December 1991

UILU-ENG-91-2251
CRHC-91-30

Center for Reliable and High-Performance Computing

HOW TO SIMULATE 10 BILLION REFERENCES CHEAPLY

John W. C. Fu and **Janak H. Patel**

Coordinated Science Laboratory
College of Engineering
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Approved for Public Release. Distribution Unlimited.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CRHC-91-30	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (*If applicable*) N/A	7a. NAME OF MONITORING ORGANIZATION Semiconductor Research Corporation Joint Services (Office of Naval Research)
6c. ADDRESS (*City, State, and ZIP Code*) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (*City, State, and ZIP Code*) Research Triangle Park, NC 27709 Arlington VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 7a 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*) 7b	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (*Include Security Classification*)

How to Simulate 100 Billion References Cheaply

12. PERSONAL AUTHOR(S)

Fu, John W. C. and Janak H. Patel

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Technical	FROM _____ TO _____	91/11/20	22

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)

performance evaluation, trace-driven simulation, cache memories, numerical benchmarks, sampling

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

Trace driven simulation is a well known method for evaluating computer architectures and the technique of choice in most cache studies. Ideally, a trace should contain all the memory references made by a program but this is usually impractically expensive for large programs because of the trace storage and computation costs. Statistical sampling is often use to reduce the data set size and is beginning to attract wide attention as a method for reducing trace driven simulation costs.

This paper presents a prediction method to solve the cold-start or fill reference problem when simulating with a sampled trace. We show how taking a number of short samples from a much larger trace can capture the characteristics of the larger trace and show single and multiple cache simulation results using our prediction method.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (*Include Area Code*) / 22c. OFFICE SYMBOL

How to Simulate 100 Billion References Cheaply

John W. C. Fu

Janak H. Patel

ABSTRACT

Trace driven simulation is a well known method for evaluating computer architectures and the technique of choice in most cache studies. Ideally, a trace should contain all the memory references made by a program but this is usually impractically expensive for large programs because of the trace storage and computation costs. Statistical sampling is often use to reduce the data set size and is beginning to attract wide attention as a method for reducing trace driven simulation costs. This paper presents a prediction method to solve the cold-start or fill reference problem when simulating with a sampled trace. This approach allows detailed simulation of cache miss events. We show how taking a number of short samples from a much larger trace can capture the characteristics of the larger trace and show single and multiple cache simulation results using our prediction method.

KEYWORDS: performance evaluation, trace-driven simulation, cache memories, numerical benchmarks, sampling.

1. Introduction

Trace driven simulation is a well known method for evaluating computer architectures and the technique of choice in most cache studies. There are essentially two steps to trace driven simulation. First, a trace is generated by collecting information from the activity of interest and in the second step, a model is simulated using the collected trace as the input. This paper is mainly concerned with cache simulation, so a trace usually means a set of memory references collected from program executions.

Ideally, a trace should contain all the memory references made by a program. However, a major drawback in trace driven simulation is that for non-trivial programs the trace storage cost, and computation costs for generating and simulating the trace can be impractically expensive. For example, TRACK, one of the shorter PERFECT benchmark programs [Berr89] generates a trace length of 250 million references in about 40 hours on a Alliant FX/8 and uses 6 GB of disk storage[1]. Simulating a basic cache organization with this trace requires about 40 hours on the Alliant FX/8. This becomes prohibitively expensive if many combinations of cache parameters need to be investigated using multiple benchmarks. A simple program will generate a shorter trace but it is generally agreed that results with such programs can be misleading. This paper presents a method to reduce trace driven simulation costs with non-trivial programs.

A number of researchers have proposed methods to reduce storage and computation costs. Storage cost can be reduced by compressing the trace information [Samp89] or by filtering out particular information [Puza85, WaBa90]. Trace compression reduces the storage cost by encoding the trace information. Compression factors of 10 to 20 are typical and can be effective for traces of a few million references in length but is insufficient if a program generates a billion references and several traces are necessary for simulation. A filtered trace is generated by simulating a cache of a particular size with the original trace and retaining only the miss references. Trace filtering can result in a much smaller trace as the number of misses is usually a small fraction of the total references but the filtered trace is limited to cache studies as it has been biased by the filter. Trace filtering is an effective technique if only cache sizes larger than the

[1] Trace generation is through a detailed emulation of the Alliant FX/8. Reference information includes processor id, address, type, size, and vector length. This is described in § 2.

filter is of interest.

Trace storage costs can be eliminated by simulating the model immediately after collecting or generating the references [SoZe88, BoKW90]. This requires references to be regenerated for each simulation of the model and sometimes trace generation can cost more than model simulation. Inline tracing [StFu89, EKKL90] may reduce the trace generation cost but this can still be expensive if the study evaluates 50 to 100 cache configurations. Multi-cache simulation techniques [HiSim89, WaBa90, CoHw90] reduce computation costs by processing multiple cache configurations in a single pass of the the trace but these are currently limited to simple cache organizations and not applicable to more complex memory models or non-cache models.

An alternative approach to reducing trace driven simulation costs is to collect a smaller representative trace. Sampling is a general statistical technique often used in experiments with a large data set to obtain a smaller representative set. For instance, sampling has been successfully applied in performance analysis of real computer systems [ClBK88, DiIy91]. In these papers, the parameters of interest are sampled to obtained a sampled data set and analysis performed on this smaller set. For example, Clark in [ClBK88] uses hardware instrumentation to periodically record the VAX 8800 processor's microcode program counter. This sampled set is then used to analyze the processor performance. Similarly, trace sampling reduces a program trace size by periodically sampling its execution. The smaller sampled trace is then used as the input for model simulation.

Trace sampling is beginning to attract wide interest since it enables large realistic programs to be used in simulation and is a practical method for collecting traces in real-time from a machine. Early work in applying sampling to cache simulation [LaPI88] showed that a sampled trace has the same characteristics as the trace being sampled. Simulating a cache with a sampled trace poses a cold-start or fill problem. The first reference to a cache location in a sample fills the cache location. But it is unknown if this is truly a miss or hit. Several models were presented to estimate these fill references as misses in [LaPI88] and more recently in [Pate90] and [WoHK91]. Trace samples has been used in studying cache behavior in [FuPa91] and [AgSH86]. In [AgSH86] data was collected from a real machine by patching the micro-code. This

study used very large samples to overcome the cold-start effect. Traces sampled using this method has been used in a number of studies.

