

Using the BACH Trace Collection Mechanism to Characterize the SPEC 2000 Integer Benchmarks

Niki C. Thornock and J. Kelly Flanagan
Performance Evaluation Laboratory
Computer Science Department
Brigham Young University
Provo, UT 84602

August 25, 2000

Abstract

Computer systems are modeled before construction to minimize errors and performance bottlenecks. A common modeling approach is to build software models of computer system components, and use realistic *trace* data as input. This methodology is commonly referred to as trace-driven simulation. Trace-driven simulation can be very accurate if both the system model and input trace data represent the system under test. The accuracy of the model is typically under the control of the researcher, but little or no trace data is available that accurately represents current or future workloads. The objective of this work is to describe the Brigham Young University Address Collection Hardware (BACH) and illustrate the types of traces that we can collect and make available to others. We also provide some cache performance statistics for the SPEC 2000 integer benchmarks.¹

1 Introduction

Simulation is an indispensable tool in the design of a computer system and can take many forms. One common example is trace-driven simulation. This simulation technique is accurate if both the simulation model and input trace data faithfully represent the system being evaluated. Researchers have control over the accuracy of their simulation models, but there is a lack of representative trace data that is available to the research and educational communities. The purpose of this work is to describe the Brigham Young University Address Collection Hardware (BACH) and illustrate the types of traces that we can collect and make available to others. We will also

provide some data related to the cache performance of the SPEC 2000 integer benchmarks.

This paper will proceed as follows. Section 2 describes our hardware monitoring technique. Section 3 describes the cache statistics of the SPEC 2000 integer benchmarks. Finally, Section 4 summarizes our work.

2 Trace Collection Mechanism

The difficulties inherent in trace collection are well known, and the development of better trace collection, trace-driven simulation, and execution-driven simulation methods has been the goal of previous work [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. We have used the techniques described in this section to collect and distribute trace data from many systems running several operating systems and numerous application workloads. The Brigham Young University Address Collection Hardware (BACH) system is described and evaluated.

2.1 BACH: BYU Address Collection Hardware

We have designed and constructed a hardware monitor which, in conjunction with limited software support, enables the acquisition of very accurate trace data. BACH traces are extremely long, and contain all operating system references, all multitasking activity, and virtually no time dilation effects.

We have developed several novel techniques that provide BACH with the advantages of a hardware monitoring technique with few of the disadvantages. In this section, the BACH mechanism is described in general terms. The specific implementation details for an i486, Intel Pentium, Motorola 68030 and SPARC based system used to gener-

¹This material is based upon work supported by the National Science Foundation under Grant No. 9807619.

process or the operating system. This method has the advantage that no new trace records are created; it simply adds extra information to each trace record.

Full Trace Annotation

To provide more information in the trace, extra CPU references can be generated. This is done by adding code that writes useful data to unused memory or I/O locations. These data writes are collected and stored in the trace by BACH with no side effects to the system being traced. This has been used to gather information on system calls, interrupts, exceptions, and context switch points.

2.1.3 Evaluation of the BACH System

The advantages and disadvantages of other tracing methodologies are discussed in [15]. In this section, BACH and its trace data are evaluated:

- + The traces from BACH are contiguous, since the traced processor can be halted during the unloading of the trace buffer. After the buffer is emptied, the traced machine resumes execution at the point where it was interrupted and tracing resumes. All such buffers are concatenated, resulting in a complete trace. The process of collecting and concatenating buffers can continue indefinitely, resulting in traces of arbitrary length. Thus BACH permits simulation studies to be carried out using very long traces as argued for in [2].
 - + BACH captures all bus cycles generated by the CPU resulting in complete traces, prefetched references are an example [16].
 - + BACH is flexible, capable of collecting traces from a variety of hardware and software platforms. It has been used to collect traces from machines running System V R3.2 and System V R4 UNIX, Mach 2.6 and Mach 3.0 on an i486 hardware platform. BACH has also been used to collect traces from a SUN SPARC 1+ running SunOS Release 4.1.2, and a Hewlett-Packard 340, 68030-based workstation running HP-UX 7.0 [4, 14]. BACH has recently been used to collect trace data from Intel Pentium, Pentium Pro, and Pentium II based systems running MS Windows 95, Windows NT, and Red Hat Linux [13]. In addition, BACH can trace any application, regardless of whether source or object code is available, allowing the tracing of important commercial workloads [17].
- BACH traces suffer from minor dilutions caused by the device driver responsible for halting the system. While the system is halted other devices may cause interrupts that must be serviced when execution is resumed. In typical systems, the only interrupts seen at these boundaries may be the timer and hard disk routines. Analysis of trace data shows that a dilation of approximately 1.125% exists [4]. This dilation adds 1.125% more references than would be generated if the machine was not being traced. There are several other sources of trace perturbations due to BACH:
 1. The acquisition of full address traces directly requires that TRACEE's on-chip cache, if present, be disabled. This slows execution and may alter the interleaving of processes in a multiprogramming workload. Disabling multiple levels of cache in a modern system will introduce unacceptable perturbations, but this is not necessary when capturing complete instruction traces as described in [18].
 2. The interrupt routine which halts TRACEE is itself traced, resulting in additional trace references. These references can be identified and removed.
 3. The references generated by the execution of the interrupt routine which suspends the traced machine could displace otherwise useful data in cache, slightly altering normal system operation.

