Combining Trace Sampling with Single Pass Methods for Efficient Cache Simulation

Thomas M. Conte *Member, IEEE*, Mary Ann Hirsch *Student Member, IEEE*, and Wen-Mei W. Hwu *Member, IEEE*

Manuscript received _____.

Affiliation: Thomas M. Conte and Mary Ann Hirsch are with the Department of Electrical and Computer Engineering, P. O. Box 7911, North Carolina State University, Raleigh, NC 27695–7911. E-mail: conte@eos.ncsu.edu . Wen-Mei W. Hwu is with the Center for Reliable and High Performance Computing, University of Illinois, Urbana, IL 61801.

Key-Phrases

PERFORMANCE ANALYSIS

SAMPLING TECHNIQUES

SINGLE PASS ALGORITHMS

STACKING ALGORITHMS

TRACE-DRIVEN SIMULATION

Abstract

The design of the memory hierarchy is crucial to the performance of high performance computer systems. The incorporation of multiple levels of caches into the memory hierarchy is known to increase the performance of high end machines but the development of architectural prototypes of various memory hierarchy designs is costly and time consuming. In this paper, we will describe a single pass method used in combination with trace sampling techniques to produce a fast and accurate approach for simulating multiple sizes of caches simultaneously.

I. Introduction

The development of computer architectures through physical prototyping is no longer a viable practice. Today, simulation is the most popular method to evaluate computer system performance in the development stage. Trace-driven simulation techniques have been introduced into the development process to reduce the initial investment of time and financial resources in proposed system enhancements. Its use allows the researcher to study the effects of the proposed enhancement in a timely, detailed manner without the investment in hardware prototypes. Trace-driven simulation is a powerful tool that can be used to explore the system at many different levels, such as the TLB, caches, and main memory. Yet this technique is extremely slow and sometimes inefficient.

Trace-driven simulation and the development of tools for trace-driven simulation are very active research areas due to their inadequate performance rates. In response to this problem, many different computer architecture groups have developed approaches, such as analytic models and novel simulation techniques, for analyzing data in an efficient manner [1],[2],[3]. Analytic cache simulation models can be constructed rather rapidly, but they produce results of dubious accuracy. Therefore, analytic models are inappropriate for prototyping memory systems. The analytic models, however, can be used for other system components, such as processors, where accurate, reproducible results are not the primary focus of the model.

The most common technique of cache simulation, *functional cache simulation*, simulates the cache at the register-transfer level. Functional cache simulation of a large number of cache designs is quite expensive and requires a large number of simulation runs. Early work by Mattson, *et al.* exploited the properties of stacking replacement algorithms to

devise a single-pass cache simulation algorithm [1]. The best known member of this class of replacement algorithms is the *least-recently used* algorithm [1]. This work was then extended to other cache organizations by Traiger and Slutz [4], and extended even further for additional cache management policies by Thomson and Smith [5], and Hill and Smith [6]. At the same time, work has been done on improving the performance of functional cache simulation for multi-megabyte caches [7]. Such techniques are based on statistical sampling of caches, first proposed by Laha, *et al.* [8] and Stone [9]. These techniques take 30–40 contiguous stripes or *clusters* of address references from the trace to produce an input to a simulation. The simulation results based on the sampled data are only approximations of the simulation results for the entire trace yet Kessler, *et al.* found that these results are highly accurate [7]. Their accuracy, however, depends on the methods used for repairing the state of the cache at the beginning of each cluster before the simulation is applied. Variations in these repair mechanisms produce different simulation results which is known as the *state repair problem*.

In this paper, we will describe a fast and accurate cache simulation technique that only requires a single data pass while simultaneously exploring multiple sizes of cache. The solution employs trace sampling to solve the state repair problem by exploiting the properties of inclusion that make single-pass methods possible. The single-pass method is described in Section 2. Section 3 provides a discussion of sampling and a description of our techniques. In addition, it includes a statistical evaluation and validation of our techniques using simulation results for the SPECint92 benchmark suite. Lastly, there is a discussion of the impact of these techniques on simulation performance and directions for future research.

II. Single-Pass Simulation

The most commonly used metric in memory system studies is the miss ratio. The miss ratio is the ratio of the number of references that are not satisfied (i.e., a *miss*) by a cache at a level of the memory system hierarchy to the total number of references made at that level. The miss ratio is characteristic of the workload (e.g., the memory trace) yet it is also independent of the memory access time of the requested elements. A given miss ratio can be used to decide whether a potential memory element design will meet the required

access time for the memory system [10]. The recurrence/conflict model proposed by Hwu and Conte [11] calculates the miss ratios for a range of cache configurations on a single pass. In the ideal case of an infinite cache, the miss ratio may be expressed as,

$$\rho = \frac{N - R}{N}, \tag{1}$$

where R is the total number of recurrences and N is the total number of references. *Conflicts* cause non-ideal behavior. A *dimensional conflict* is defined as an event which converts a recurrence into a miss that is due to limited cache capacity or mapping inflexibility. The following formula can be used for deriving cache miss ratio, ρ, for a given trace and cache dimension:

$$\rho = \frac{N - (R - D)}{N}, \tag{2}$$

where D is the total number of dimensional conflicts. (For the example, $\rho = (8-(4-2))/8 = 0.75$.) This is a general recurrence/conflict model that can be extended to account for other effects and in this paper, we adapt this model for statistical sampling of the trace.