Previous models used with sampled traces only estimated the fraction of fill references that are misses and can only calculate the cache miss ratio. These models are inadequate in studies that require more detail simulation of cache miss events such as a multiprocessor. This type of simulation requires each fill reference to be identified as a hit or a miss. In this paper we present a simple method to predict the state of fill references based on the miss history of the reference stream. We show results that compare the distribution of inter-miss distances with a large continuous trace and a sampled trace for uniprocessor and multiprocessor caches.

In the next section we introduce the sampling method and describe how the traces in the reported experiments were generated. In § 3, we look at how using a single larger section from a program execution can lead to incorrect conclusions. Smith has shown in [Smit85] that a cache can have a wide range of performance with different workloads. We show that different sections of a program's execution can have varying characteristics but a sampled trace of less than 10% can capture these characteristics. Section 4 discusses cache simulation with a sampled trace and present our prediction method and show results for the uniprocessor and multiprocessor caches. Concluding remarks are made in § 5.

2. Trace Generation

This paper uses traces collected from a detailed emulation of an Alliant FX/8 vector multiprocessor system [Alli85]. The Alliant emulator is an accurate cycle by cycle model of the Alliant FX/8. Programs to be emulated are compiled with the Alliant FX Fortran compiler that does both program transformation and vectorization. Operating system and library routine calls made by the program are also emulated. The memory references produced by an emulated program are the same as the memory references produced by a program executed on an Alliant FX/8.

There are two step in sampled trace driven simulation. First a sampled trace is generated by taking *s*

samples of r references from a program execution.[2] This is illustrated in Figure 1. In the second step the model is simulated using the sampled trace.

For the sampling experiments we use three programs from the Perfect Club benchmark set: ADM, BDNA and TRACK. The complete trace lengths of these programs range from 0.25 to 2 billion references making the use of their complete traces impractical. To evaluate our sampling method we have collected continuous traces of over 100 million references for uniprocessor executions and over 50 million references for multiprocessor executions. A sampled trace is generated by sampling a continuous trace and the results from simulating with the two traces are compared to evaluate the sampling method.

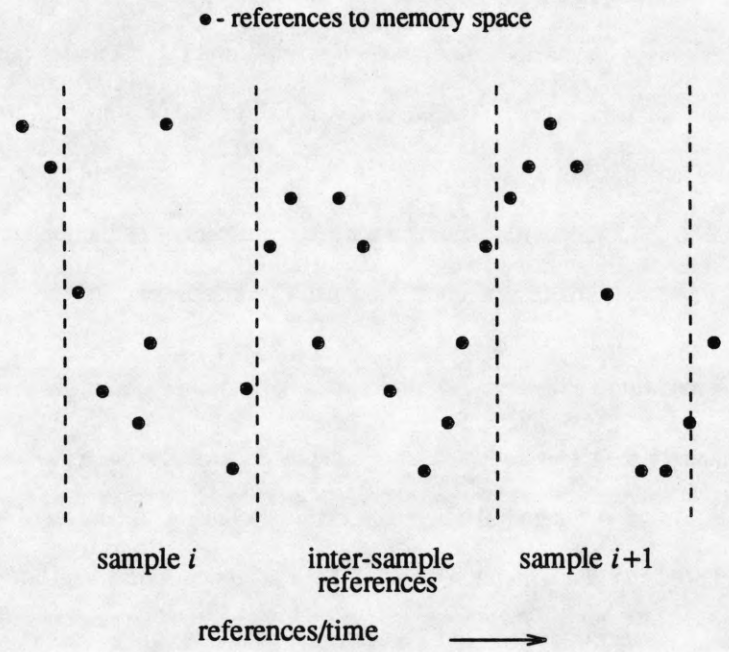

Figure 1: Trace Sampling. Time based trace sampling collects r consecutive references for each sample i, where $0<i\leq s$. A sampled trace has $s \times r$ references. This paper uses $s=40$ and $r=200{,}000$. References within the inter-sample interval are ignored and do not appear in the sampled trace.

[2] An alternative method only collects references to a particular set of cache sets [LiPe90]. This approach has similarities to both trace sampling and trace filtering but is not not consider for this paper.

A continuous trace is collected from the initial period of program execution and is not suppose to represent its complete execution. The size and length of the continuous traces were constrained by storage costs. The initial period of program execution was selected for the continuous traces as this tended to have the most transient reference behavior. Obtaining a representative sampled trace from within this period is likely to be more difficult than from a continuous trace (taken from the middle of the execution) and a good test for the sampling method. Table 1 shows some characteristics of the continuous traces.

2.1. Sampled Trace Generation

Results in [LaPI88], show that s=35 and r=100,000 is sufficient to obtain a good representative sampled trace and an accurate cache result regardless of a program's complete trace length. Furthermore, it was shown that the inter-sample interval can be kept constant unless periodic behavior is suspected in a

Program	continuous†	sampled‡	% scalar	application
ADM	112	7.0	60.0	air pollution, fluid dynamics
BDNA	108	7.4	63.9	nucleic acid simulation, molecular dynamics
TRACK	113	7.1	95.7	missile tracking, signal processing

a) Uniprocessor

Program	continuous†	sampled‡	% scalar	% distribution of references across processors							
				0	1	2	3	4	5	6	7
ADM	56	14.3	76.5	22.8	12.4	12.3	12.5	8.3	8.5	7.5	15.8
BDNA	60	13.3	63.4	7.6	7.0	6.6	6.5	6.2	5.9	7.8	52.4
TRACK	87	9.2	96.6	9.3	9.1	5.7	59.3	2.5	2.4	2.4	9.4

b) Multiprocessor

† in million of references ‡ as a percentage of the continuous trace

Table 1: Continuous Trace Characteristics. Due to program size large continuous traces are used for the sampling experiments. The traces are for single and multiprocessor executions. The table shows the fraction of scalar references for the continuous trace, and the distribution of the references over the the processors for the multiprocessor trace. Each sampled trace is generated from the continuous trace and its relative size to the continuous trace is shown. A reference in a trace consists of the processor id, address, reference type, access type, time of access, size of access and stride. Each continuous trace takes over 250MB of disk storage in its uncompressed format.

program's execution. This paper uses, s=40 and r=200,000 as the sampling parameters. Accuracy is expected to improve if the number of samples is increased. The use of s=40 is a tradeoff between increased accuracy and storage costs. The increase in the sample size is because of the new simulation model described in § 4.