2.2 More Recent Trace Collection Efforts

Over the past several years BACH has undergone three major changes. The first unit was a multiple circuit board design that was capable of collecting 512 K samples of 96 bits each at a rate of 20 MHz. This system was adequate for a very short time as the clock rates and bus speeds of processors in the late 1980's and early 1990's increased rapidly. The second major version was a single board implementation of the first design which simply increased the operating speed to nearly 30 MHz. This system was used until 1993 when commercial logic analyzers with deep memory buffers became available. The third BACH used a Tektronix TLA 520 logic analyzer capable of collecting 512 K samples of 200 bits each at a rate of 100 MHz. In addition to the benefit of increased speed, the manufacturer distributes CPU probes for most commercial microprocessors. This relieves our research group from having to design future CPU probes as we have done for numerous processors in the past.

It is no longer possible to collect trace data from modern processors using our TLA 520 logic analyzer because processor and bus speeds have exceeded its capabilities. The current trace collection buffer is based around a new Tektronix TLA 700 logic analyzer. This device is currently capable of collecting 16 M samples that are each 272 bits wide. Samples can be acquired at 200 MHz. This device is connected to the SUT from which traces are acquired using a special purpose CPU probe.

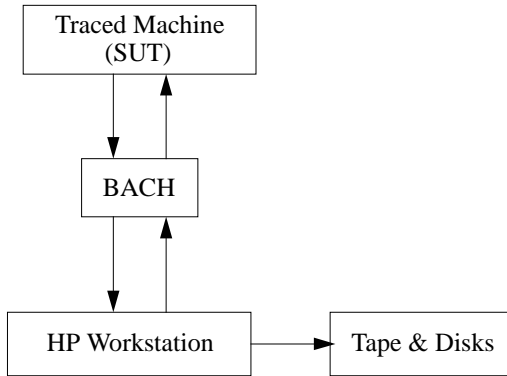


Figure 3: Trace collection configuration

2.3 Summary of Previous Work

To collect address trace data from a variety of computer systems, a hardware based tracing mechanism has been constructed. BACH uses a combination of hardware and software to overcome the buffer capacity limitation associated with previous hardware monitors. The addition of annotation software allows useful information to be placed in the trace data with little overhead or trace perturbation. All trace data collected using BACH has been made available to interested parties.

3 Characterization of SPECInt 2000

We used BACH to evaluate and characterize the SPEC 2000 integer benchmarks. We collected address traces of the first 100 million reference of each of the integer benchmarks in the SPEC 2000 suite and conducted cache simulations. The traces were taken from a Pentium II 450 MHz processor running RedHat Linux 2.2. The caches were disabled during trace collection so the traces are unfiltered. The cache configurations which were investigated were cache sizes of 8 Kbytes to 256 Kbytes with a 32 byte line size and associativities from direct-mapped to 8-way. Table 1 lists the references, instructions, reads and

writes for each trace. In cases where there were multiple input files for a benchmark, we only list information for the first input file since the results for the other files were very similar. Figures 4 through 15 show the cache miss rates for each of the benchmarks. These graphs all have the same maximum value. Figures 16 through 27 show the miss rates over time for the benchmarks. Each graph has a different maximum value in order to give a clear view of the variation of the miss rates over time. These graphs are of direct-mapped caches. We found that as the associativity became wider, the miss rate decreased but the pattern stayed the same. Table 2 is a chart of the miss rates of the cache configurations for each of the benchmarks.

4 Summary

As previously mentioned, simulation is an indispensable tool in the design of computer systems. Trace-driven simulation is widely used and is accurate if both the simulation model and input trace data are accurate and representative of the system being studied. There is a lack of representative trace data available to the research and educational communities. We have presented the BACH trace collection mechanism and illustrated the types of traces that we can collect and make available to others. We have also provided cache miss rate statistics and graphs of the miss rates over time for traces of the SPEC 2000 integer benchmarks. For information on available traces, see our webpage: <http://traces.byu.edu>.

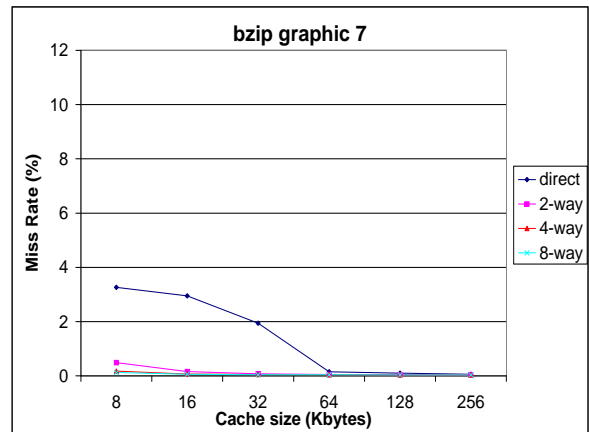


Figure 4: Miss rate graph of the Bzip2 Graphic 7 benchmark. The direct-mapped line descends at an unusual angle.