A. Reference streams and cache dimensions

A formal abstraction of a benchmark's trace is termed a *reference stream.* This is a sequence of references to addresses, $w(k)$, of length N $(0 \leq k < N)$. When required, the addresses are represented by lower-case Greek letters, such as α, β, γ. Note that a reference at $w(k)$ occurs later than $w(k-1)$ in time, but the parameter k does not take into account the difference in service times between cache hits and cache misses, so it does not represent parameterized time. For this reason, k is referred to as the *reference count.* The dimension of a cache is expressed using the notation, (C, B, S), for a cache of size 2^C bytes, with a block size of 2^B bytes, and 2^S blocks contained in each associativity set. The term *set size* is defined as the associativity level, or the number of blocks per set. *Cache size* is the total number of bytes per cache and *block size* has been called *line size* elsewhere [12]. It should be noted that this notation requires $C \geq B + S$. The notation (C, B, ∞) is an abbreviation for the dimension of a fully-associative cache $(S = C - B)$. All caches are assumed to use Least Recently Used (LRU) replacement and map addresses into sets using bit selection [6].

It is useful to partition the reference stream by setting the block offset portion of all addresses in the stream to zero. This produces a *block reference stream*, $w_B(k)$, is defined such that,

$$w_B(k) = 2^B \left\lfloor \frac{w(k)}{2^B} \right\rfloor .$$

In binary, this is equivalent to setting the least-significant B bits to zero.

B. Least recently used (LRU) stack operation

LRU stacks were first introduced by Mattson, *et al.* in [1] as a way to model the behavior of paging systems. An LRU stack operates as follows: when an address, $w_B(k) = \alpha$, is encountered in the block reference stream, the LRU stack is checked to see if α is present on the stack. If α is not present, it is pushed onto the stack. However, if α is present (e.g, it is a recurring reference), it is removed from the stack, then repushed onto the stack. This stack maintenance policy is specific to a particular block size, as is the discussion below.

A stack is represented as $S_B(k)$, maintained for a block size B at time k. The ith ordered item of $S_B(k)$ is expressed as, $S_B(k)[i]$. The stack may also be expressed as an ordered list, such that $S_B(k) = \{S_B(k)[0], S_B(k)[1], \ldots, S_B(k)[m]\}$, where m is the depth of the stack. The following operations are defined for a stack:

the $\mathbf{push}(\cdot)$ function,

$$\mathbf{push}(S_B(k), \alpha) = \Big\{ \alpha,\ S_B(k)[0],\ S_B(k)[1],\ \ldots,\ S_B(k)[m] \Big\},$$

the $\mathbf{\Delta}(\cdot)$ function,

$$\Delta(S_B(k), \alpha) = i, \qquad \text{if } S_B(k)[i] = \alpha,$$

and, the $\mathbf{repush}(\cdot)$ function,

$$\begin{aligned} \mathbf{repush}(S_B(k), \alpha) \ = \ & \Big\{ \alpha,\ S_B(k)[0],\ S_B(k)[1],\ \ldots,\ S_B(k)[\Delta(S_B(k), \alpha) - 1], \\ & S_B(k)[\Delta(S_B(k), \alpha) + 1],\ \ldots,\ S_B(k)[m] \Big\}. \end{aligned}$$

$\Delta(S_B(k), \alpha)$ and $\mathbf{repush}(S_B(k), \alpha)$ are undefined when $\alpha \notin S_B(k)$. When $S_B(k)$ and α are understood, it is convenient to use $\mathbf{\Delta} = \mathbf{\Delta}(S_B(k), \alpha)$. Note that $\mathbf{push}(\cdot)$ and $\mathbf{repush}(\cdot)$ are

defined as side-effect-free functions rather than procedures. This is to remove dependence on the reference count variable, k.

For an address $\alpha = w_B(k)$, the least recently used (LRU) management policy for a stack is shown in Figure 1. In Step 1.1, the references between the top of stack and the recurring reference have been referred to as the set $\mathbf{\Gamma} = \{\beta_i \mid \beta_i = S_B(k-1)[i], 0 \leq i \leq \mathbf{\Delta}\}$. Figure 1

1. **if** $\alpha \in S_B(k-1)$ **then**
1.1 *determine* D *from* $\mathbf{\Gamma}$
1.2 $S_B(k) \leftarrow \mathbf{repush}(S_B(k-1), \alpha)$,
2. **else** $S_B(k) \leftarrow \mathbf{push}(S_B(k-1), \alpha)$
3. $N \leftarrow N + 1$

Fig. 1. The least recently used management policy for a stack, $S_B(k)$ (adapted from Mattson *et al.*).

is applied to $\alpha = w_B(k)$ for all k. The LRU policy is essentially a definition for calculating $S_B(k)$ from $S_B(k-1)$ and α. In most situations, $S_B(k)$ is calculated in order to obtain other statistics, such as the stack depth distribution.

