

# FACILITATING LEVEL THREE CACHE STUDIES USING SET SAMPLING

Niki C. Thornock and J. Kelly Flanagan

Computer Science Department  
Brigham Young University  
Provo, UT 84602, U.S.A.

## ABSTRACT

We discuss some of the difficulties present in trace collection and trace-driven cache simulation. We then describe our multiprocessor tracing technique and verify that it accurately collects long traces. We propose sampling as a method to reduce required disk space, enable simulations to run faster, and effectively enlarge the trace buffer of our hardware monitor, decreasing trace distortion. To this end, we investigate time sampling and two types of set sampling. We conclude that the second set sampling technique achieves the most accurate results. The miss rate for the second set sampling method is calculated as the number of misses to sampled sets divided by the total number of references scaled by the sample size. We determined that a 10% sample size was the most accurate while still reducing required disk space.

## 1 INTRODUCTION

Processor speeds are increasing at a much greater rate than memory speeds. This difference causes a bottleneck in the system and decreases performance. With the increasing popularity of multiprocessor computers, the bottleneck is becoming even worse. Researchers are searching for ways to reduce this problem. Since it is now common to have two caches (level 1 and level 2) integrated onto each processor chip, adding a third, very large, off-chip cache (level 3 or L3) seems a likely candidate for reducing the bottleneck. Simulation, especially trace-driven simulation, is a frequently used method of testing new cache configurations. Creating a simulator is a fairly straightforward, albeit very time consuming, task. The difficulty lies in obtaining the long, accurate traces necessary for simulating extremely large L3 cache systems used in current and future multiprocessor systems.

## 1.1 Background And Previous Work

For our purposes, a trace is a stream of successive address requests containing references for memory reads, writes, and instruction fetches. There are several different address trace collection methods; for example, instruction modification (Borg 1990, Sun 1992, Larus 1992), microcode modification (Agarwal 1986), single stepping (Agarwal 1986), processor simulation (Sohi 1991), and hardware monitors (Clark 1983, Torrellas 1992, Nagle 1993). These methods often introduce one or more of the following four types of errors into collected trace data, (1) missing operating system references, (2) absent multitasking behavior, (3) time dilation, or (4) short traces (Harper 1993). These introduced errors mean that simulation runs are not completely accurate. A hardware monitor which overcomes many of these problems is described in (Flanagan 1993, Crockett 1994), but this technique requires frequently halting the system under test (SUT). Thus, although this type of hardware monitor eliminates the four problems mentioned above, it introduces the potential for new errors due to halting the SUT.

One major disadvantage associated with trace-driven simulation is the storage requirements for long traces. Another disadvantage is the long run times of simulations. Sampling is a technique that partially overcomes these disadvantages. This is a statistical method where a selected fraction of a population is used to represent the whole population. This method has been demonstrated to work effectively with first- and second-level caches in single processor systems (Kessler 1994, Martonosi 1995). Sampling has three potential advantages. It reduces disk space needed to store traces, enables simulations to run faster, and effectively enlarges trace buffers of hardware monitors. We will investigate whether the sampling techniques described in (Kessler 1994) perform acceptably when simulating L3 caches in symmetric multiprocessor (SMP) systems.

This paper proceeds in the following order. Section 2

describes and evaluates our technique for tracing multiprocessor systems. Section 3 gives a general description of sampling and describes our implementations. It also presents the results of our sampling study. Finally, Section 4 presents our conclusions and lists possible future work.

## 2 MULTIPROCESSOR TRACING

Trace-driven simulation is very accurate if both the model and input data represent the real system under test. The accuracy of the model is typically under the control of the researcher. Unfortunately, many researchers do not have the facilities to collect accurate trace data and good data is difficult to acquire from other sources (Smith 1982, Przybylski 1989, Borg 1990, Flanagan 1993). In this paper, we use a hardware monitor designed to collect address traces from a Hewlett-Packard SMP system containing four Intel Pentium Pro processors.

### 2.1 Trace Collection Technique

Our trace collection technique uses a Tektronix TLA520 logic analyzer to collect trace data. We also require two parallel I/O cards for the system under test (SUT) and a workstation which is used to process the trace data. These items are used to implement a system similar to the BACH system described in (Flanagan 1993, Crockett 1994). Our hardware monitor is identical, but the required software is substantially modified to enable its use in SMP systems.

#### 2.1.1 Logic Analyzer Configuration

The logic analyzer is connected to a probe which fits between one of the processors and its socket and forwards signals from the SUT to the Tektronix logic analyzer (TLA520 or TLA). The probe monitors the address, data, and control lines from all four processors via the shared bus. It collects signals after the L2 cache since both L1 and L2 caches are integrated on the Pentium Pro chip. This does not pose a problem for this work since we are interested in determining if sampling is useful for evaluating L3 caches.

Figure 1 shows a typical setup that consists of the machine being traced (SUT), TLA520, and a workstation for collecting the acquired data. This tracing setup is identical to the BACH system, but the SUT device driver that handles the MSO signal is substantially different. When enabled, the TLA520 monitors the pins of the SUT, storing timing, address, and control information in an internal buffer; the buffer is 512 K entries

long. When the buffer fills, the workstation downloads the trace for storage or processing.