Benchmark & Input	References	Instructions	% In	Reads	% Rd	Writes	% Wr
bzip2 graphic 7	103961736	69573178	66.9%	21906274	21.1%	12482284	12.0%
crafty	103963082	63909470	61.5%	26397183	25.4%	13656429	13.1%
eon cook	103964585	61519737	59.2%	23974588	23.1%	18470260	17.8%
gap	103961748	73883772	71.1%	19981944	19.2%	10096032	9.7%
gcc 166	103962637	66419086	63.9%	24520192	23.6%	13023359	12.5%
gzip graphic	103962580	67530274	65.0%	22878841	22.0%	13553465	13.0%
mcf	103961528	72713382	69.9%	23787824	22.9%	7460322	7.2%
parser	103962716	66454690	63.9%	24827609	23.9%	12680417	12.2%
perl diffmail	103962803	65870781	63.4%	21229228	20.4%	16862794	16.2%
twolf	103962574	63974208	61.5%	27868978	26.8%	12119388	11.7%
vortex one	103964660	57713337	55.5%	23399288	22.5%	22852035	22.0%
vpr place	103962618	64479057	62.0%	28425845	27.3%	11057716	10.6%

Table 1: Statistics for each of the SPEC 2000 benchmarks. The input files all have very similar statistics so only one is displayed. Rounding may cause the percentages to not equal 100%.

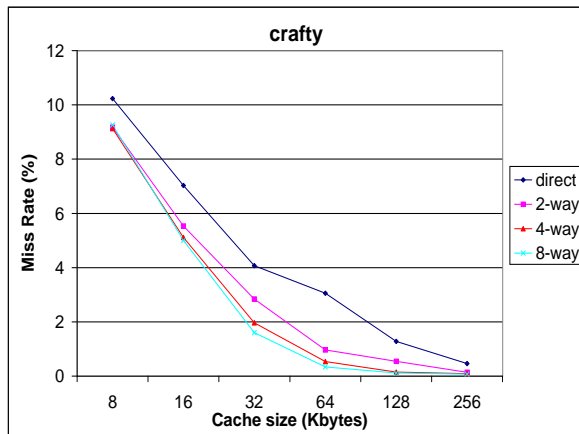


Figure 5: Miss rate graph of the Crafty benchmark. It has the highest initial miss rate.

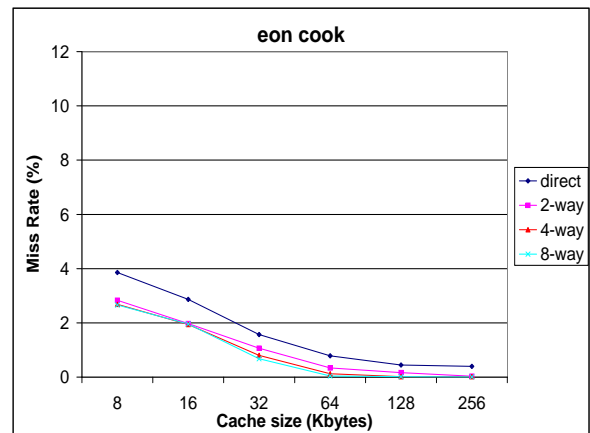


Figure 6: Miss rate graph of the Eon - Cook benchmark.

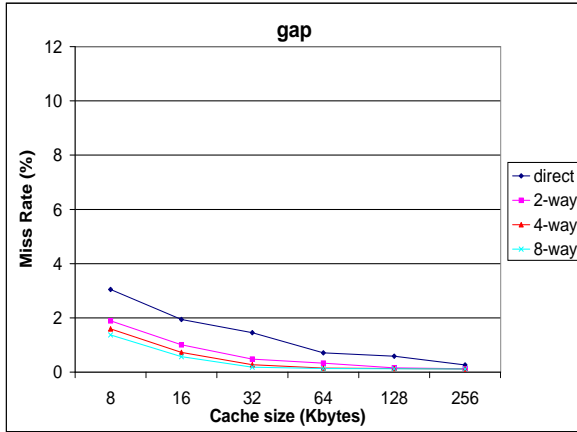


Figure 7: Miss rate graph of the Gap benchmark.

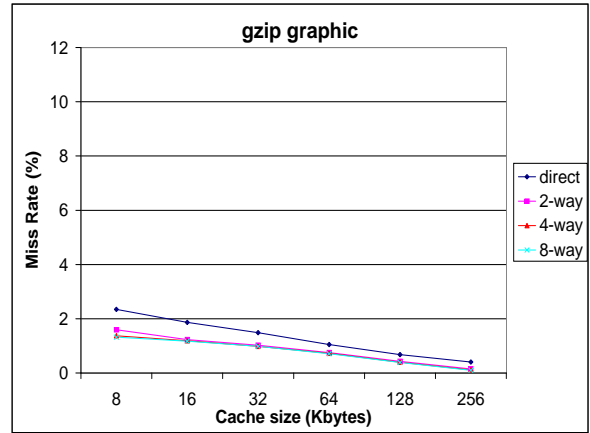


Figure 9: Miss rate graph of the Gzip-Graphic benchmark. It has a linear miss rate.

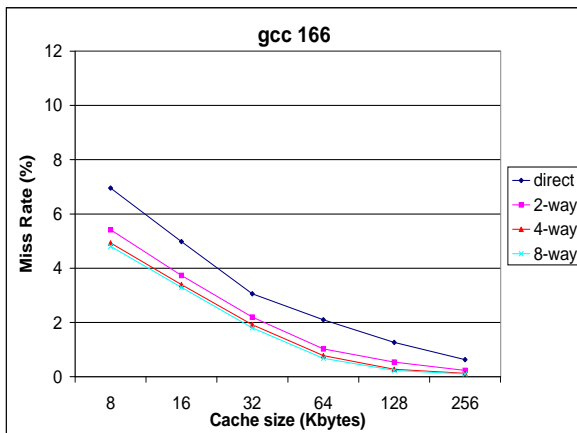


Figure 8: Miss rate graph of the Gcc-166 benchmark.