III. Statistical Sampling of Address Traces

Trace-driven cache simulation produces an enormous amount of data so it is important to develop methods that accurately and efficiently summarize the performance results into a small set of statistics. The most common performance statistic for cache simulations is the miss ratio. The miss ratio is an arithmetic average over time that can be accurately predicted by statistically sampling the trace [8], [7]. Previously, statistical sampling has been applied to functional cache simulation. Our extension applies statistical sampling to the single pass techniques.

A. Definition of sampling

Consider a reference stream, $w_B(k)$. Statistical sampling takes N_S clusters of length L_S from this trace. Note that each sample is a *contiguous* block of L_S references. Typical

values for N_S are 40 samples and typical values for L_S are in the range of 100,000 references [8],[7]. The removed references between two samples comprise a *sample gap*. In order to reduce sampling bias, the gap size is a uniformly distributed random number with a mean of $\hat{L}_G$. Without loss of generality, it is assumed that L_G is a constant and that the samples are applied to the cache simulator in the order they are taken from the trace. Yet, the state of the cache is unknown between each sample, so the cache state must be repaired between the application of each sample.

In the following discussion, the actual miss ratio is denoted ρ and the miss ratio estimated by sampling is denoted ρ'. This convention is used for the components of the miss ratio, $R[B]$, $D[C,B,S]$ and N. The sampled miss ratio is, therefore,

$$\rho' = \frac{N' - (R'[B] - D'[C,B,S])}{N'}. \tag{3}$$

There are several ways of measuring the error between ρ and ρ'. The percentage change, or *relative error*, can be calculated as $\mathrm{RE}(\rho) = |\rho - \rho'|/\rho$. The use of relative error in conjunction with miss ratios may inflate differences between ρ and ρ'. These differences, however, would not normally matter to a designer. Designers typically pick a target miss ratio value and accept any design that satisfies that value [13]. In particular, a cache with a miss ratio of $\rho = 0.01\%$ is as acceptable for a design as a cache with a sampled miss ratio of $\rho' = 0.005\%$, since both values are very small. However, the relative error between these values is $\mathrm{RE} = 50\%$. Therefore relative error is appropriate for large values of the miss ratio and is less significant for smaller values. This suggests that an error measure, such as the *absolute miss ratio error*, $\mathrm{AE}(\rho) = |\rho - \rho'|$, that is weighted by the actual miss ratio would better characterize the impact of the difference.

B. State repair techniques

The accuracy of functional cache simulation techniques that employ statistical sampling of address traces depends on the method used for repairing that state of the cache at the beginning of each sample. This is better known as the *state repair problem*. Several approaches have been proposed for functional cache simulation [8],[14],[9],[7]. This section presents two approaches to this problem. The first, the *fill-flush* technique, is an adaptation of Stone's approach [9] to the single-pass technique. The second method, the *no-state-*

loss technique, applies a new approach to the single-pass technique. The two single-pass approaches to state repair are compared using empirical simulation results based on the SPECint92 benchmark suite.

The *fill-flush* approaches to state repair entail removal of all unknown unique references from the trace [9]. To apply this approach to single-pass simulation, the recurrence/conflict single-pass method is extended to measure the number of references whose states are unknown due to the lost state between samples. It is traditional to term these references, *fill references* [9]. Let $F[B]$ be the count of fill references for block size B, L_S be the cluster size, and L_G be the gap size. In the adapted algorithm, the cache is flushed between each cluster and $F[B]$ is collected. References with unknown state are removed from calculation of the miss ratio by using Equation (4):

$$\rho = \frac{N - (R[B] - D[C, B, S])}{N - F[B]}. \tag{4}$$

Figure 2 shows the recurrence/conflict single-pass algorithm modified for this form of sampling. The flushing of the LRU stack is represented as "$S_B(0) \leftarrow \emptyset$" (Step 1.1 of Figure 2). Step 1.2.1 shows the sequential removal of the sampled references from the reference stream, w_B. Step 1.2.1 is not required if the samples could be written to disk or consumed in-process because the reference stream, w_B is not stored.