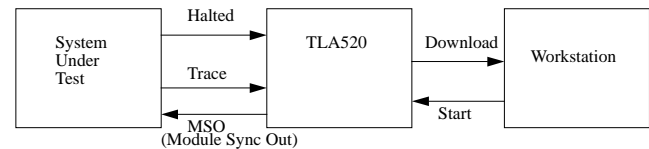


Figure 1: Communications Between the SUT, TLA520 and a Workstation

#### 2.1.2 Operation and Organization

We use a technique that overcomes the trace length limitation of previous hardware monitors. When the TLA520's internal buffer is almost full, it sends a signal through Module Sync Out (MSO), a line connected to the interrupt line of a parallel I/O card located in the Eisa slot of the SUT. This is a low priority interrupt which allows the SUT to finish other higher priority interrupts before responding. The buffer does not overflow under normal operating conditions. After all other devices have been serviced, the SUT enters an interrupt routine, disables interrupts, and spins in a tight loop. While it spins, it sends a continuous signal back to the TLA indicating that it has halted. When the TLA receives this signal, it raises MSO and stops collecting traces. The workstation has been waiting for the TLA to halt and it promptly downloads the buffer over the network and restarts the TLA. The buffer contents may be stored to secondary storage media or processed while being extracted. The TLA lowers MSO to signal the SUT that it's ready to collect another buffer. The SUT stops asserting its signal and resumes processing. The TLA raises MSO and proceeds to collect another buffer. This process may be repeated as many times as desired, producing a contiguous trace. The difference between the single processor technique used in (Flanagan 1993) and the multiprocessor technique used here is that it is necessary to wait until all four processors have entered the interrupt loop before continuing.

#### 2.1.3 System Under Test

The system under test is a Hewlett-Packard NetServer LX Pro symmetric multiprocessor with four Intel Pentium Pro processors. It has 16 gigabytes of disk and 1 gigabyte of memory. Each processor has on-chip L1 and L2 caches. The L1 caches are 8 Kbyte, four-way set-associative instruction and data caches; the L2 caches are 512 Kbytes and 4-way set-associative. The multiprocessor is running Microsoft Windows NT Server 4.0

with Service Pack 3. It also has two parallel I/O cards in its Eisa slots as part of the trace collection mechanism.

#### 2.1.4 Trace Download And Storage

We used a workstation running HP-UX for this stage. The workstation is responsible for storing or processing the data. Once the TLA520 has collected a buffer of data and halted, the data is downloaded to the workstation. While the TLA520 collects another buffer, the workstation strips a header off of the data, compresses and stores it.

## 2.2 Verification And Evaluation

After implementing our technique, it was necessary to verify that halting was possible in an SMP system and to measure the trace perturbation caused by the halting.

### 2.2.1 Verification

Once we had developed our halting mechanism it was important to verify that it actually worked. We wrote a program that spun in a loop, incrementing a counter and writing the value to an I/O port. When the counter reached 0xFFFF, it rolled over to zero. The upper 16 bits of the counter were fixed, so we could identify it in the trace. We chose an I/O port because it's non-cacheable so all values would appear in the trace. We configured the logic analyzer to collect a thousand buffers of the counter program (256 million references). The logic analyzer would obviously collect consecutive counter values within a buffer. If the halting mechanism did *not* work correctly, we would find non-consecutive counter values between consecutive buffers. We analyzed the collected trace and found that the counter values between buffers were consecutive in all cases. We collected another thousand buffers and achieved identical results. We concluded that the halting mechanism worked correctly.

As a further illustration, we contrast the working halting mechanism with the earlier, incomplete halting mechanism. As part of the debugging process, we opened several programs in various windows so that we could tell if a processor was halted. One window contained a clock program. When we tried to halt the system, a window would freeze or the mouse would freeze or occasionally the entire screen would freeze, indicating that at least one processor had halted. Once we released the halting mechanism, the clock would jump ahead to the correct time. This is significant because it showed that the clock interrupt was being processed even if it

wasn't being updated on the screen; this meant that at least one processor had not been halted. We could tell that all four processors were halted when we could release the halting mechanism and have the clock continue running as if nothing had happened. We mention the clock interrupt in particular because it has one of the highest interrupt levels while our halting mechanism has a fairly low priority interrupt; if it were possible for our halting mechanism to be interrupted, the clock interrupt would do so.

### 2.2.2 Limitations

Although we can effectively halt the processors, we can't halt the peripheral devices. Since the processors are halted for approximately 30 seconds for downloading, the I/O devices have plenty of time to complete any pending requests. As a result, at the beginning of each buffer, there are a number of interrupts waiting to be processed. As noted in an earlier version of this technique (Flanagan 1993), this distortion is negligible. Since the distortion occurs at the beginning of each buffer, a deeper buffer would clearly be beneficial. The halting also makes it difficult to trace real-time applications such as interactive games.

## 2.3 Summary

We have designed and implemented a trace collection mechanism that collects long, contiguous, and accurate traces with negligible amounts of distortion. Trace distortions could be reduced further if a larger trace buffer were available. The following section describes sampling, a technique that effectively enlarges the buffer size for trace-driven cache simulations.