Numerical programs can exhibit periodic behavior since they typically process data organized as matrices. Therefore, the sample start points were semi-randomly generated as follows. The total number of references in a continuous trace is divided by the number of samples, s, to be collected. This gives an approximate starting point S_i for a sample i. The actual sample start point is then randomly selected within the range of $S_i \pm x$ where x is some constant chosen so that the samples do not overlap.

3. Sampled Trace Accuracy

An alternative to taking a number of small samples from a larger trace is to take an single large section. In this section we show some results from simple experiments that compare the characteristics of two arbitrary sections and a sampled trace taken from a continuous trace. The first trace section is taken from the beginning of the continuous trace. This period of execution has been used in some previous cache studies. The second trace section is from around the mid-point and attempts to capture more representative[3] behavior of the continuous trace. The sampled trace was generated as described in § 2.1. The experiments use the uniprocessor and multiprocessor continuous traces for BDNA.

Table 2 compares some characteristics of the continuous uniprocessor trace, the two trace sections, and a sampled trace. The two trace sections show more deviation from the characteristics of the continuous trace than the sampled trace. The first trace section has different characteristics from the continuous trace. The results suggest that the initial period of program execution is likely to result in a poor representative trace. A more accurate approach may be to use a section after a period of execution. For example, the second trace section consists of the 56*th* to 96*th* million references. This second trace section is more representative than the first trace section but there is still a significant difference between the percentage of

[3] We informally say representative is having similar characteristics in the parameters of interest as the trace being sampled e.g. same ratio of read to writes, scalar to vector etc.

TRACE	% vectors	vector		%scalars	scalars		
		% read	% write		% read	% write	% test&set
section 1 (0-30)†	3.75	59.73	40.27	96.25	73.72	24.50	1.78
section 2 (56-96)†	52.42	63.00	37.00	47.58	71.50	28.50	0.0
sampled trace	35.31	63.55	36.45	64.69	72.30	26.92	0.78
continuous trace	36.13	63.14	36.69	63.87	72.37	26.84	0.78

a) Uniprocessor trace

proc	% vectors				% scalars			
	contin-uous	section 1 (0-20)†	section 2 (27-47)†	sampled trace	contin-uous	section 1 (0-20)†	section 2 (27-47)†	sampled trace
0	13.89	2.05	14.48	13.76	5.75	1.73	11.24	4.82
1	13.41	0.88	13.93	13.27	5.12	0.47	11.03	4.67
2	12.87	0.89	13.31	12.79	4.76	0.01	10.66	4.55
3	12.32	0.88	12.71	12.33	4.77	0.26	10.46	4.39
4	11.75	0.89	11.96	11.70	4.51	0.12	9.91	4.33
5	11.24	0.87	11.39	11.27	4.34	0.01	9.60	4.26
6	10.75	0.89	10.88	10.76	6.93	4.53	12.14	7.39
7	13.78	92.64	11.33	14.12	63.83	92.85	24.95	65.59

b) Multiprocessor trace

† million of references within a trace section

Table 2: Reference Characteristics for BDNA. This compares some basic characteristics of the continuous trace, 2 trace sections and a sampled trace. With the uniprocessor trace we compare the vectorization and the ratio of read and write references. In the multiprocessor trace we look at the distribution of the references across the processors.

vector references in the second trace section and in the continuous trace. The sampled trace characteristics are similar to the continuous trace. Furthermore, the sampled trace is smaller than both trace sections.

A similar experiment was performed with the multiprocessor trace of BDNA. The distribution of vector and scalar references for the continuous multiprocessor trace of BDNA, two trace sections and the sampled trace are shown in Table 2b. The results again show that neither of the two trace sections represent the continuous trace. The first trace section has completely different characteristics from the continuous trace. The second trace section again improves on the first trace section but cannot be considered similar to the continuous trace since there is significant difference between the scalar reference distribution in the trace section and the continuous trace. The sampled trace has similar characteristics as the continuous trace and is smaller than both trace sections. Furthermore, using a trace section has the problem of

determining the most representative period, if one exists, for collection but the the start points for each sample for a sampled trace can be easily determined.

Figure 2 shows the cumulative and instantaneous miss ratios for the continuous uniprocessor trace of BDNA for a 128Kbyte cache with a set size of 2 and a block size of 32 bytes. The cache was cleared at the start of the simulation and 1 million references used to warm up the cache. The cumulative miss ratio is the miss ratio of the cache after the simulation of a particular number of references. For example, after simulating 50 and 108 million references the cumulative or average cache miss ratios are 0.0033 and 0.0216, respectively. The cumulative cache miss ratio is a metric usually reported in cache studies. The instantaneous cache miss ratio is recorded over each interval of 1 million references.

The behavior of the cumulative and instantaneous miss ratios show that using a trace section from the continuous trace will lead to a wide range of cache results. If the first trace section of the continuous trace is used for simulation then the average miss ratio is 0.0035. The resulting cache miss ratio is very

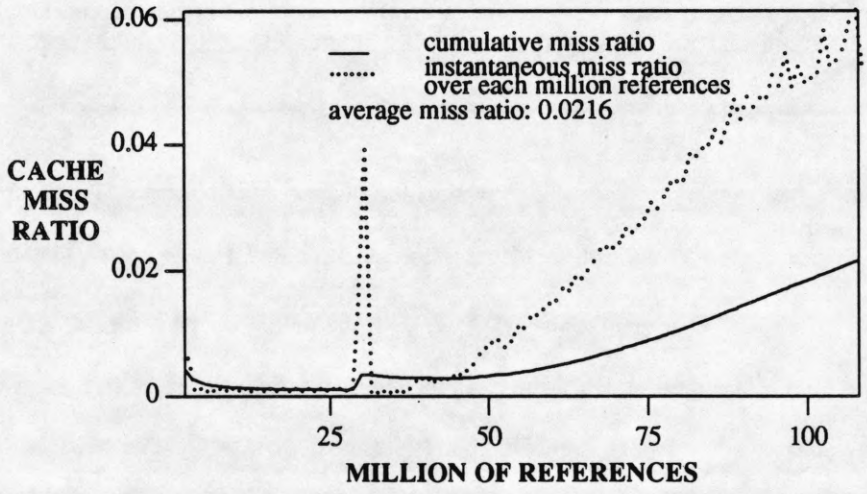

Figure 2: Cumulative and Instantaneous Miss Ratios. This shows the cumulative and instantaneous miss ratios for the continuous trace of BDNA. The cumulative miss ratio is the usual definition of the miss ratio i.e. *number of misses/number of references*. The instantaneous miss ratio is taken over a number consecutive references, in this case over each 1 million references i.e. *number of misses*/1,000,000.

different if the second trace section is used. Assuming that the second trace section is large enough that cold start does not dominate, then the resulting cache miss ratio is about 0.0196. The wide variance in the instantaneous miss ratio suggests that no trace section is likely to lead to a representative result.