Typical values for AE are presented in Figures 3–4 for direct-mapped and fully associative caches with cluster sizes of $L_S = 100,000$ references. From these results, it can be seen that AE decreases as the cache size increases, however, large cache sizes have a small constant error due to the fact that the fill references can potentially contain first-time references to locations. When C is increased to the point where $D[C, B, S] = 0$, then,

$$\rho(C \text{ sufficiently large}) = (N - R[B])/N. \tag{5}$$

However,

$$R[B] = R'[B] + f_r \times F[B], \tag{6}$$

where f_r is the fraction of fill references that recur. The fraction f_r can only be known by measuring the entire trace since the status of fill references is unknown due to state loss. If all fill references are assumed to be recurrences, this effectively sets $f_r = 1$, which

1. **for** $i \leftarrow 0$ **to** $N_S - 1$
1.1 $S_B(0) \leftarrow \emptyset$
1.2 **for** $j \leftarrow 0$ **to** L_S
1.2.1 $\alpha \leftarrow w_B(i \times (L_S + L_G) + j)$
1.2.2 **if** $\alpha \in S_B(j-1)$ **then**
1.2.2.1 `do_recurrence`$(\alpha, \mathbf{\Gamma})$
1.2.2.2 $S_B(j) \leftarrow \mathbf{repush}(S_B(j-1), \alpha)$,
1.2.3 **else**
1.2.3.1 $S_B(j) \leftarrow \mathbf{push}(S_B(j-1), \alpha)$
1.2.3.2 $F[B] \leftarrow F[B] + 1$
1.2.4 $N \leftarrow N + 1$

Fig. 2. Extension of the recurrence/conflict single-pass algorithm to sampling using fill-flush state repair.

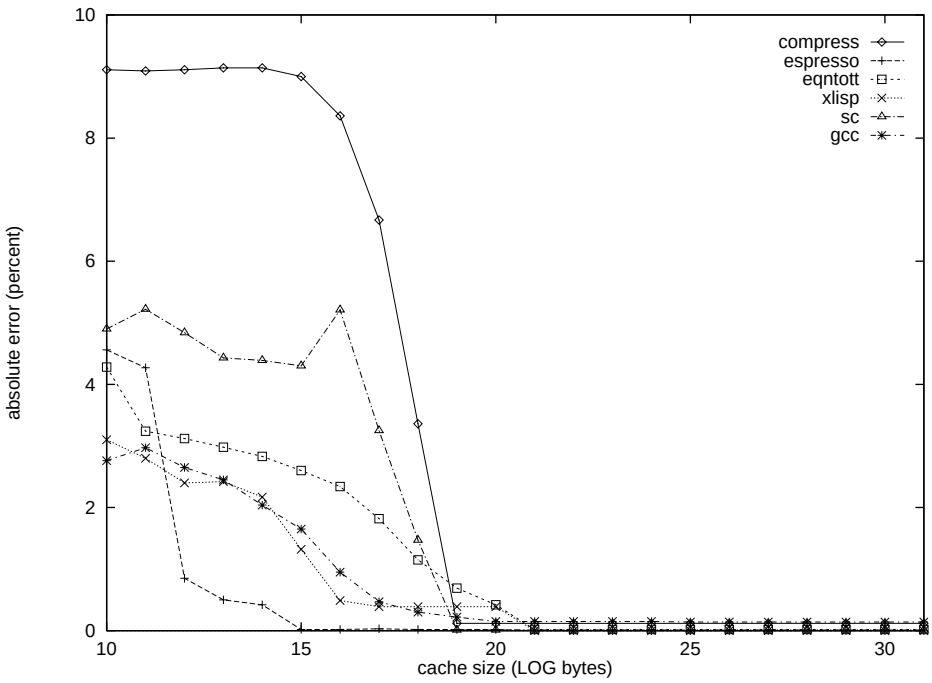

Fig. 3. Absolute error for fill-flush, direct-mapped caches, $L_S = 100,000$.

results in the constant error observed in the figures. The ratio of sample simulation time for the full-flush approach to the non-sampled approach (i.e., using the full trace) for the SPECint92 benchmark suite is presented in Figure 5. This ratio can be thought of as the *speedup* of the sampling approach. As expected, small sample sizes usually obtain a higher speedup over large sample sizes in the figure since sampled traces with smaller sample sizes are smaller and produce less work for the simulator.

There are several other approaches to state repair that can collectively be termed *cache warm-up* approaches. In these approaches, calculations of performance metrics are delayed for each sample until the cache contents are stable, or "warmed up." This warm-up state is specified by some criterion, where the criterion may vary based on the approach. One possible warm-up criterion is to use a fraction of the sample to prime the cache and then record recurrences and conflicts for the second half of the sample. This approach might work well for small caches using functional simulation. It does not, however, perform well for single-pass methods or for larger caches using functional simulation [14] [7]. Several other possible warm-up criteria are

1. Delay calculation of performance metrics for each set in the cache until each set is filled with references [14],
2. Redefine the miss ratio in terms of the lifetime of references and measure expected lifetime length [7].

Method 1 is not applicable to single-pass cache simulation because the state of every associativity set in every possible cache dimension is not kept. If the state was kept, the maintenance of this information would be equivalent to the maintenance of state information for functional simulation, and the algorithm would reduce to functional simulation. Method 2 also requires maintenance of additional state information. The lifetime of a reference is defined as the period of time a reference is needed in the LRU stack. In this case, method 2 maintains the lifetime of each reference in the LRU stack. Lifetime is specific to a particular cache configuration and this method has similar problems to the first when large data structures are considered. Instead of using this second method to extend single-pass methods to achieve lower AE than the fill-flush techniques, an approach that avoids the problem of state repair entirely is possible.