## 3 SAMPLING

In this section we will discuss time sampling and two types of set sampling. We will define a metric for accuracy and then compare the results of the sampling techniques for various workloads and determine which method produces the most accurate results.

### 3.1 Sampling Methods

Sampling is a statistical process of using a subset (or sample) of a population to represent the whole. It is possible to directly compute characteristics of the population like the mean or standard deviation. When we take a sample of a population we can calculate the sample mean and use it as an estimate of the real mean. We would like to verify that sampling works so that we can integrate a sampling method with the trace collection

mechanism or use it in cases where complete trace collection is not possible.

Since trace perturbations occur at the beginning of each buffer, we would like to make the buffers very large. This is not always physically possible so we would like to fit as much information as possible into a single buffer. We will accomplish this by *sampling* the trace. Sampling has three potential advantages. It reduces the needed disk storage space, enables simulations to run faster, and effectively enlarges the trace buffer.

When we collect a complete trace, we are able to directly calculate the cache miss rate by running the entire trace through a cache simulator. Two possible sampling methods would be to take pieces of the trace and run them through a cache simulator or to only look at certain sets in the cache during simulation. If we take pieces of the trace, it eliminates the need to halt the SUT. A buffer could be collected as usual, but instead of halting the SUT as the buffer is saved to disk, the SUT would continue to run. After the buffer is saved, another can be collected, and so forth. If, on the other hand, we choose to sample cache sets, we can do this by only acquiring references to the selected sets. This can be accomplished by using a bit mask on the cache set bits. This will take less storage space than a complete trace and effectively enlarge the trace buffer.

*Time sampling* refers to the option of taking pieces of the trace and feeding them through a simulator. If we want to use 10% of the trace, we will send every tenth buffer through the simulator and calculate the miss rate. Time sampling was successfully used in (Fu 1994, Martonosi 1995). Time sampling eliminates the need to halt the SUT and therefore decreases trace distortions.

*Set sampling* refers to selecting certain sets in the cache. Upon entering the cache, an address is broken into three parts, tag, set selector, and line offset. The set selector bits choose which set the address maps to. We will use a bit mask on the set selector to filter the trace; if the selected bits match the given pattern, that address will be sent through the cache simulator. We placed the bit mask on the low-order bits of the set selector bits to get the maximum distribution of addresses. The pattern was chosen arbitrarily. If either of the set sampling techniques described below prove to be useful, we can integrate the bit mask into our trace collection mechanism, allowing us to collect only the references that map to the desired sets. This effectively increases the size of the trace buffer and will allow the CPU to run longer before it is halted. Set sampling was used successfully in (Kessler 1994).

The problem of choosing which pieces of the trace to take or which sets to focus on is non-trivial. Another problem is the sample size: deciding how much of the

trace or how many sets to use. If we only use a small fraction of the trace we won't get very accurate results, but if we use a very large fraction of the trace, the size improvement isn't worth the effort it takes to do the sampling. As a basic heuristic, we decided that the sample miss rate should be within 10% of the real miss rate using at most 10% of the trace. We will first look at several methods of choosing which trace pieces or cache sets to focus on.

We will investigate two set sampling techniques which are useful when conducting cache studies. In the first technique (set1), we filter the trace using a bit mask as described above and keep track of misses and references to the selected sets. In this case, the estimated miss rate is calculated as the number of misses to the selected sets divided by the number of references to the selected sets.

$$\frac{\textit{SampledMisses}}{\textit{SampledReferences}} \quad (1)$$

In the second technique (set2), we record the number of misses to the selected sets. Unlike the first technique, we count references to all sets in the cache and then scale that number by the ratio of sampled sets to total sets. The scaled number is then used to calculate the estimated miss rate.

$$\frac{\textit{SampledMisses}}{\textit{ScaledReferences}} \quad (2)$$

The first technique could be implemented by configuring the TLA to only collect references that map to the selected sets. The references could be counted during simulation. The second technique would use the same configuration with the addition of a running counter to keep track of all cacheable references. Both sampling techniques were proposed in (Kessler 1994); our contribution is demonstrating that sampling is a viable technique for simulating huge L3 caches in an SMP environment.

The following examples demonstrate how each set sampling method works. We will use a small 128 byte, 16 byte line, direct-mapped cache for this illustration. This means that a 16-bit address will consist of 4 offset bits, 3 set bits, and 9 tag bits. The offset bits are ignored for simulation purposes. There are 8 sets in the cache. Table 3.1 lists an address trace and the set that each address maps to.