The next section describes how a sampled trace can be used to obtain an accurate result.

4. Sampled Trace Driven Simulation

Whether a cache reference is a hit or a miss depends on the state of the particular cache location when the reference is made. If the cache location already holds the memory block being referenced then it is a hit otherwise it is a miss. When simulating a cache with a sampled trace, the state of the cache before simulating a particular sample is always unknown since the references in the inter-sample interval are unavailable. Maintaining the state of the cache after simulating one sample to simulate with the next sample may lead to inaccurate results since the inter-sample interval may be a few million references in length. To establish a consistent cache state, though not necessarily the true state, the cache is always cleared before simulating a sample. A reference that maps to a cache location that is cleared causes the cache location to be filled. This reference, however, cannot be determined as a hit or a miss since the true state of the cache location before the sample is unknown. This reference is referred to as a *fill*. A reference that maps to a cache location previously filled is called *significant*. A significant reference can always be determined as a hit or miss and the same reference always results in the same cache state for both the sample and continuous trace. For n samples the estimated cache miss ratio is:

$$\frac{\sum_{i=1}^{i=n} \textit{signifcant misses in sample } i + p \times \textit{fills in sample } i}{\sum_{i=1}^{i=n} \textit{total references in sample } i}$$

where p is the fraction of fills that are true misses. Calculating the cache performance by assuming fills are misses always over-estimates the cache miss ratio. When simulating a cache model with a sampled trace the fill references must be resolved as cache hits or misses for an accurate result.

4.1. Fill Prediction Model

Models proposed in [LaPI88] and more recently in [Ref90] and [WoHK91] all calculate the fraction of the fill references, p, that are true misses and can only obtain the cache miss ratio metric. The miss ratio is a useful measure of cache performance but it is an inadequate performance index when other system components are being simulated. For example, in a multiprocessor system the miss ratio does not account for bus and memory conflicts. Detailed simulation of the cache miss events is desirable in these cases. Models that only calculate p are inadequate since detailed simulation requires a fill reference to be identified as a hit or miss. This section presents a simple method to predict a fill reference as a cache hit or miss. This method was initially proposed in [Ref90].

Figure 3 shows a sequence of hits (h_i), misses (m_i) and a single fill (f). The inter-miss distance or miss distance, d, of a sequence of significant references, is the number of references per miss. If a fill occurs, that particular miss distance cannot be determined unless the fill is resolved as a hit or a miss. A distribution of the miss distances is a more detailed measure of a cache's performance than the miss ratio and is the metric used in the reported experiments. (Note, that the reciprocal of the mean miss distance, $\bar{d}$, is the miss ratio). Comparing the miss distance distribution is a simple method for evaluating the success of the prediction model and more rigorous than simply comparing the average miss ratios.

$$\ldots, m_1,\ h_1,\ h_2,\ h_3,\ h_4,\ h_5,\ m_2,\ h_6,\ f,\ h_7,\ m_3,\ h_8,\ h_9,\ m_4,\ h_{10},\ h_{11},\ h_{12},\ m_5, \cdots$$

$$\longleftarrow \quad d{=}6 \quad \longrightarrow$$

Figure 3: Miss Distance. The inter-miss or miss distance d, is the number of references between misses including the first miss. In the above, the miss distance of 6 includes the miss m_1 and hits h_1 to h_5. The fill reference f means the second miss distance cannot be determined. If f is a truly a miss then the second miss distance is m_2 and h_6 else it also includes f, since its a hit and h_7. If the miss miss distance is $\bar{d}$ then the mean miss ratio is $1/d$.

4.2. Method

Observations of the miss distance behavior of the traces indicate that references do not randomly hit and miss in the cache but depend, not surprisingly, on previous references and the contents of the cache. Our method attempts to capture some simple history of references and use this to predict the fill references. This is similar to techniques used in branch prediction using dynamic history tables [LeSm84].

Each sample is divided into a *priming interval* and a *evaluation interval*[4]. The priming interval is initially simulated to warm up the cache by creating a set of filled cache locations. These filled locations reduce the number of fills (and increase the number of significant references) in the evaluation interval. The evaluation interval is simulated and each fill reference is predicted as a hit or miss using a dynamic history of previous miss distances and the contents of the cache. Only miss distances recorded in the evaluation interval are used to produce the miss distance distribution.

The method is as follows, for each sample:

- Simulate the priming interval and start the history table. The history table is a finite list of the most recent miss distances.
- At each miss, the number of references are recorded until the next miss or the next fill. This is the miss distance *d*. If a miss occurs, then *d* is recorded. If a fill occurs in the priming interval then it is ignored. If the fill occurs during the evaluation interval then it is predicted as follows:

(1) If the history table is empty i.e. no significant misses recorded then predict a *hit*. A hit is predicted because with an empty history table the sequence of references are likely to be hits.

(2) else if *d* is within the range of the miss distances held in the history table then predict a miss. This attempts to capture the behavior where blocks are being replaced. Tracing the miss distances indicate that there are regular patterns where a block is loaded followed by a set of hits, followed by another block replacement. By recording sets of miss distances we assume that these can be used to predict

[4] The results presented here have the same number of references for each interval. Results not presented in this paper due to space show that a it is less sensitive to a smaller priming interval than to a larger evaluation interval i.e. the evaluation interval should be large at the expense of the priming interval.

the fill reference. Results, not reported here, indicate that the size of the history table used is quite small. Recording more than 3 distances had little effect on the overal result.

(3) else if a prediction cannot be made based on the history then the contents of the cache is searched. If the adjacent sets (to the set being filled) hold addresses of adjacent memory blocks to the memory block being loaded then a *hit* is predicted, else a *miss* is predicted. This assumes that if an adjacent memory block is in the cache then there is good probability that the fill block should already be in the cache i.e. a hit is predicted because we assume locality. It is possible that these adjacent memory blocks are replacement blocks but this will be reflected in the history table (§ 5 indicates that this may not be true for a small cacke).

(4) else if non of the prediction conditions are met then we predict a *miss* by default.