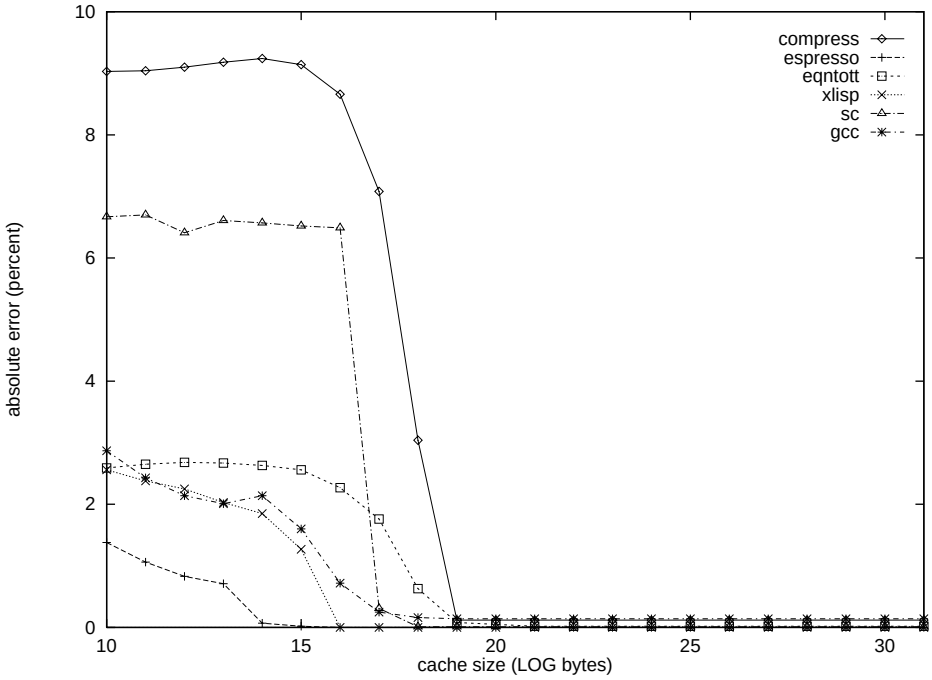

Fig. 4. Absolute error for fill-flush, fully associative caches, $L_S = 100,000$.

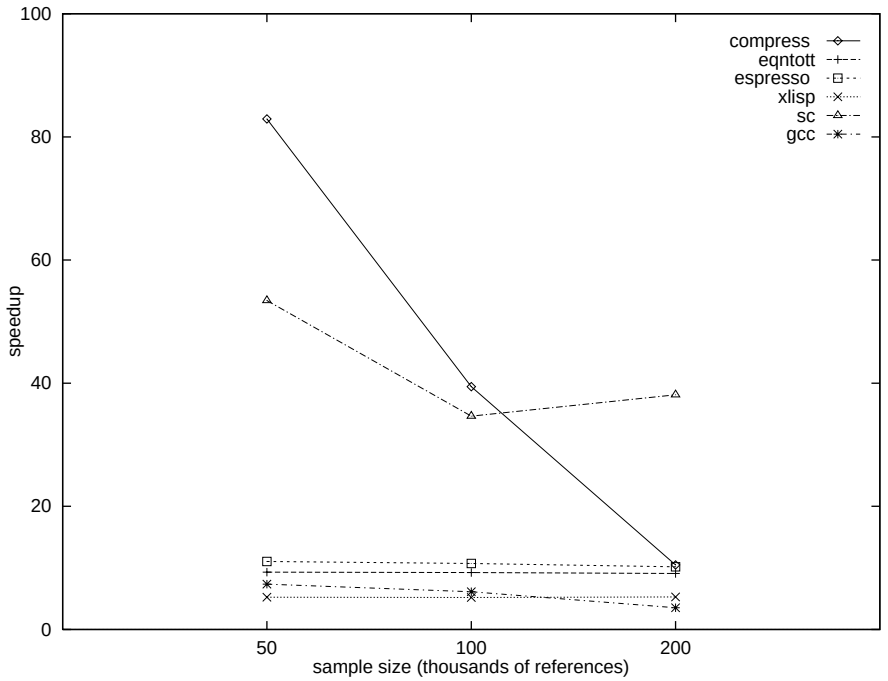

Fig. 5. Speedup of fill-flush approach over no sampling for SPECint92 benchmark suite .