Table 2 shows the cache after simulation with the trace. It lists the addresses for the misses to each set and the number of hits to each set. As an example, we will use the first set sampling technique with a 50% sampling rate. We will arbitrarily select the odd numbered sets for our sample. In the trace, this would include all addresses where the lowest set bit is a one. We can see

Table 1: Mapping of Addresses in a Trace to Sets in a Cache

Address	Maps to Set	Address	Maps to Set
0x7d91	1	0x9f2a	2
0x7d94	1	0x9f28	2
0x7e6b	6	0x9f24	2
0x833b	3	0x9f26	2
0xcc5c	5	0x9f22	2
0x833a	3	0x9f1e	1
0x9339	3	0x9f1c	1
0x6604	0	0x9f1a	1
0x1604	0	0xa458	5
0x9f27	2	0x7540	4
0x9f21	2	0x5a68	6

Table 2: Hits and Misses to a 128 Byte, 16 Byte Line, Direct-Mapped Cache After Simulating the Trace in Table 1

Set#	Misses	# of Hits
0	660, 160	0
1	7d9, 9f1	3
2	9f2	6
3	833, 933	1
4	754	0
5	cc5, a45	0
6	7e6, 5a6	0
7		0

from Table 2 that there are 6 misses to odd sets; adding in the 4 hits to sets 1 and 3 gives us 10 references. This sample has a miss rate of 60%. The real miss rate is 12 / 22 which is 54.5%.

As a second example, we will demonstrate the second set sampling technique with a 50% sampling rate. We will again select the odd numbered sets. Again, there are 6 misses to odd sets. To calculate the references, we note that there are 22 addresses listed in the trace in Table 3.1. Since this is a 50% sampling rate, we will multiply 22 by 0.5 for a result of 11. The miss rate for this method is 6 / 11 which is 54.5%.

For both of the sampling techniques, we investigated three sample sizes: 6% (1/16), 10% (1/10), and 25% (1/4). (We note that the 10% size is approximate for set sampling since the  $2^n$  sets will not divide evenly by 10.) The 10% sample size exactly meets the size criteria. With the 6% size we investigate whether it's possible to use a smaller fraction of the trace. The 25% sample size will allow us to investigate a larger fraction if we can't achieve the needed accuracy with the smaller

sample size.

### 3.2 Workloads

We collected traces from several different workloads; the traces were collected after the L2 cache. The first workload consisted of 4 instances of an MP3 compressor running simultaneously. Each instance compressed one of four distinct 500 megabyte sets of audio files. The trace contained about 350 million references. The second was a threaded version of NONA which is a genetic program that looks at two sets of DNA to see if they're related. This trace contained 1.2 billion references. The third workload is Winbench, an address trace of the Ziff Davis Winbench benchmark running the All Winmarks suite, except for the disk benchmarks. It had 700 million references. The fourth is Winstone, an address trace of the Ziff-Davis Winstone 99 benchmark running the dual-processor inspection tests (MicroStation SE, Photoshop 4.0, Visual C++ 5.0) with 300 million references. The fifth is an address trace of a demo playback of the Descent video game with 200 million references. The sixth workload is an address trace of the Transaction Processing Council Benchmark C (TPC-C) using an Informix database. This benchmark is frequently used in the computer industry. We used a 5 warehouse implementation which exercised the cache thoroughly. The trace contains 1 billion references. Table 3 summarizes this information.

Table 3: Names and Number of References of Traces Used in Our Sampling Study

Workload	References
MP3	350 million
Para-NONA	1.2 billion
Winbench	700 million
Winstone	300 million
Descent	200 million
TPC-C	1 billion

### 3.3 Preliminary Results

As we mentioned previously, we decided that a sampling method must be within 10% of the actual miss rate in order to be acceptable. Error rates for the miss rates were calculated using the following formula:

$$\frac{(real - sample)}{real} \quad (3)$$

A more accurate result has a lower error rate. We conducted a preliminary study using the MP3 and para-

NONA workloads in hopes of eliminating some simulation runs. We used sampling ratios of 6%, 10%, and 25%. The 25% sample size allowed us to investigate a larger fraction if acceptable results were not achieved using the smaller fractions. We simulated caches with sizes of 8, 16, and 32 megabytes, line sizes of 32 and 64 bytes, and associativities of 1, 2, 4, and 8. We repeated the studies with different samples in order to calculate confidence intervals. We used a 95% confidence level.

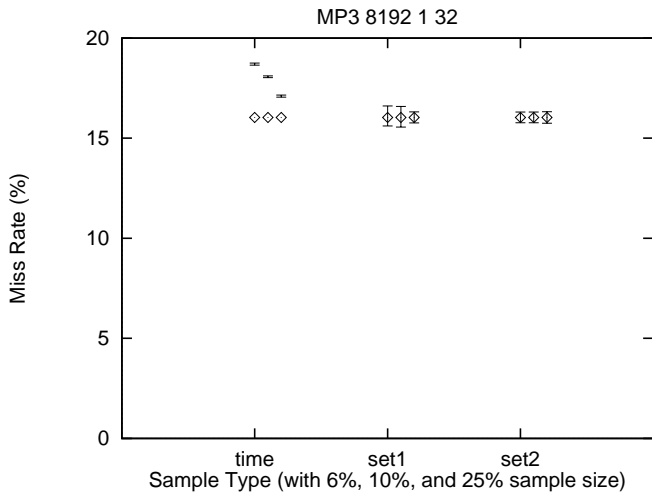


Figure 2: Results of Sampling Techniques for an 8 Megabyte, 32 Byte Line, Direct-Mapped Cache using the MP3 Compressor Workload

