably larger. Interestingly, while the cache filter and block filter contribute about equally towards overall trace length compression for a low compaction level, the compaction attributable to the block filter is over twice the compaction due to the filter cache for a high compaction level.

Figure 6 summarizes the average performance of our compaction algorithm for the three benchmarks, PS1, AL1 and TMIL1. The miss rates of the three benchmarks are averaged to obtain the depicted miss rates. It is easy to see that reliable cache performance estimates for a variety of cache organizations can be obtained with a compaction of about an order of magnitude in trace length; the results are less accurate in absolute terms when the compaction is close to two orders of magnitude. Nevertheless, the relative performance of the various caches is still reliably predicted.

4.1 Sensitivity Analysis

An interesting question that one would like to answer is: How sensitive are the cache performance estimates to the choice of blocking parameters? Answering these questions is important because for trace compaction to be useful the estimated cache performance results must be robust with respect to filter parameters. A sensitivity analysis identifies such robust regions in the parameter space. We investigated the robustness of the block filter parameters for the three traces. Because the results for the three traces were very similar, we present only the dependence of the average miss rates for brevity.

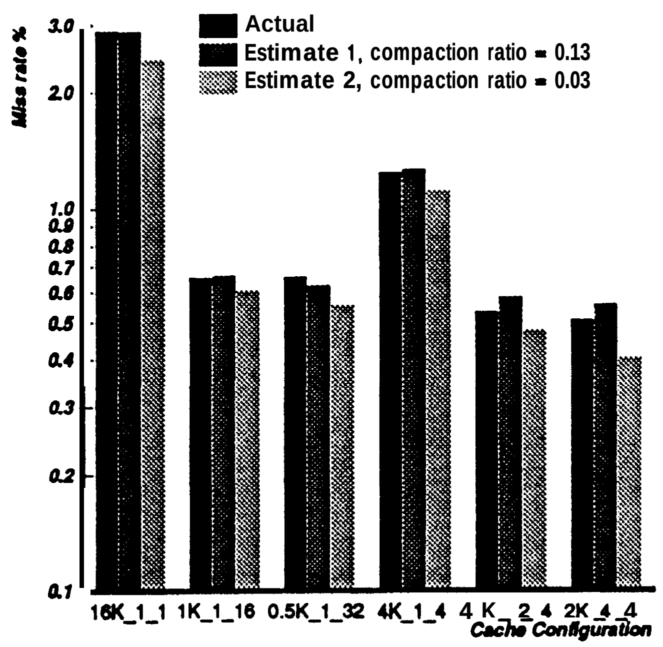

Figure 6: Comparison of actual and estimated miss rates for various compactions. The results for benchmarks **PS1, AL1** and **TMIL1** are averaged. Estimate 1 uses a cache filter $C_f : (256, 1, 1, 1)$, and the **block filter** parameters are $w = 128$ and $b = 4$; estimate 2 uses $C_f : (1K, 1, 1, 1)$, $w = 128$ and $b = 16$.

Figures 7(a) and (b) show the miss rate sensitivity of the cache of interest C : $(16K, 1, 1, 1)$ on **the** window size w and blocking parameter b for filter cache sizes of 256 and 1K sets respectively. **The** first observation is that the window size that yields the best estimate increases with blocking parameter b. The reason is that for a larger b one needs to look at a greater number of references to collect runs of size close to b. As an extreme example, it is ridiculous to look for runs of length **16** in a time window w of length 2. For this reason, performance for values of $w < 64$ diverges from the actual; the error becomes worse for larger b. Similarly, the time window cannot arbitrarily increase in size because of the increased probability of putting unrelated references in the same strata. Thus we see that the error becomes large for $w > 256$. The problem arises when the miss rates of entries within a strata are weakly correlated. If the window size is very large, neighboring references can be found with a high probability even if they do not occur close together in time. Including these references with uncorrelated miss rates in the strata results in large errors in estimated miss rates.