There are several styles of trace collection. One style uses a 'store and forward' approach where the trace is collected, written to disk, then later used for simulation. A drawback of this approach is the required disk space for long-running benchmarks. Another style is 'on-the-fly' simulation that consumes the trace as the trace is being generated. In a situation where on-the-fly simulation is being used, the entire trace is available even if sampling is occurring at the input to the cache simulator. Yet, it is possible to make use of these excluded references. In this approach, statistical sampling is used for the conflict metrics only and $R[B]$ and N are recorded for the whole trace. The miss ratio is then calculated as follows,

$$\rho = 1 - \frac{R[B]}{N} - \frac{D'[C,B,S]}{N_S L_S}. \tag{7}$$

The power of this approach is twofold. First, if the simpler algorithm exploits a hash table or similar search method for stack blocks, it can be made to run much faster than the more complicated basic single–pass algorithm. Secondly, it is known whether any reference recurs for all references inside the sample because the stack is maintained. Therefore, the *state of all references is known*, so this approach is termed a *no-state-loss sampling technique*.

The modified sampling algorithm is shown in Figure 6, where the predicate `sampling` tests if reference $\alpha = w_B(k)$ falls inside a sample. To demonstrate the accuracy of the approach,

1. **if** $\alpha \in S_B(k-1)$ **then**
1.1 **if** `sampling`(k) **then** `do_recurrence`$(\alpha, \mathbf{\Gamma})$
1.2 $S_B(k) \leftarrow$ **repush**$(S_B(k-1), \alpha)$,
2. **else** $S_B(k) \leftarrow$ **push**$(S_B(k-1), \alpha)$
3 **if** `sampling`(k) **then** $N \leftarrow N + 1$

Fig. 6. A no-state-loss approach to extending a single-pass cache simulation algorithm for sampling.

the absolute errors are presented in Figures 7–8.

Comparison of Figures 7–8 to Figures 3–4 reveals several advantages of the no-state-loss approach over the fill-flush approach:

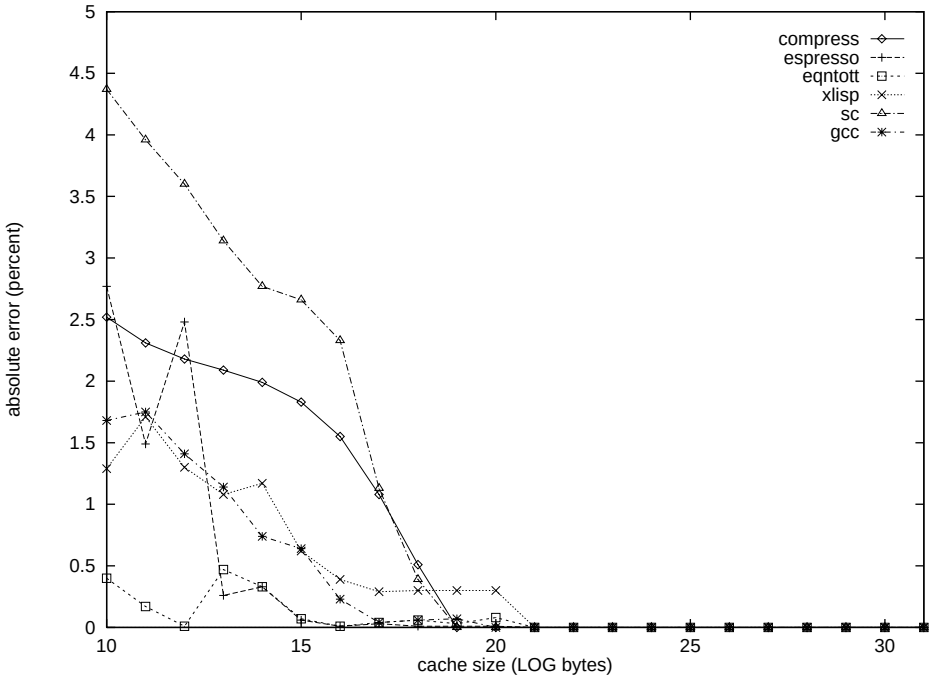

Fig. 7. Absolute error for no-state-loss, direct-mapped caches, $L_S = 100,000$.