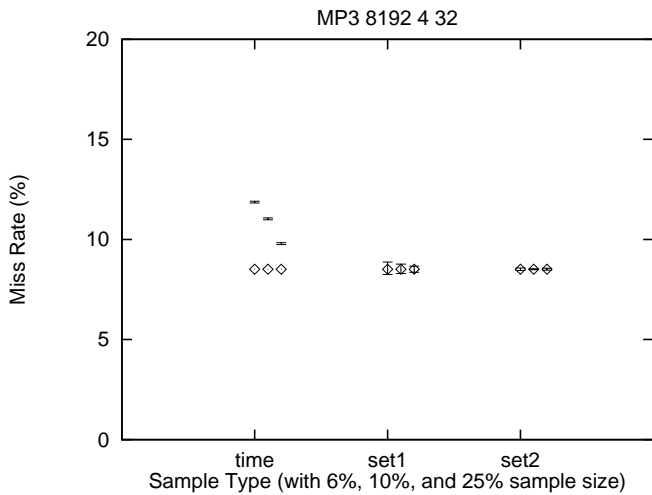


Figure 3: Results of All Sampling Techniques for an 8 Megabyte, 32 Byte Line, 4-Way Set-Associative Cache for the MP3 Compressor Workload

Figures 2 and 3 show the results for the MP3 Compressor workload. The diamonds represent the actual miss rate. The confidence intervals represent the variation of the sample miss rates. This means that the time sampling miss rates were very consistent but they were

far away from the real value. The set sampling miss rates were not as consistent (narrow) but they were much closer to the real miss rate. Figures 4 and 5 show the results for the para-NONA workload. These graphs are for an 8 megabyte, 32 byte line cache. Again, the real miss rate is consistently outside of the time sampling confidence interval. The other cache configurations had similar results and are not shown.

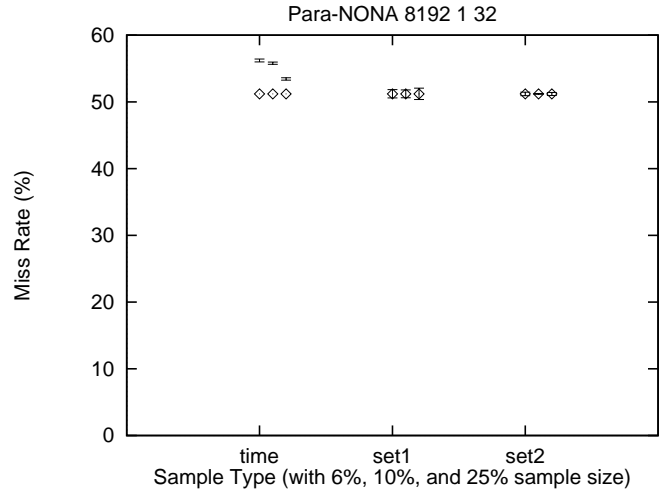


Figure 4: Results of Sampling Techniques for an 8 Megabyte, 32 Byte Line, Direct-Mapped Cache Using the Para-NONA Workload

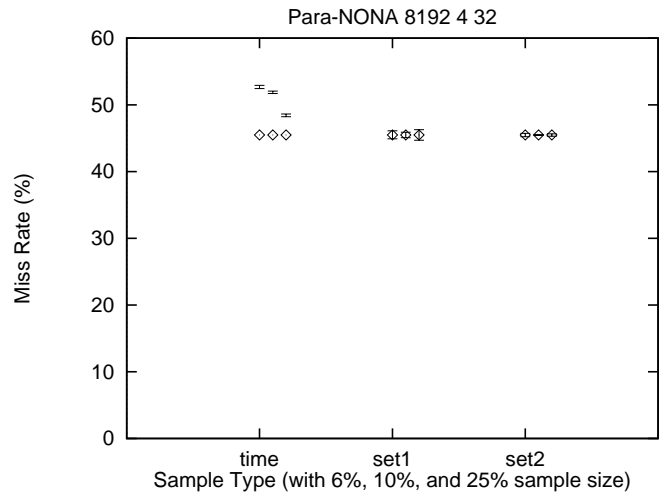


Figure 5: Results of Sampling Techniques for an 8 Megabyte, 32 Byte Line, 4-Way Set-Associative Cache for the Para-NONA Workload

### 3.4 Time Sampling

After analyzing the results of the preliminary study, we determined that the time sampling method was unacceptable. Although it has a very low variation, the confidence interval does not include the real miss rate.

It has an error rate of 50 to 90 percent for the first workload and over 10 percent for the second.

### 3.5 Set Sampling

The preliminary study showed that the first set sampling method performed acceptably for both the MP3 compression workload and the para-NONA workload. Confidence intervals consistently centered around the real miss rate and were within 10% error for the 10% and 25% sample sizes. They were within 15% for the 6% sample size.

The second sampling method behaved like the first method for both workloads: confidence intervals centered around the real miss rate and all error rates were less than 10%.

Each set sampling method performed acceptably for both workloads. Since neither of the set sampling methods was conclusively better than the other, further investigation was necessary.

### 3.6 Further Set Sampling Study

Following are the results for the other four workloads: Descent, Winbench, Winstone, and TPC-C using the two set sampling methods. Figures 6 through 9 are for a direct-mapped cache configuration. The set-associative configurations had similar graphs with smaller miss and error rates and are not shown. The direct-mapped configurations were invariably worse than the set-associative configurations since they are much more dependent on the stream of addresses.