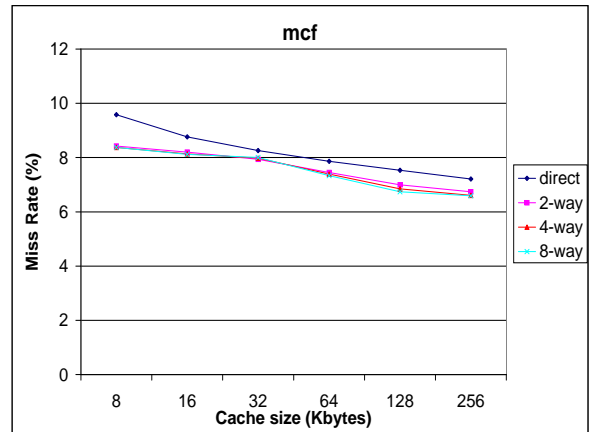


Figure 10: Miss rate graph of the Mcf benchmark. This graph has a very flat curve; the caches weren't big enough to make a difference.

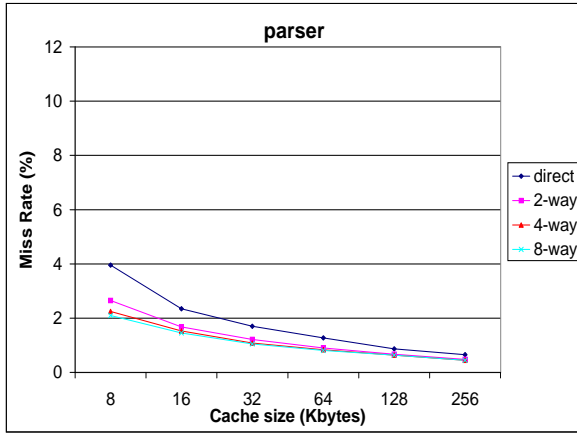


Figure 11: Miss rate graph of the Parser benchmark.

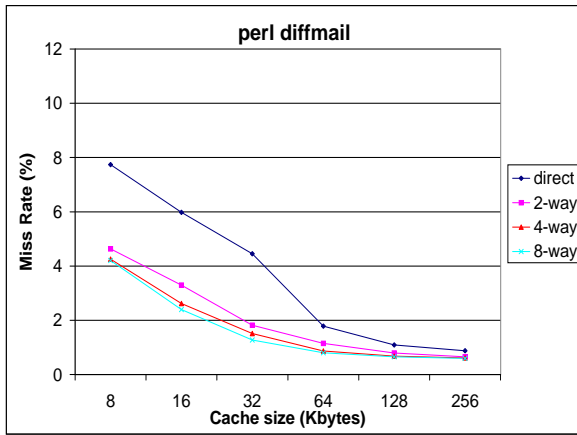


Figure 12: Miss rate graph of the Perl-Diffmail benchmark. The associativity initially has more effect than the cache size.

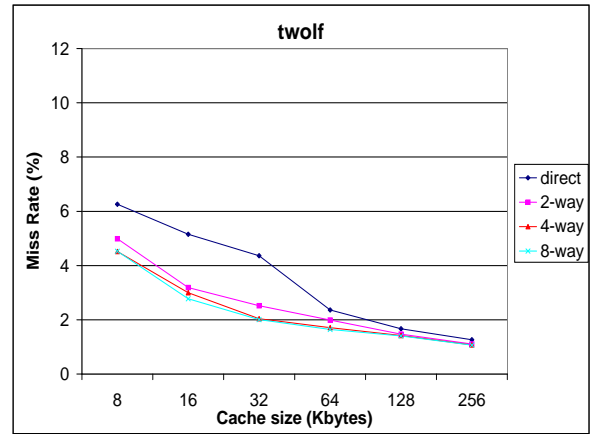


Figure 13: Miss rate graph of the TWolf benchmark.

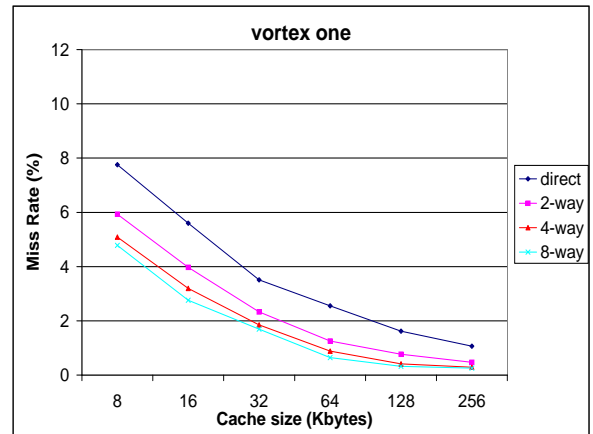


Figure 14: Miss rate graph of the Vortex One benchmark.