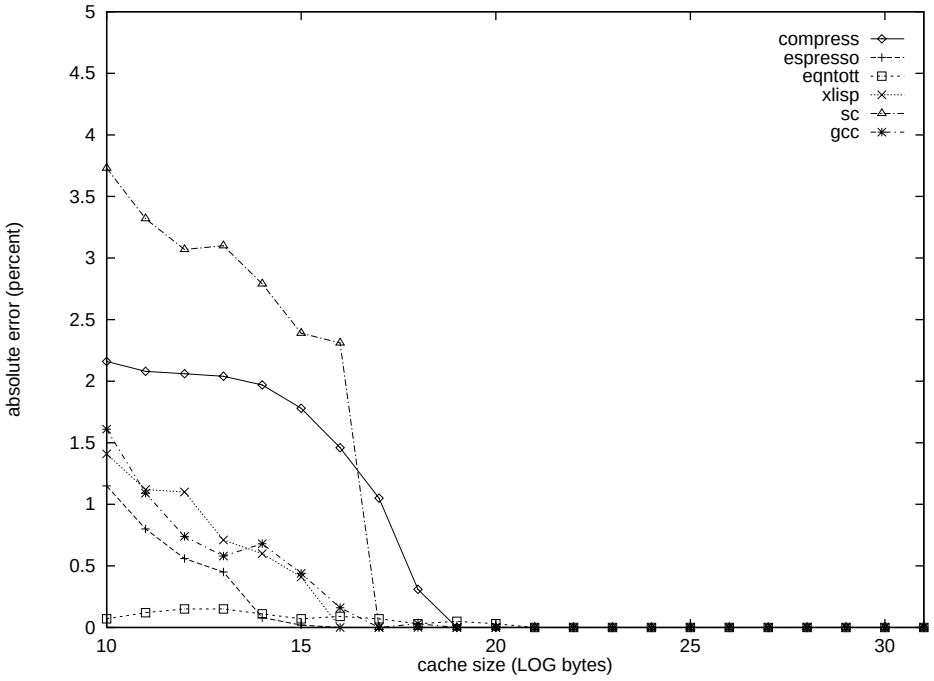

Fig. 8. Absolute error for no-state-loss, fully associative caches, $L_S = 100,000$.

1. In general, the error for no-state-loss is less than 50% of the error of fill-flush ρ.
2. A constant error for large caches ($C > 20$, empirically) is observed for fill-flush while the error in the case of no-state-loss is close to zero.

Both of the advantages can be attributed to the full-trace measurement of $R[B]$. In particular, the second advantage is a direct result of Equations (5) and (6) because f_r is known for no-state-loss. The no-state-loss technique processes all references to some

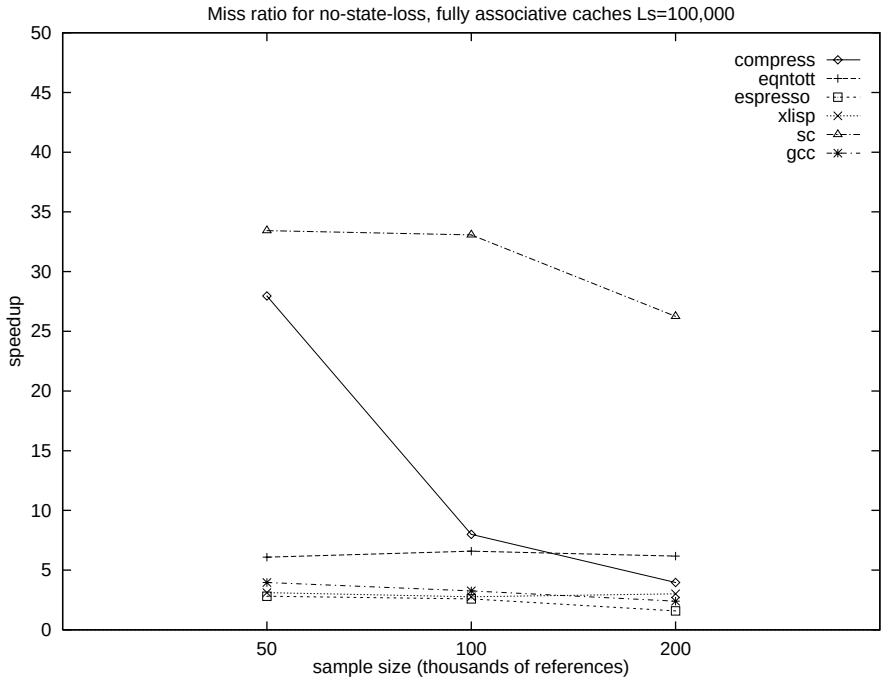

Fig. 9. Speedup of no-state-loss approach over no sampling for SPECint92 benchmark suites .

extent, so it is not as fast as other simple sampling techniques, such as the fill-flush approach. Figure 9 shows the speedup of no-state-loss over full-trace simulation for the SPECint92 benchmarks. Comparison of Figure 9 to Figure 5 reveals that no-state-loss sampling is approximately 2-3 times slower than fill-flush sampling. Therefore, the fill-flush method might be preferable when the execution time of a benchmark is extremely long. Yet, if the accuracy of the results is considered, the fill-flush method produces larger error bounds for medium-sized caches. Thus the fill-flush approach should not be used for the prototyping of first-level cache designs due to their relative sizes. The no-state-loss approach, however, produces higher accuracy than the fill-flush approach for all cache sizes and should be used in the case where accuracy is important.