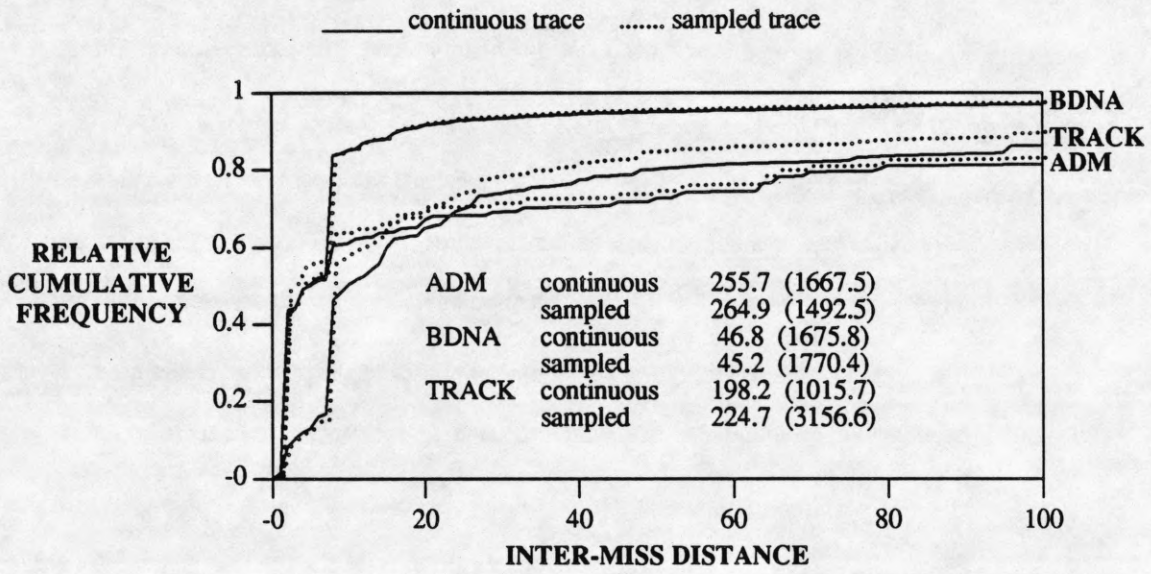

Figure 4: Unified Cache Miss Distance Distribution. The plot shows the relative cumulative frequency distribution of miss distances and the corresponding mean and standard deviation for the continuous and the sampled traces with a 128KB cache with 32 byte block and a set size of 2. For example, BDNA's distribution shows that 44% of its miss distances are ≤ 2 and 84% of the miss distances are ≤ 8.

The relative cumulative frequency distribution of the miss distances from simulating with a 128KB cache is shown in Figure 4 with the mean and the standard deviation. The prediction method is accurate in predicting both the mean and standard deviation. Fills incorrectly predicted do not have a significant effect on the distribution. (Table 5 show that 60%-80% of the fills were predicted correctly.) The largest deviations between the continuous and sampled trace distributions in Figure 3 occurs after a sharp transient. A sharp transient in the distribution is when a large percentage of the miss distances is of a particular value. For example, the large transient in TRACK's continuous trace distribution is because 30% of the distances have a value of 8, but the sampled distribution predicted 40%. Since the transient accounts for a large percentage of the distances it is not surprising that it can incur the largest sampling and prediction errors.

Table 3 compares the results obtain using no prediction and random prediction with the prediction result for BDNA. In no prediction we collect only known or significant miss distances for the miss distance distribution i.e. when a fill reference occurs the miss distance is ignored. The result with no prediction while less accurate than the prediction case, is nevertheless acceptable. However, no prediction is only useful if only a cache miss ratio is required since in detailed miss event simulation fills cannot be ignored. Furthermore, no prediction may produce very a large error if there are insufficient significant miss distances. In random assignment when a fill occurs we generate a random number to assign the fill as a hit or miss. The results show that for this particular method of random assignment a large error occurs.

BDNA	inter-miss distribution	
	mean	std. dev.
continuous trace	46.8	1675.8
sampled trace, *no predication*	57.4	2136.4
sampled trace, *with predication*	45.2	1770.4
sampled trace, *random assignment*	199.3	2020.6

Table 3: Sampled trace with and without predictions and random assignment. This shows the results in using no prediction and one form of random prediction. Note that for detailed simulation of events no prediction is not usable since we cannot ignore the fill references.

In cache studies the cache performance (i.e. the miss ratio) is often presented as a function of the cache size. Figure 4 shows the cache performance as a function of the cache size for three traces, continuous, sampled and sampled_100%, for each benchmark. The sampled_100% result is the sampled trace with 100% prediction success. The difference between the continuous and sampled_100% result is due to sampling error and the difference between the sampled and sampled_100% result is due to prediction error. The miss ratio is calculated as $1/\bar{d}$ from the distributions shown in Table 4.

The results for ADM show that the sampled trace can be used to produce accurate results for the cache sizes except for the 1MB cache. With the 1MB cache there are very few fills and the prediction