The first set sampling method had centered confidence intervals for all workloads except for TPC-C. All error rates for the Winbench and Winstone workloads were under 10%. The Descent workload had error rates ranging from  $\pm 9\%$  for the 6% and 25% sample sizes to  $-18\%$  to  $+27\%$  for the 10% size. The TPC-C workload had error rates ranging from  $-83\%$  to  $+126\%$ . Error rates for the set-associative configurations were similar for Winbench, Winstone, and TPC-C. Descent error rates for 6% and 25% sample sizes were well under 10% and ranged from  $-8\%$  to  $+12\%$  for the 10% size.

The second set sampling method had centered confidence intervals in all cases. Its error rates were less than 10% for all sample sizes for the TPC-C, Winbench, and Winstone workloads. The Descent workload did worse than the other workloads with error rates ranging from  $\pm 10\%$  to  $\pm 14\%$ . Error rates for the set-associative configurations were well under 10% for all four workloads. We conclude that the second set sampling method performs better than time sampling and the first set sampling method.

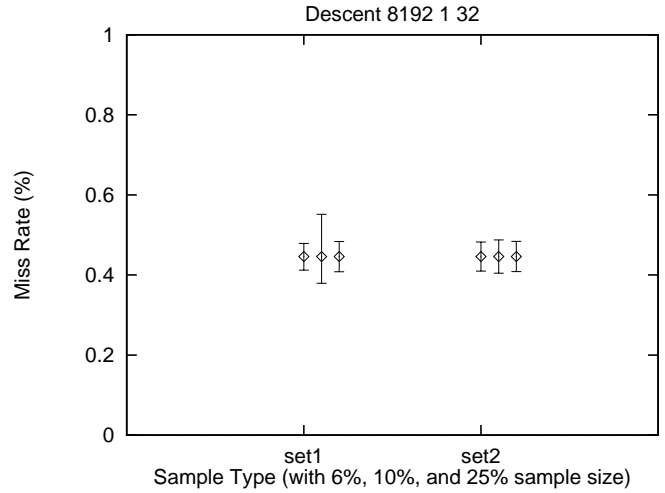


Figure 6: Results of Sampling Techniques for an 8 Megabyte, 32 Byte Line, Direct-Mapped Cache Using the Descent Workload

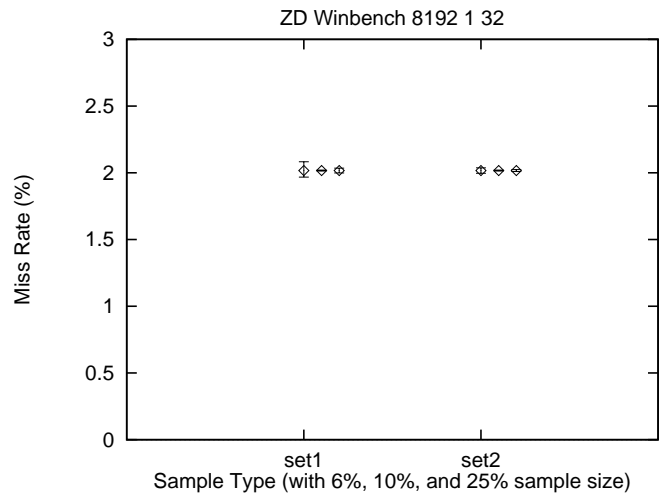


Figure 7: Results of Sampling Techniques for an 8 Megabyte, 32 Byte Line, Direct-Mapped Cache Using the Ziff-Davis Winbench Workload

The results for the set sampling techniques can be explained by studying the following graph. Figure 10 displays the number of references to each set in a direct-mapped cache for the MP3 workload. We observe that the MP3 graph has some tall spikes. In the first sampling technique, we use the number of references to certain sets. The chances of selecting a set with a spike are about 1 in 10 for a 10% sampling size. This means that about 90% of the time, those references will not be included. In the second sampling technique, we count *all* of the references and then scale them by the sample size; this has the effect of distributing the references equally across all cache sets. This is why the second technique has a narrower confidence interval for MP3, TPC-C, Winbench, and Winstone. Graphs for other configurations have similar spikes and are not shown.

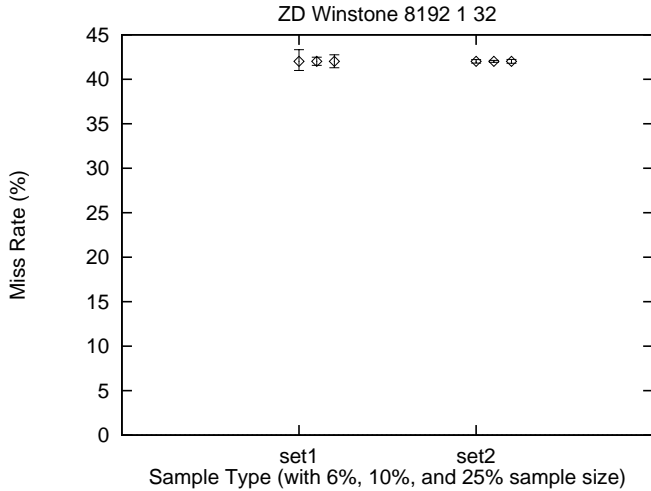


Figure 8: Results of Sampling Techniques for an 8 Megabyte, 32 Byte Line, Direct-Mapped Cache Using the Ziff-Davis Winstone Workload