TABLE I

PERCENTAGE OF TRACE SAMPLED

Benchmark	Percentage
compress	16.5
eqntott	0.6
espresso	1.7
gcc	15.3
li	0.3
sc	1.0

IV. CONCLUSIONS

Trace-driven simulation is a simple way of evaluating memory hierarchies with varying hardware parameters. But to evaluate real world workloads, simulating a few million addresses is not adequate and a very large scale simulation is still costly [8]. In this paper, we presented two single-pass methods that address this efficiency problem. Both techniques use statistical sampling of address traces in combination with the single-pass methods, produce accurate results using a relatively small number of samples and provide excellent speedup over no sampling. The first technique, *fill-flush*, is an efficient adaptation of functional cache simulation techniques. While the new technique, the *no-state-loss* technique, exploits the properties of the single-pass method to achieve very accurate results. However, there exists a tradeoff between speed of evaluation and accuracy. The simulation time using the no-state-loss technique is approximately 2-3 times longer than the equivalent simulation time using the fill-flush method. However, the no-state-loss method produces very accurate results. In fact, this method produced perfect accuracy for large caches and is twice as accurate for smaller caches than the fill-flush approach. Additionally, the no-state-loss technique, unlike other sampling approaches, does maintain a stack between samples so it can be used for other extensions, such as the multiprogramming extension [15].

Acknowledgements

This research was supported by donations from IBM, Intel, and NCR; and, via research funding from the National Science Foundation under grant MIP-9696010.

References

[1] R. L. Mattson, J. Gercsei, D. R. Slutz, and I. L. Traiger, "Evalutation techniques for storage hierarchies," *IBM Systems J.*, vol. 9, no. 2, pp. 78–117, 1970.

[2] A. Agarwal, M. Horowitz, and J. Hennessy, "An analytical cache model," *ACM Trans. Comput. Sys.*, vol. 7, pp. 184–215, May 1989.

[3] A. J. Smith, "A second bibliography on cache memories," *Comput. Architecture News*, vol. 19, pp. 138–153, June 1991.

[4] I. L. Traiger and D. R. Slutz, "One-pass techniques for the evaluation of memory hierarchies," IBM Research Report RJ 892, IBM, San Jose, CA, July 1971.

[5] J. G. Thompson and A. J. Smith, "Efficient (stack) algorithms for analysis of write-back and sector memories," *ACM Trans. Comput. Sys.*, vol. 7, pp. 78–117, Feb. 1989.

[6] M. D. Hill and A. J. Smith, "Evaluating associativity in CPU caches," *IEEE Trans. Comput.*, vol. C-38, pp. 1612–1630, Dec. 1989.

[7] R. E. Kessler, M. D. Hill, and D. A. Wood, "A comparison of trace-sampling techniques for multi-megabyte caches," *IEEE Trans. Comput.*, vol. C-43, pp. 664–675, June 1994.

[8] S. Laha, J. A. Patel, and R. K. Iyer, "Accurate low-cost methods for performance evaluation of cache memory systems," *IEEE Trans. Comput.*, vol. C-37, pp. 1325–1336, Feb. 1988.

[9] H. S. Stone, *High-performance computer architecture*. New York, NY: Addison-Wesley, 1990.

[10] K. R. Kaplan and R. O. Winder, "Cache-based computer systems," *Computer*, vol. 6, pp. 30–36, Mar. 1973.

[11] W. W. Hwu and T. M. Conte, "A simulation study of simultaneous vector prefetch performance in multi-processor memory subsystems," in *Proc. ACM SIGMETRICS '89 Conf. on Measurement and Modeling of Comput. Sys.*, (Berkeley, CA), p. 227, May 1989.

[12] A. J. Smith, "Cache memories," *ACM Computing Surveys*, vol. 14, no. 3, pp. 473–530, 1982.

[13] T. M. Conte and W. W. Hwu, "Benchmark characterization," *IEEE Computer*, pp. 48–56, Jan. 1991.

[14] J. W. C. Fu and J. H. Patel, "How to simulate 100 billion references cheaply," Tech. Rep. CRHC-91–30, Center for Reliable and High-Performance Computing, University of Illinois, Urbana, IL, Nov. 1991.

[15] W. W. Hwu and T. M. Conte, "The susceptibility of programs to context switching," *IEEE Trans. Comput.*, vol. 43, pp. 993–1003, Sept. 1994.

List of Figures

1 The least recently used management policy for a stack, $S_B(k)$ (adapted from Mattson *et al.*). 8
2 Extension of the recurrence/conflict single-pass algorithm to sampling using fill-flush state repair. 11
3 Absolute error for fill-flush, direct-mapped caches, $L_S = 100,000$. 11
4 Absolute error for fill-flush, fully associative caches, $L_S = 100,000$. 13
5 Speedup of fill-flush approach over no sampling for SPECint92 benchmark suite 13
6 A no-state-loss approach to extending a single-pass cache simulation algorithm for sampling. 14
7 Absolute error for no-state-loss, direct-mapped caches, $L_S = 100,000$. 15
8 Absolute error for no-state-loss, fully associative caches, $L_S = 100,000$. . . . 15
9 Speedup of no-state-loss approach over no sampling for SPECint92 benchmark suites 16

Please send all correspondence and/or return of proofs to:

Thomas M. Conte

Department of Electrical and Computer Engineering

North Carolina State University

Box 7911

Raleigh, North Carolina 27695-7911