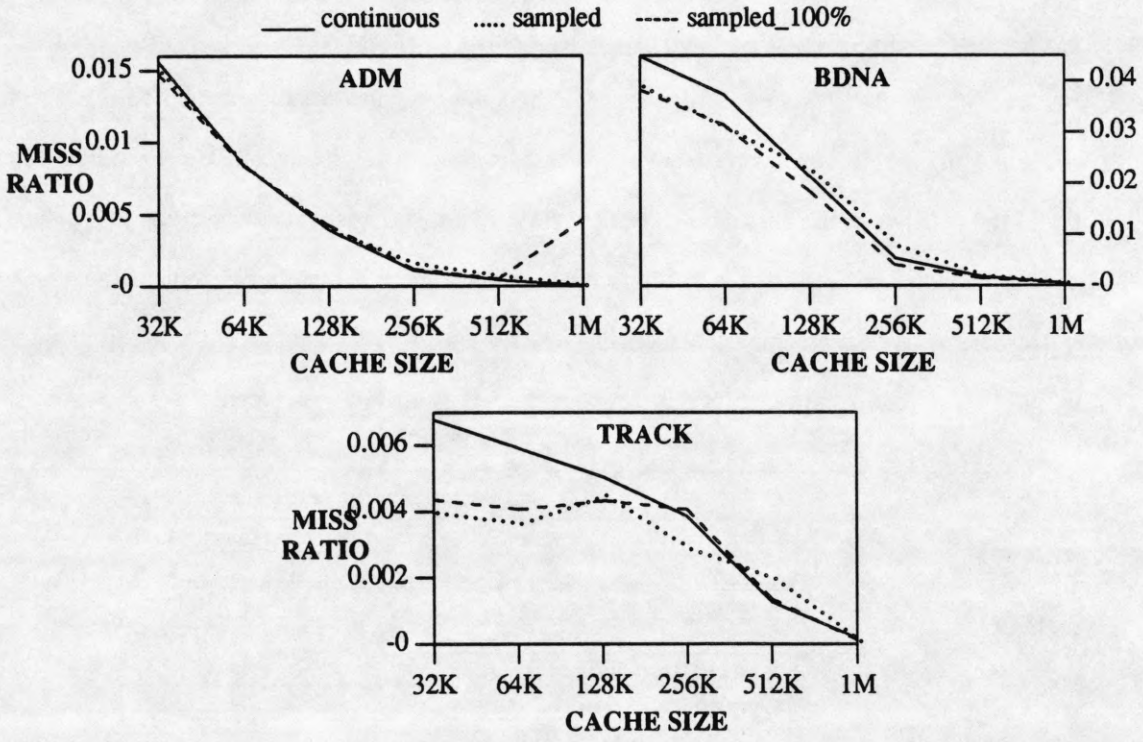

Figure 5: Increasing Cache Size. Shows the prediction results for increasing cache size from 32K to 1M byte caches for three trace types: continuous trace (solid), sampled trace (dotted) and sampled_100% (dashed). The sampled_100% is the sampled trace with 100% prediction success. The miss ratio is $1/\bar{d}$ shown in Table 5.

method did not predict any of them. Interestingly, the sampled_100% results also has a very high error which is not true for TRACK and BDNA. The results for BDNA show more inaccuracy for the smaller cache than for the large caches which is somewhat surprising since a smaller cache has less fills and therefore less effect on the result. However, the mean and standard deviation show the sampling as the source of most of the error. The most unpredictable results occur with TRACK where there seems to be larger errors. However, some caution should be taken in interpreting these results since TRACK has a hit ratio > 99% for all the cache sizes. While the relative error in the miss ratio for the different traces seen significant whether it is, for practical purposes, is questionable. Similarly, in both BDNA and ADM the plots show good results for the very large caches (256KB-1MB) but the relative error in the mean and standard deviation in Table 4 is quite large.

4.3. Prediction Success

The success of each prediction condition is shown in Table 5 for a small cache (32 KB) and a large cache (128KB). For each trace, the percentage of fills predicted with a particular condition is given in the first column and its success in the second column. The weighted average prediction success is also shown.

TRACES	32KB	64KB	256KB	512KB	1MB
ADM	64.1 (315.9)	118.5 (501.4)	929.01 (9076.7)	2408.9 (17638.6)	21812.3 (1523322.6)
	67.7 (392.5)	121.9 (796.6)	675.1 (6787.7)	1558.5 (11101.7)	†
	67.8 (330.7)	119.6 (502.9)	851.6 (8736.1)	2000.9 (16369.1)	215.8 (1362.2)
BDNA	22.3 (627.9)	26.9 (1080.4)	187.6 (3508.7)	570.6 (8040.0)	2380.2 (18841.6)
	25.9 (954.2)	32.8 (1384.7)	133.3 (4916.4)	574.7 (26068.7)	2751.8 (48689.6)
	26.2 (720.1)	32.0 (1401.2)	250.2 (4340.5)	720.4 (9158.4)	3144.6 (21462.8)
TRACK	147.7 (636.4)	169.0 (796.6)	260.2 (1577.96)	801.8 (4311.5)	6404.9 (108507.8)
	254.5 (1545.9)	277.6 (1220.2)	352.6 (9502.2)	518.3 (24695.1)	18887.9 (372314.9)
	229.7 (883.1)	245.7 (681.2)	246.3 (1501.7)	743.8 (3764.9)	10096.9 (142109.53)

† - no misses predicted

Table 4: Mean and Standard Deviations. The table shows the mean and distribution for cache sizes from 32KB to 1MB caches. The first line is the continuous trace, the second is the sampled trace and the last line is the sampled_100% trace, where the sampled_100% result is the sampled trace with 100% prediction success. The miss ratio in Figure 4 is the reciprocal of the mean miss distance.

prediction conditions (prediction)	32KB						128KB					
	ADM		BDNA		TRACK		ADM		BDNA		TRACK	
	%	true	%	true	%	true	%	true	%	true	%	true
no misses (hit)	10	66	13	70	14	96	37	90	37	89	70	60
history table (miss)	26	71	29	82	47	93	10	20	10	30	14	82
block search (hit)	46	42	40	45	32	15	49	89	51	91	11	40
default (miss)	18	72	18	72	7	10	4	19	2	28	5	86
weighted avg.		57		64		62		80		83		62

Table 5: Prediction Success. The first column is the percentage of fills that met the condition and the second column is the percentage of these fills predicted correctly by the condition. More misses occur in the small cache and the history table is quite successful predicting fills but the block search is less successful since there is more replacement. For a large cache the reverse is generally true.

Predicting a hit when there is insufficient history is quite successful in the small cache with success rates ranging from 66%-96%, but less than 15% of the fills appear during this period. In the large cache the success rate is in a similar range but more of the fills appear during this period. This is not surprising since the hit rate is higher than 99.9% for a 128KB cache and the likelyhood of fills being true hits is high. The prediction success with the history table and the cache block search similarly depends on whether there are a large number of misses. In the small cache, prediction with the history table has 71%-93% success rate. A small cache has have more misses and hence more dynamic history. A large cache has a smaller number of misses so the history table is less likely to contain useful information and the success rate is quite poor for ADM and BDNA, but the number of references predicted with the history table is below 15%. The block search assumes that blocks are not being replaced and this not surprisingly is untrue in the small cache where the success is below 50% and very poor for TRACK. This result with the block search suggests that more more information is necessary to make this prediction condition successful for small caches. The block search condition is more successful with the large cache.

The general trend is that the history table is successful for small caches and the block search better for large caches. However, TRACK does not entirely follow this trend since in the large cache the history table shows high success but is less than 50% for the block search. The reason for this is currently under

investigation.

4.4. Multicache Prediction

In a system with multiple caches some mechanism is usually necessary to maintain data coherence among the caches. To a particular cache there three possible reference types; read, write and coherence, where a coherence reference hit results in cache block invalidation or updated. This section discusses how a coherence reference can be used to resolve fill references when simulating with a sampled trace. The discussion uses write invalidation as the coherence scheme but the concept equally applies to write update. Note that in the following description a cleared cache block refers to a block that was cleared at the start of a sample and an invalidated cache block refers to a block cleared for coherence.