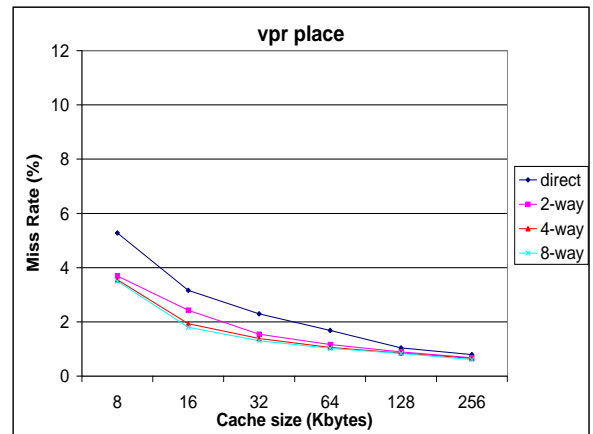


Figure 15: Miss rate graph of the VPR-Place benchmark.

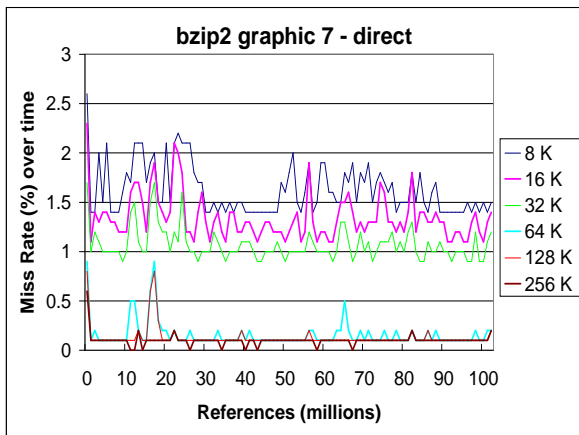


Figure 16: Direct-mapped miss rates over time graph of the Bzip2 Graphic 7 Benchmark. Each line represents a different cache size. The working set size is less than 64K. Note that the maximum value of the y-axis is different for each graph.

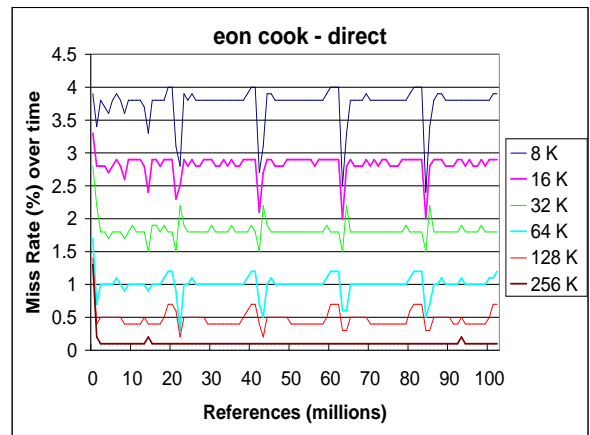


Figure 18: Direct-mapped miss rates over time graph of the Eon - Cook Benchmark. Each line represents a different cache size. Eon has a rather periodic miss rate. Note that the maximum value of the y-axis is different for each graph.

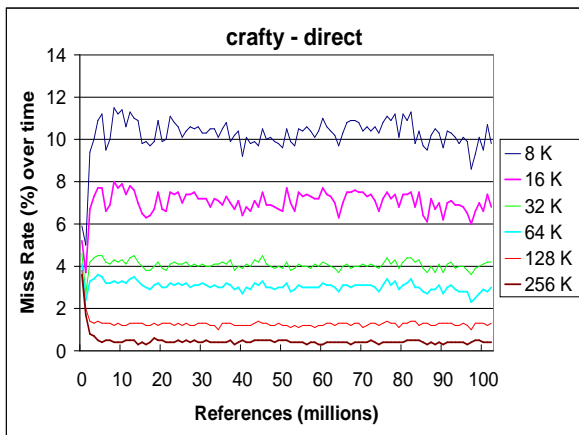


Figure 17: Direct-mapped miss rates over time graph of the Crafty Benchmark. Each line represents a different cache size. Note that the maximum value of the y-axis is different for each graph.

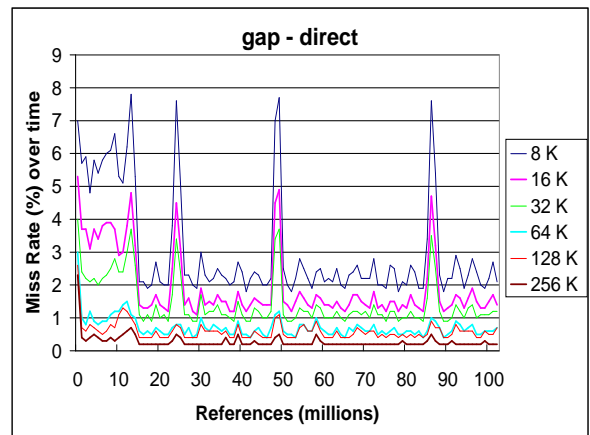


Figure 19: Direct-mapped miss rates over time graph of the Gap Benchmark. Each line represents a different cache size. Note that the maximum value of the y-axis is different for each graph.