Figures 8(a) and (b) depict the miss rate sensitivity of the cache C : $(1K, 1, 16, 16)$. While the miss rates in Figure 8(a) behave as expected, the dip in the miss rate curves in Figure 8(b) (filter cache $C_f : (1K, 1, 1, 1)$) for $w > 256$ needs explanation. This dip occurs because the large window size can capture a wide spatial locality that includes multiple **runs** whose references have uncorrelated miss rates. A similar behavior is not seen in Figure 8(a) (**filter** cache C_f : $(256, 1, 1, 1)$) **because** the smaller **filter** cache **causes** the cache-filtered trace to be less dense, which reduces the size of a locality that can be captured in a window.

Figure 9(a) depicts the tradeoff between accuracy of prediction and compaction achievable for a cache C : $(1K, 1, 16, 16)$ (assuming the filter cache is $C_f : (1K, 1, 1, 1)$). **PS1** is the benchmark

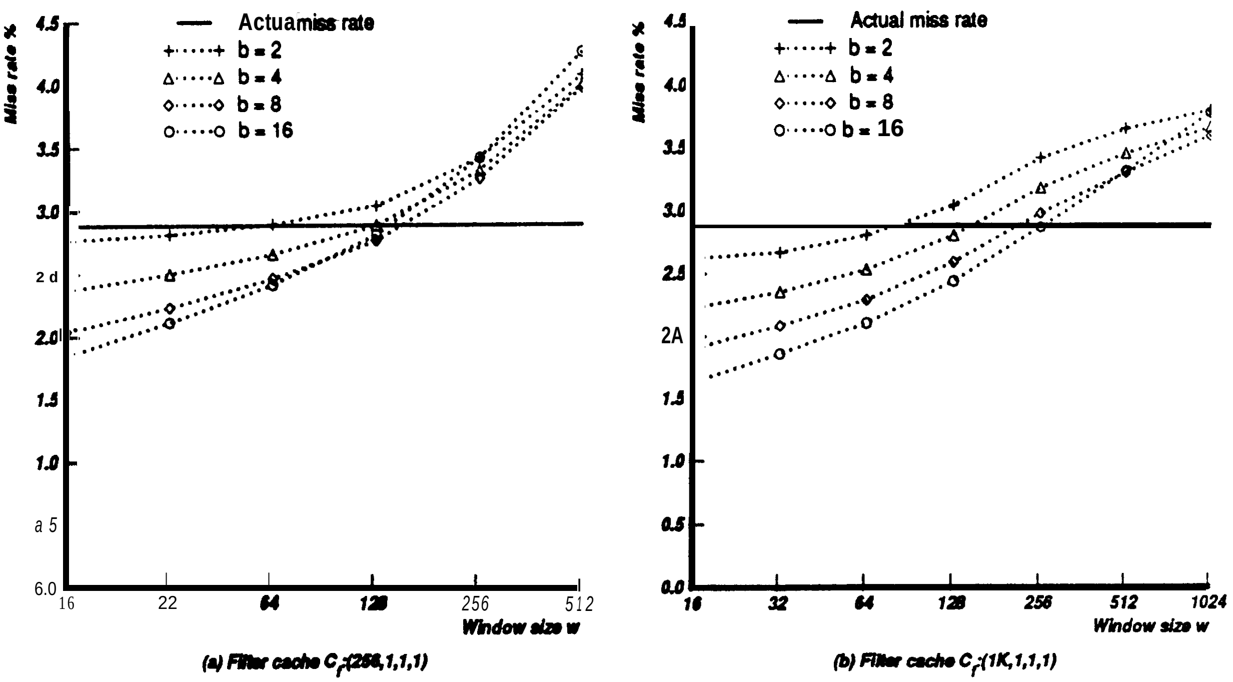

Figure 7: Sensitivity of the miss rate of a cache C : $(16K, 1, 1, 1)$ on the block filter parameters w and b. The dotted curves show the estimated miss rates and the solid line represents the actual miss rate.