Consider a multiprocessor system where each processor P_i has a private cache C_i and a processor can only directly access its own cache. A common bus is used to broadcast write accesses to all other caches. Consider the following time ordered sequence of references:

$$\ldots\ldots,R_0^i,W_0^j,R_0^i,\ldots\ldots$$

where W_0^j is a write to memory block 0 by P_j its cache C_j and R_0^i is a read to memory block 0 by P_i.

In the continuous trace, W_0^j writes to memory block 0 and broadcasts the write to the other caches where it causes the invalidation of memory block 0 in C_i. The subsequent read reference to memory block 0 is a miss in C_i due to the invalidation.

Consider a sample where W_0^j is the first reference to block 0 and maps it to a cleared cache block. Thus, reference W_0^j is a fill reference in C_j. The corresponding coherence reference in C_i is also to a cleared cache block. If the address of the coherence reference, due to W_0^j, is held in C_i, it can be used to determine that the next reference, R_0^i, is a miss.

In our simulation with sampled traces, the coherence addresses to a set that has at least one cleared cache block are held in a list for that set. If a read or write address cannot be found in the cache tags of the set, and at least one of the cache blocks in the set is cleared, the addresses in the coherence reference list are checked. If the read or write address is found in the coherence address list, the read or write access

must be a miss. This is because even if the memory block had been previously referenced in the continuous trace it would have been invalidated by the coherence reference. Of course, had the memory block not been in the cache to be invalidated, the access is still a cache miss.

The above only applies to a set with at least one cleared cache block because it is unknown which memory block will be loaded into the cache. When a cache set is fully loaded or primed, subsequent references are all significant and the coherence address list can be removed. A reference that loads a cleared cache block but does not find a matching address, in the coherence address list, is still a fill.

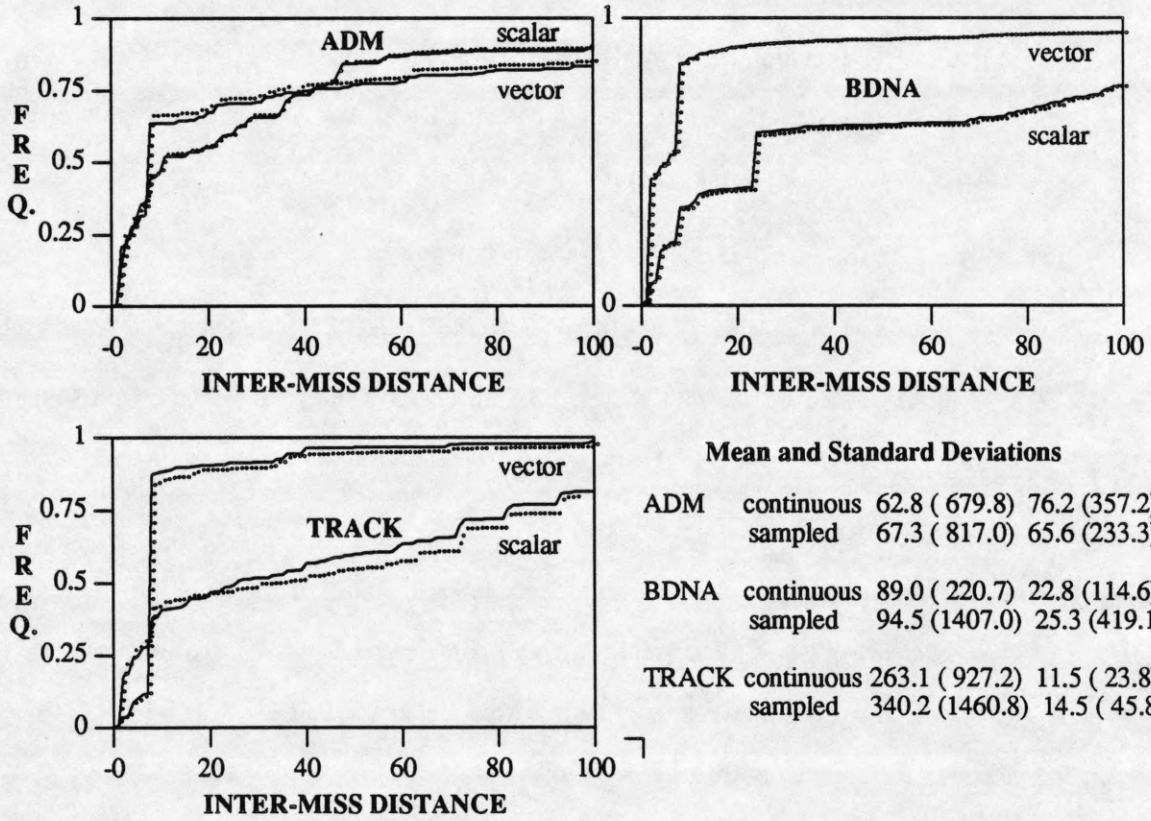

Figure 6: Split scalar-vector cache inter-miss distributions. Each scalar and vector cache is 128 Kbytes, 32 byte block and a set size of 2. The relative cumulative frequency distribution is show with the corresponding mean and standard deviation in ()s.

4.5. Multicache Prediction Results

In multicache simulation with a sampled trace, each cache maintains a separate inter-miss history to predict the fills generated by that cache. Figure 6 shows the inter-miss distribution plots for the uniprocessor traces with a split scalar-vector cache and the corresponding mean and standard deviations.

The prediction tends to be more accurate for vector than for scalar references. This is because vector references are generally used to access data held in matrices and contained within a loop. Thus, the inter-miss distances have a more regular pattern making them easier to predict. As with the distribution plots for the unified cache, the largest deviation between the continuous and sampled traces occur when there is a sharp transient in the distribution.

The multiprocessor trace results are shown in Table 6. Each cache is 128 Kbytes with a set size of 2 and a block size of 32 bytes. The prediction is accurate across all the caches except for cache 7 for ADM's trace which shows a substantial difference between continuous and sampled traces.