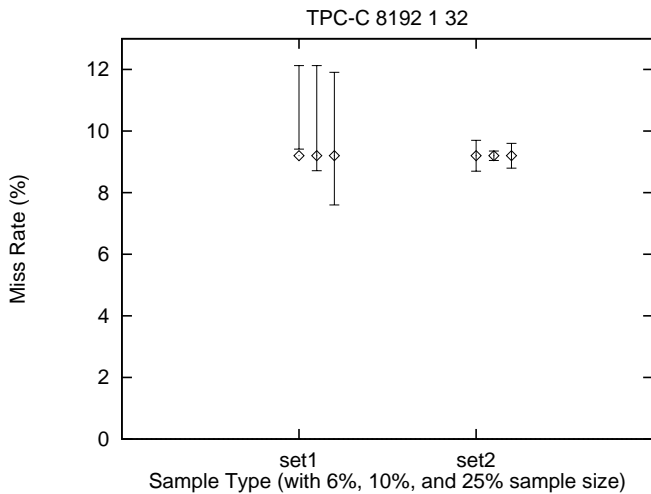


Figure 9: Results of Sampling Techniques for an 8 Megabyte, 32 Byte Line, Direct-Mapped Cache Using the TPC-C Workload

### 3.7 Sample Size

Once we had determined that the second sampling method worked best, we needed to decide which sample size had an error rate of less than 10% while using less than 10% of the trace. We investigated 6%, 10%, and 25% sample sizes. We chose the 10% sample size for further investigation because its error rate was acceptable (less than 10% error), and it had much greater trace compression than the 25% sample size. We find it unusual that the 10% sample size performs better than the 25% sample size in most cases. Figures 11 and 12 show the error rate for each sample size for an 8 megabyte cache with a 32 byte line size, direct-mapped and 4-way set-associative, respectively.

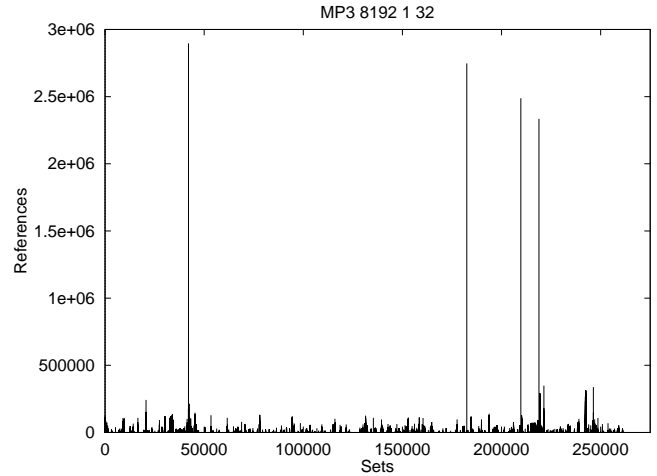


Figure 10: References to All Sets in an 8 Megabyte, 32 Byte Line, Direct-Mapped Cache with the MP3 Compressor Workload

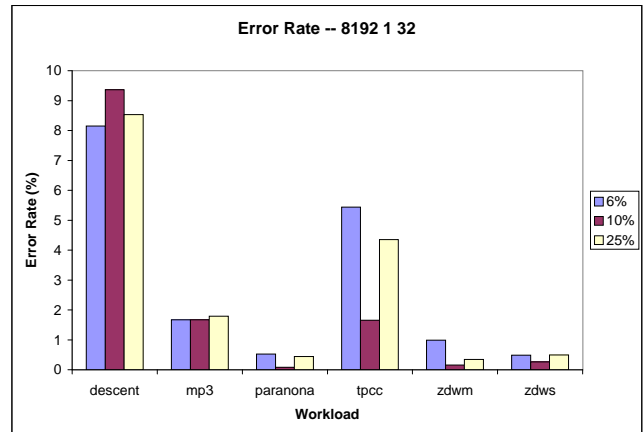


Figure 11: Sample Size Comparison for an 8 Megabyte, 32 Byte Line, Direct-Mapped Cache

### 3.8 Summary

We investigated time sampling and two versions of set sampling. The time sampling and the first set sampling method produced unacceptable results. The second set sampling method appears to produce accurate results. We determined that the 10% sample size had the lowest error rate while still using less than 10% of the trace.