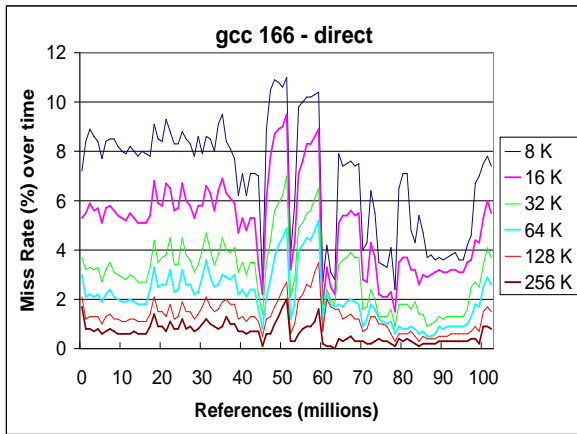


Figure 20: Direct-mapped miss rates over time graph of the Gcc 166 Benchmark. Each line represents a different cache size. The Gcc benchmark varies greatly over time. Note that the maximum value of the y-axis is different for each graph.

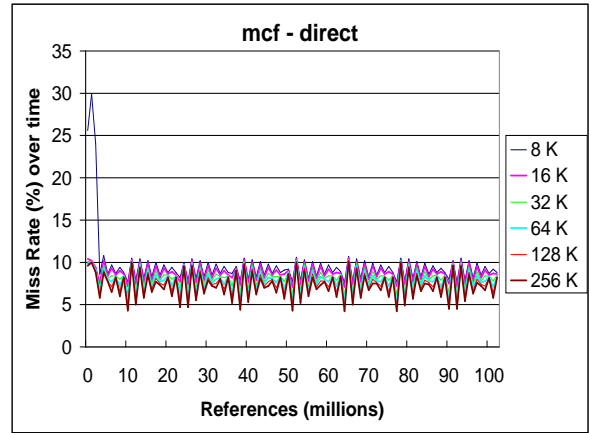


Figure 22: Direct-mapped miss rates over time graph of the MCF Benchmark. Each line represents a different cache size. Except for the direct-mapped startup cost, MCF has a very flat miss rate. All of the curves are about the same height at a miss rate of approximately 8%. Note that the maximum value of the y-axis is different for each graph.

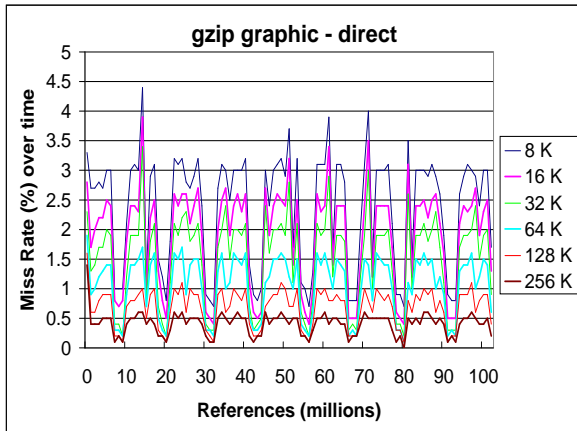


Figure 21: Direct-mapped miss rates over time graph of the Gzip Graphic Benchmark. Each line represents a different cache size. Gzip has a periodic miss rate. Note that the maximum value of the y-axis is different for each graph.

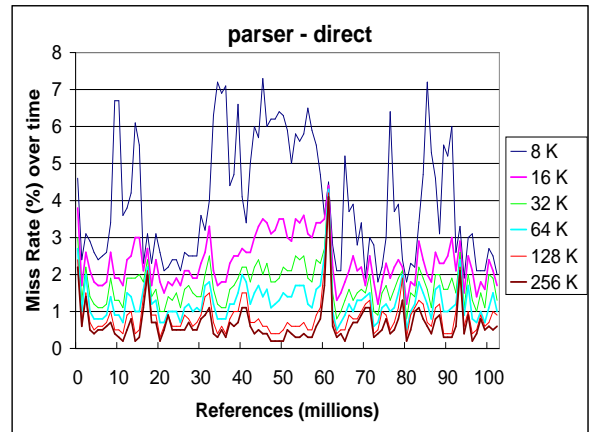


Figure 23: Direct-mapped miss rates over time graph of the Parser Benchmark. Each line represents a different cache size. Parser has a rather random miss rate over time. Note that the maximum value of the y-axis is different for each graph.

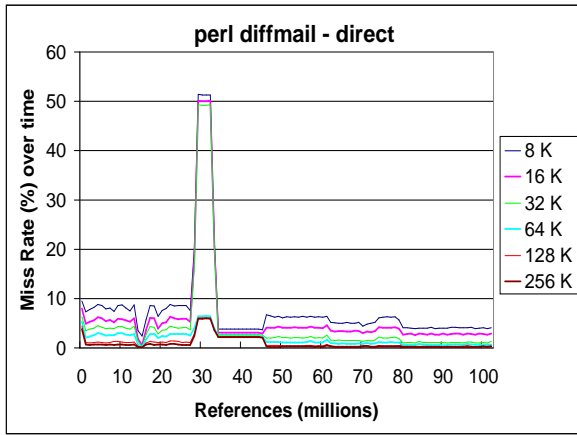


Figure 24: Direct-mapped miss rates over time graph of the Perl Diffmail Benchmark. Each line represents a different cache size. The spike at 30 million references in the Perl graph is probably caused by conflict misses since it disappears in the 2-way associative graph. Note that the maximum value of the y-axis is different for each graph.