5. Conclusions

Trace sampling is beginning to attract interest as it offers an inexpensive and practical approach to reduce trace driven simulation costs. This paper has shown that taking a number of small samples can

TRACE	inter-miss distribution for each processor/cache							
	0	1	2	3	4	5	6	7
ADM	14.9 (34.8)	7.6 (19.7)	7.8 (19.8)	7.6 (20.5)	5.7 (10.6)	5.9 (14.0)	5.8 (10.9)	15.3 (44.2)
	15.5 (36.0)	7.7 (19.8)	7.9 (20.2)	7.6 (20.6)	5.7 (10.2)	5.6 (10.9)	5.8 (10.8)	20.6 (70.6)
BDNA	6.6 (47.4)	6.2 (23.1)	5.7 (18.6)	6.0 (19.1)	5.8 (20.3)	6.0 (19.5)	7.8 (53.1)	44.7 (3402.5)
	6.3 (20.1)	6.0 (16.9)	5.6 (16.0)	5.8 (16.4)	5.6 (16.5)	5.9 (17.3)	8.1 (485.9)	46.7 (2845.0)
TRACK	8.2 (110.9)	7.4 (131.4)	4.7 (56.0)	55.9 (474.3)	4.4 (48.3)	4.6 (51.8)	4.7 (45.8)	12.9 (64.6)
	10.3 (184.0)	8.4 (621.0)	5.2 (80.9)	687 (611.0)	4.0 (17.0)	4.4 (23.4)	4.9 (37.2)	16.4 (186.7)

Table 6: Multiprocessor Mean and Standard Deviations. For each trace, the continuous trace results is the first line and the predicted results using the sampled trace is the following line. The mean and standard deviation is shown for each of the 8 processors. The mean is the first number followed by the standard deviation enclosed with ().

capture the characteristics of a large continuous trace. Simulating cache models require fill references to be resolved as hits or misses. We have presented a simple method to predict fill references based on the miss history of the reference stream and the contents of the cache. Unlike previous models that only calculate the miss ratio this approach allows detailed simulation of individual miss events. We have shown results for both single cache and multiple cache simulations and obtained accurate results for cache sizes up to 256KB. For larger caches we believe that the sample size needs to be increased for consistency in the sampling result. The prediction success rate was lower with a small cache and more information is needed to improve the prediction.

Our method has been applied to continuous traces of 50-100 million references from numerical programs. We have not experimented widely with increasing the sample size with these trace since the sampled trace can quickly approach the size of the continuous trace. Indications are that taking larger samples does improve accuracy and consistency particular as the cache size increases. We are in the process of obtaining general purpose program traces with billions of references. These will be used to experiment with larger sample and cache sizes and to verify our our prediction method.

REFERENCES

[Alli85] Alliant Computer Systems Corporation, FX/Series Product Summary, June 1985.

[BoKW90] A. Borg, R. E. Kessler, D. W. Wall, "Generation and Analysis of Very Long Address Traces", *Proc. 17th. Int'l. Symp. on Comp. Arch.*, pp. 270-279, May 1990.

[Berr89] M. Berry, et. al., "The Perfect Club Benchmarks: Effective Performance Evaluation of Supercomputers," *Int'l. Journal for Supercomputer Applications*, Fall 1989.

[ClBK88] D. W. Clark, P. J. Bannon and J. B. Keller, "Measuring VAX 8800 with a Histogram Hardware Monitor," *Proc. of 15th. Int'l Symp. on Comp. Arch.*, pp. 176-185, June 1988.

[CoWh90] T. M. Conte and W. W. Hwu, "SIngle Pass Memory System Evaluation for Multiprogramming Workloads," Tech. Rpt. CSG-122, Center for Reliable and High Performance Computing, University of Illinois, May 1990.

[DiIy91] R. T. Dimpsey and R. K. Iyer, "Performance Prediction and Tuning on a Multiprocessor," *Proc. of 18th. Int'l Symp. on Comp. Arch.*, pp. 190-199. May, 1991.

[EKKL90] S. J. Eggers, D. R. Keppel, E. J. Koldinger and H. M. Levy, "Techniques for Inline Tracing on a Shared Memory Multiprocessor", *Proc. ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems*, pp. 37-47, May 1990.

[FuPa91] J. W. C. Fu and J. H. Patel, "Prefetching in Multiprocessor Vector Cache Memories", *Proc. of 18th. Int'l Symp. on Comp. Arch.*, pp. 54-63.

[HiSm89] M. D. Hill and A. J. Smith, "Evaluating Associativity in CPU Caches," *IEEE Trans. on Comp.*, vol. 38, no. 12, pp. 1612-1630, Dec. 1989.

[LaPI88] S. Laha, J. H. Patel and R. K. Iyer, "Accurate Low-cost Method for Performance Evaluation of Cache Memory Systems," *IEEE Trans. Comp.*, vol C-36, pp. 1063-1075, 1987.

[LeSm84] J. K. F. Lee and A. J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," *Computer*, Vol. 17, No. 1, pp. 6-22, Jan. 1984.

[LiPe90] L. Liu and J. K. Peir, "Sampling of Cache Congruence Class," IBM Tech. Rpt 1990.

[Pate90] J. H. Patel, "How to Simulate 100 Billion References Cheaply," *nt'l Symp. on Comp. Arch.i, workshop on tracing*, May 1990, pp. 190-199.

[Samp89] A. D. Sample, "Mache: No-loss Trace Compaction," *Proc. ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems*, pp. 89-97, May 1989.

[Smit77] A. J. Smith, "Two Methods for the Efficient Analysis of Memory Address Trace Data," *IEEE Trans. on Soft. Eng.*, no. 3, pp. 94-101, Jan. 1977.

[Smit85] A. J. Smith, "Cache Evaluation and the Impact of Workload Choice," *Proc. 12th. Int'l. Symp. on Comp. Arch.*, June 1985.

[SoZe88] K. So and V. Zecca, "Cache Performance of Vector Processors", *Proc. 15th. Int'l. Symp. on Comp. Arch.*, pp. 261-268, June 1988.

[StFu89] C. B. Stunkel and W. K. Fuchs, "TRAPEDS: Producing traces for multicomputers via execution driven simulation", *Proc. ACM Sigmetrics Conf. on Measurement and Modeling of Compter Systems*,pp. 70-78, May 1989.

[Ston88] H. S. Stone, High Performance Computer Architecture, Addison-Wesley Publishing Company, 1987.

[WaBa90] W. Wang and J. Baer, "Efficient Trace-Driven Simulation Methods for Cache Performance Analysis," *Proc. ACM Sigmetrics Conf. on Measurement and Modeling of Compter Systems*, pp. 27-36, May 1990.