## 4 CONCLUSIONS AND FUTURE WORK

We discussed some of the difficulties of trace collection and trace-driven cache simulation. We then described our multiprocessor tracing technique and verified that it accurately collects long traces. Three types of sampling were described: time sampling and two types of set sampling. With time sampling, pieces of the trace

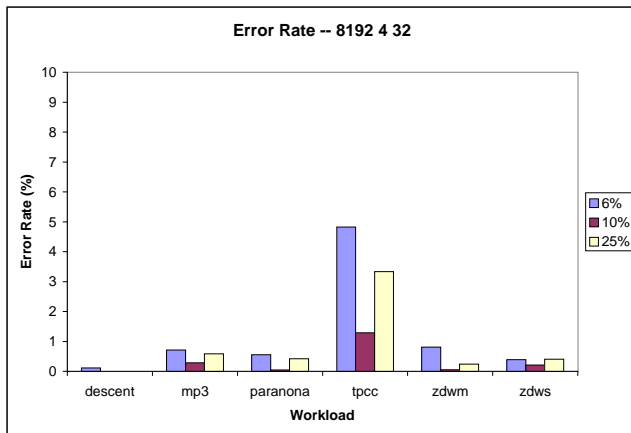


Figure 12: Sample Size Comparison for an 8 Megabyte, 32 Byte Line, 4-way Set-Associative Cache

represented the entire trace. In the set sampling techniques, a subset of the sets represented all sets in the cache. The difference between the two set sampling techniques is the calculation of the miss rate. In the first technique, the number of misses to sampled sets is divided by the number of references to sampled sets. In the second technique, the miss rate is calculated as the number of misses to sampled sets divided by the total number of references scaled by the sample size. We concluded that the second set sampling technique achieved the most accurate results for all caches. This can be integrated into our tracing hardware to effectually increase the length of the buffer, reducing trace distortions. We determined that the 10% sample size had the lowest error rate while still using less than 10% of the trace.

#### 4.1 Future Work

Future work includes adapting the hardware monitor to use sampling techniques and performing L3 cache studies using set samples of commercial workloads.

#### ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. 9807619.

#### REFERENCES

Agarwal, A., R. L. Sites and M. Horowitz. 1986. ATUM: A new technique for capturing address traces using microcode, *Proceedings of 13th International Symposium on Computer Architecture*, 119–127.

Borg, A., R. E. Kessler and D. W. Wall. 1990. Generation and analysis of very long address traces, *Proceedings of 17th International Symposium on Computer Architecture*, 270–279.

Clark, Douglas W. 1983. Cache performance in the VAX-11/780, *ACM Transactions on Computer Systems*. 1(1):24–37.

Crockett, Niki and J. Kelly Flanagan. 1994. i486 address trace collection using a TLA510, *Technical report*, Department of Computer Science, Brigham Young University, Provo, UT 84602.

Flanagan, J. Kelly. 1993. *A New Methodology for Accurate Trace Collection and its Application to Memory Hierarchy Performance Modeling*, PhD thesis, Brigham Young University.

Fu, J. W. C. and J. H. Patel. 1994. Trace driven simulation using sampled traces, In *27th Hawaii International Conference on System Sciences*, 211–220.

Harper, D. T. III and Z. Zhang. 1993. Cache miss ratio analysis using sampled traces, Submitted to *ACM Transactions on Modeling and Computer Simulation*.

Kessler, R. E., M. D. Hill, and D. A. Wood. 1994. A comparison of trace-sampling techniques for multi-megabyte caches, *IEEE Transactions on Computers*. 43(6):664–675.

Larus, J. R. and T. Ball. 1992. Rewriting executable files to measure program behavior, Technical Report 1083, Department of Computer Science, University of Wisconsin.

Martonosi, M., A. Gupta and T. E. Anderson. 1995. Tuning memory performance of sequential and parallel programs, *IEEE Computer*. 28(4):32–40.

Nagle, D., R. Uhlig, T. Stanley, S. Sechrest, T. Mudge, and R. Brown. 1993. Design tradeoffs for software-managed TLBs, *Proceedings of 20th International Symposium on Computer Architecture*, 27–38.

Przybylski, S., M. Horowitz, and J. Hennessy. 1989. Characteristics of performance-optimal multi-level cache hierarchies, *Proceedings of 16th International Symposium on Computer Architecture*, 114–121.

Smith, A. J. 1982. Cache memories, *Computing Surveys*, 14(3):473–530.

Sohi, G. S. and M. Franklin. 1991. High-bandwidth data memory systems for superscalar processors, *Proceedings of 4th International Conference on Architectural Support for Programming Languages and Operating Systems*, 53–62.

Sun Microsystems. 1992. Introduction to Shade / Shade User's Manual, Technical report, Sun Microsystems Laboratories, Inc.

Torrellas, J., A. Gupta and J. Hennessy. 1992. Characterizing the cache performance and synchronization behavior of a multiprocessor operating system, Technical Report CSL-TR-92-512, CSL, Dept. of EECS,

Stanford University.

## **AUTHOR BIOGRAPHIES**

**NIKI C. THORNOCK** is a research associate at Brigham Young University. She received her B.S. degree from Brigham Young University in August, 1995 and her M.S. degree in December, 1999. Her academic interests include computer architecture and performance evaluation studies.

**J. KELLY FLANAGAN** is Director of the Performance Evaluation Laboratory at Brigham Young University where he is an Associate Professor of Computer Science. His research interests include computer system performance evaluation and computer architecture. He received his PhD in Electrical Engineering from Brigham Young University. Before joining the faculty of the Computer Science Department at BYU, he spent a year at Intel Corporation in Oregon and taught a graduate architecture course for Oregon State University. He is a member of the Association for Computing Machinery and the IEEE.