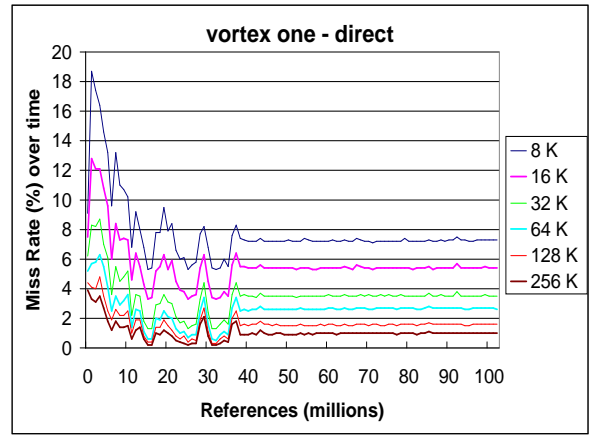


Figure 26: Direct-mapped miss rates over time graph of the Vortex One Benchmark. Each line represents a different cache size. Vortex takes about 40 million references to warm up. Note that the maximum value of the y-axis is different for each graph.

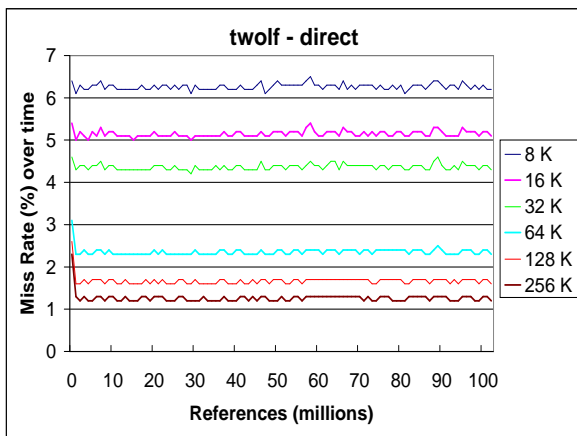


Figure 25: Direct-mapped miss rates over time graph of the TWolf Benchmark. Each line represents a different cache size. Twolf has a very flat miss rate. Note that the maximum value of the y-axis is different for each graph.

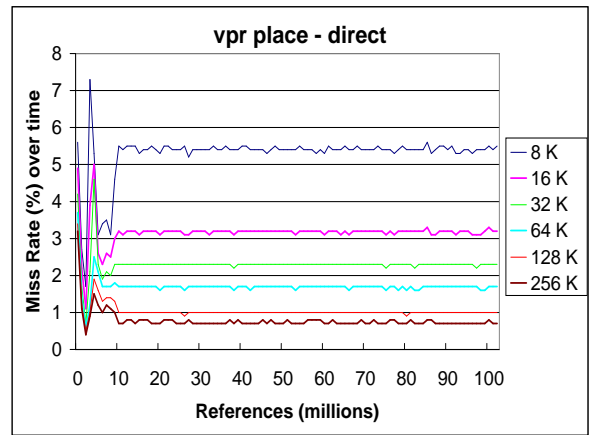


Figure 27: Direct-mapped miss rates over time graph of the VPR Place Benchmark. Each line represents a different cache size. After an initial startup cost, VPR has a very flat miss rate. Note that the maximum value of the y-axis is different for each graph.

Benchmark	Assoc	Size: 8 K	16 K	32 K	64 K	128 K	256 K
bzip2 graphic 7	1	3.261%	2.949%	1.942%	0.152%	0.097%	0.059%
	2	0.484%	0.160%	0.073%	0.050%	0.038%	0.035%
	4	0.174%	0.067%	0.047%	0.040%	0.035%	0.033%
	8	0.127%	0.059%	0.045%	0.039%	0.034%	0.033%
crafty	1	10.235%	7.029%	4.070%	3.057%	1.279%	0.465%
	2	9.175%	5.534%	2.838%	0.964%	0.546%	0.138%
	4	9.125%	5.111%	1.963%	0.538%	0.147%	0.093%
	8	9.260%	5.001%	1.600%	0.341%	0.117%	0.076%
eon cook	1	3.854%	2.868%	1.572%	0.781%	0.447%	0.398%
	2	2.836%	1.974%	1.068%	0.340%	0.169%	0.036%
	4	2.684%	1.942%	0.797%	0.126%	0.018%	0.013%
	8	2.660%	1.961%	0.678%	0.038%	0.016%	0.012%
gap	1	3.047%	1.941%	1.451%	0.707%	0.586%	0.268%
	2	1.890%	1.008%	0.479%	0.332%	0.158%	0.124%
	4	1.590%	0.728%	0.273%	0.152%	0.129%	0.115%
	8	1.372%	0.573%	0.180%	0.135%	0.125%	0.114%
gcc 166	1	6.950%	4.983%	3.054%	2.101%	1.266%	0.624%
	2	5.415%	3.736%	2.198%	1.020%	0.535%	0.231%
	4	4.927%	3.392%	1.918%	0.780%	0.275%	0.128%
	8	4.810%	3.298%	1.799%	0.680%	0.230%	0.101%
gzip graphic	1	2.346%	1.865%	1.484%	1.048%	0.675%	0.406%
	2	1.593%	1.229%	1.023%	0.755%	0.427%	0.149%
	4	1.375%	1.190%	0.983%	0.727%	0.393%	0.117%
	8	1.324%	1.174%	0.980%	0.714%	0.390%	0.098%
mcf	1	9.584%	8.760%	8.261%	7.863%	7.530%	7.214%
	2	8.428%	8.205%	7.937%	7.453%	6.995%	6.739%
	4	8.377%	8.136%	7.981%	7.390%	6.850%	6.610%
	8	8.375%	8.116%	8.001%	7.335%	6.742%	6.588%
parser	1	3.953%	2.349%	1.703%	1.269%	0.865%	0.653%
	2	2.654%	1.680%	1.216%	0.898%	0.668%	0.481%
	4	2.246%	1.528%	1.082%	0.831%	0.636%	0.443%
	8	2.097%	1.457%	1.046%	0.813%	0.634%	0.449%
perl diffmail	1	7.735%	5.978%	4.455%	1.782%	1.092%	0.874%
	2	4.631%	3.292%	1.817%	1.151%	0.792%	0.650%
	4	4.254%	2.618%	1.514%	0.870%	0.679%	0.608%
	8	4.204%	2.392%	1.272%	0.805%	0.650%	0.598%
twolf	1	6.262%	5.155%	4.362%	2.360%	1.671%	1.265%
	2	4.985%	3.187%	2.521%	1.990%	1.472%	1.106%
	4	4.514%	2.996%	2.044%	1.713%	1.417%	1.073%
	8	4.524%	2.772%	2.010%	1.640%	1.402%	1.068%
vortex one	1	7.757%	5.598%	3.514%	2.555%	1.618%	1.066%
	2	5.932%	3.975%	2.331%	1.258%	0.766%	0.471%
	4	5.084%	3.200%	1.847%	0.886%	0.416%	0.287%
	8	4.785%	2.755%	1.692%	0.648%	0.318%	0.260%
vpr place	1	5.281%	3.163%	2.298%	1.688%	1.042%	0.790%
	2	3.695%	2.427%	1.547%	1.166%	0.895%	0.676%
	4	3.554%	1.931%	1.384%	1.059%	0.847%	0.652%
	8	3.500%	1.799%	1.308%	1.025%	0.835%	0.614%