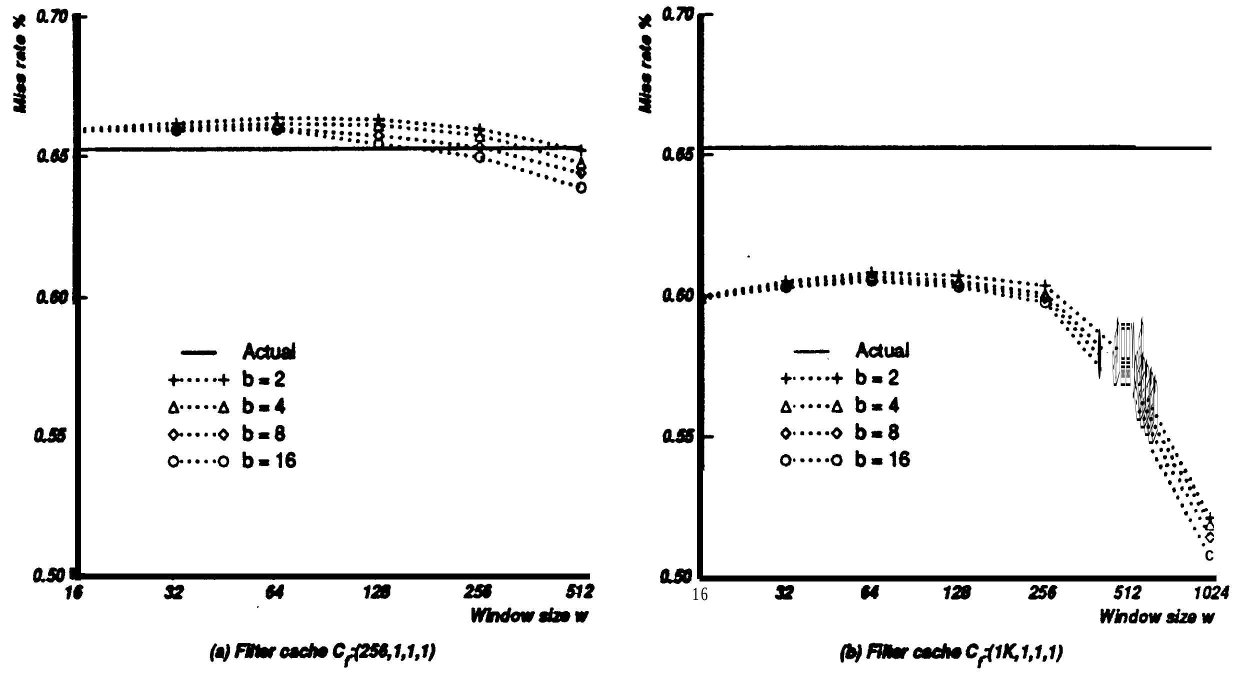

Figure 8: Sensitivity of the miss rate of a cache C : $(1K, 1, 16, 16)$ on the block filter parameters w and b.

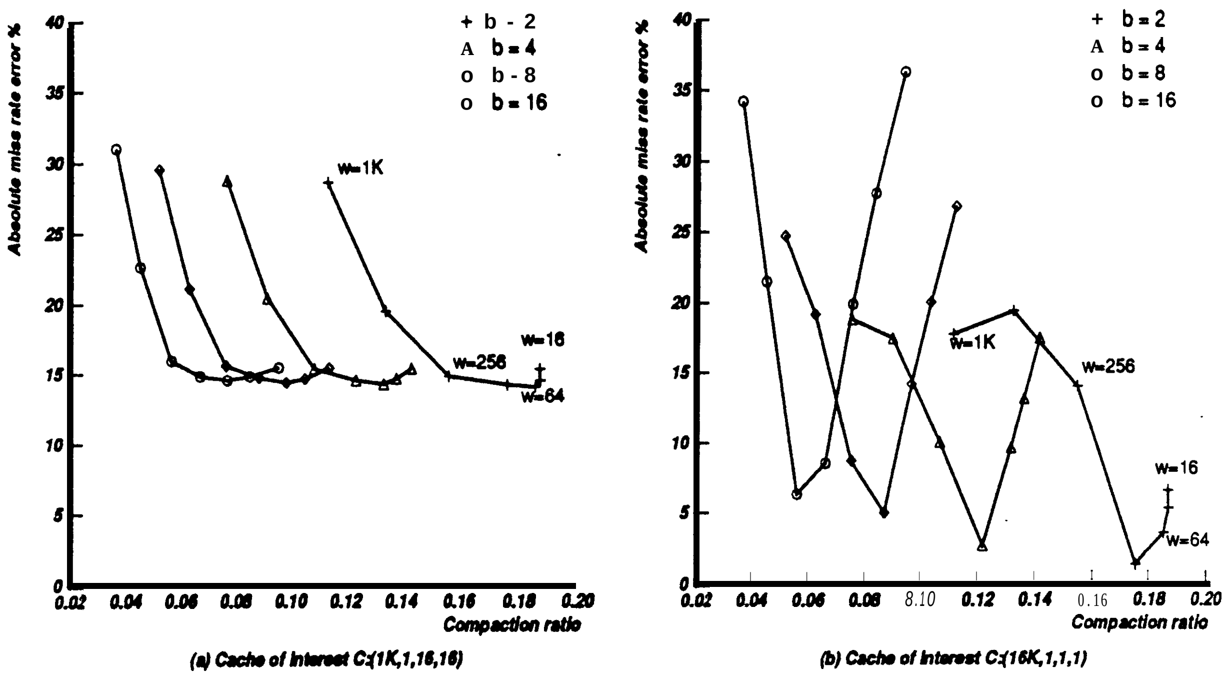

Figure 9: Compaction versus accuracy for PS1. A filter cache $C_f : (1K, 1, 1, 1)$ is used. w and b represent the block filter parameters.

used. Four curves showing the miss rate error versus compaction for four different values of b are plotted with w varying from 16 to 1024. In all cases the error in estimation is less than 15% if $w \leq 256$ with the best prediction for window sizes in the range 64 to 256. A 16K-set cache with unit set size and block size one displays a similar trend as shown in Figure 9(b). The estimates are best when $64 \leq w \leq 256$.

An important observation is that there is no reason to choose a small block filter parameter b. As long as w satisfies the required criterion, large values of b can be chosen giving high compaction with little variation in the results.

5 Conclusions

This paper introduced a new technique called cache filtering with blocking to compact traces by one to two orders of magnitude while allowing simulation of a wide range of cache organizations. The compaction algorithm can also be used to compact traces in TLB and paging studies. For cache studies, our experiments show that a block filter window size w in the range 64 to 256 gives the best tradeoff between accuracy and compaction, and a large blocking parameter b can yield significant compaction while introducing few distortions in the compacted trace. As an indication of the power of the compaction technique we can routinely compact traces by nearly two orders of magnitude while introducing less than 10% – 15% error in the miss rate estimate.

References

[1] Anant Agarwal. ***Analysis of Cache Performance for Operating Systems and multiprogramming.*** PhD thesis, Stanford University, Computer Systems Laboratory, February 1987.

[2] P. J. Denning. The Working Set Model for Program Behavior. ***Communications of the ACM,*** 11(5):323–333, May 1968.

[3] Alan Jay Smith. Two Methods for the Efficient Analysis of Memory Address Trace Data. ***IEEE Transactions on Software Engineering,*** SE-3(l), January 1977.

[4] Thomas R. Puzak. ***Analysis of Cache Replacement Algorithms.*** PhD thesis, University of Massachusetts, Department of Electrical and Computer Engineering, February 1985.

[5] J. L. Hodges Jr. and E. L. Lehmann. ***Basic Concepts of Probability and Statistics.*** Holden-day, Inc., San Francisco, 1964.

[6] Rupert *G.* Miller Jr. ***Beyond Anova - Basics of Applied Statistics*** . John Wiley and Sons, Inc. New York, 1986.

[7] Anant Agarwal, Richard L. Sites, and Mark Horowitz. ATUM: A New Technique for Capturing Address Traces Using Microcode. In ***Proceedings of the 13th Annual Symposium on Computer Architecture,*** pages 119-127, June 1986.