Table 2: Miss rates for the SPEC 2000 integer benchmarks.

References

- [1] A. Agarwal, R. L. Sites, and M. Horowitz, ATUM: A new technique for capturing address traces using microcode, In *Proc. of 13th Int. Symp. on Computer Architecture*, pages 119–127. IEEE, 1986.
- [2] A. Borg, R. E. Kessler, and D. W. Wall, Generation and analysis of very long address traces, In *Proc. of 17th Int. Symp. on Computer Architecture*, pages 270–279. ACM, 1990.
- [3] D. Nagle, R. Uhlig, and T. Mudge, Monster: A tool for analyzing the interaction between operating systems and computer architectures, Technical report, The University of Michigan, 1992.
- [4] J. Kelly Flanagan, Brent E. Nelson, James K Archibald, and Knut Grimsrud, BACH: BYU Address Collection Hardware, the collection of complete traces, In *Proc. of the 6th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 128–137, 1992.
- [5] J. Kelly Flanagan, *A New Methodology for Accurate Trace Collection and its Application to Memory Hierarchy Performance Modeling*, PhD thesis, Brigham Young University, December 1993.
- [6] J. B. Chen and B. Bershad, The impact of operating system structure on memory system performance, In *Proc. of 14th Symp. on Operating System Principles*. ACM, 1993.
- [7] A. Srivastava and A. Austace, ATOM: A system for building customized program analysis tools, In *Proc. of the Conference on Program Language Design and Implementation*, pages 196–205. ACM, 1994.
- [8] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta, The SimOS approach, *IEEE Parallel and Distributed Technology*, **4**(3), 1995.
- [9] D. Burger, T. Austin, and S. Bennett, Evaluating future microprocessors: The SimpleScalar tool set, Technical Report #1308, University of Wisconsin - Madison, July 1996.
- [10] P. A. Sandon, Y.-C. Liao, T. E. Cook, D. M. Schultz, and P. Martin de Nicolas, Nstrace: A bus-driven instruction trace tool for powerpc microprocessors, *IBM Journal of Research and Development*, **41**(3):331–344, May 1997.
- [11] J. Kelly Flanagan, Brent E. Nelson, James K Archibald, and Knut Grimsrud, Incomplete trace data and trace driven simulation, In *Proc. of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems MASCOTS*, pages 203–209. SCS, 1993.
- [12] V. Phalke and B. Gopinath, Compression-based program characterization for improving cache memory performance, *IEEE Transactions on Computers*, **46**(11):1174–1186, November 1997.
- [13] K. Flanagan, J. Archibald, and J. Su, Low power memory hierarchies: An argument for second-level caches, *Microprocessors and Microsystems*, **21**(5):279–290, February 1998.
- [14] K. Grimsrud, J. Archibald, M. Ripley, K. Flanagan, and B. Nelson, BACH: A hardware monitor for tracing microprocessor-based systems, *Microprocessors and Microsystems*, **17**(6), October 1993.
- [15] K. Flanagan, B. Nelson, J. Archibald, and G. Thompson, The inaccuracy of trace-driven simulation using incomplete multiprogramming trace data, In *IEEE International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 1996.
- [16] Douglas W. Clark, Cache performance in the VAX-11/780, *ACM Transactions on Computer Systems*, **1**(1):24–37, February 1983.
- [17] G. Thompson, B. Nelson, and K. Flanagan, Transaction processing workloads – a comparison to the SPEC benchmarks using memory hierarchy performance studies, In *IEEE International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 1996.
- [18] C. Rose and K. Flanagan, Complete instruction traces from incomplete address traces (CITCAT), *Computer Architecture News*, **24**(5):1–8, December 1996.