

INCOMPLETE TRACE DATA AND TRACE DRIVEN SIMULATION

J. Kelly Flanagan, Brent E. Nelson, James K Archibald, Knut Grimsrud
Department of Electrical and Computer Engineering
Brigham Young University
Provo, UT 84602

May 31, 1995

Abstract— We illustrate the sensitivity of trace driven simulation results to the completeness of the input trace data. This is done by comparing cache simulation results obtained using traces similar to many found in the published literature with those obtained using a hardware monitor which we have developed. Cache miss rates from single process traces without operating system references are shown to be in error by as much as a factor of 50. Published synthetic multiprogram workloads are then shown to result in simulated cache miss rates which are in error by factors that range from 0.9 to 110. Finally, it is shown that errors of this magnitude yield misleading performance estimates.

INTRODUCTION AND BACKGROUND

Trace driven simulation has been used for many years to predict the performance of computer systems. This method of simulation may provide useful estimates of system performance, but is dependent on the quality of both the simulation model and the input trace data. While the generation of complete address traces has been the subject of previous work [ASH86], little has been done to compare simulation results using them with those obtained using the incomplete and suspect [Gai91] traces used in published studies [OMB91, WHK91].

We have constructed a hardware monitor capable of producing long and complete traces on a variety of hardware and software platforms. Using these highly accurate traces we quantify the error in system performance predictions which result from the use of incomplete trace data. For this study, we define *complete address traces* to be those traces containing all CPU generated references including those produced by interrupts, system calls, exception handlers, other supervisor activities, and user processes.

First, we describe our trace gathering technique and its advantages over other reported methods. We also discuss and quantify the dilation effects present in our

traces. We then examine traces resulting from the execution of a single process by itself. We demonstrate that not only does the cache miss rate vary widely from one portion of a program to another (as previously reported in [BKW90]), but that the majority of cache misses in large sections of the program are due to system code such as interrupts and system calls and that user code is insignificant in these sections. We show that miss rates predicted from our complete traces are as much as 50 times higher than those predicted from “user code only” traces, similar to those used in many previously published studies.

We then turn to multiprogram trace generation. Using complete multiprogram traces collected with our monitor we demonstrate that proposed synthetic trace generation methods for multiprogram workloads may give results little better than those obtained by simply concatenating a collection of single-process traces together. Indeed, we show that more complex schemes which attempt to accurately model context switch intervals are no better than simpler methods and may result in cache miss rate predictions in error by as much as a factor of 110. Finally, we use a simple memory hierarchy performance model to demonstrate how small miss rate prediction errors can result in substantial errors in the estimation of effective access times.

TRACE GATHERING TECHNIQUE

The most accurate way to collect address and data reference streams is to use a hardware monitoring device. Limitations of this technique as pointed out by Agarwal et al. [ASH86] are its complexity and limited capacity. We have built a hardware monitor which, in conjunction with limited software support, overcomes these limitations. It is capable of collecting traces which include all references generated by the CPU: user references, system call references, interrupt routine references, exception handler references, instruction prefetches, and

special bus cycles. For each CPU reference the address, data values, and control signals are captured, providing sufficient information to identify specific reference sequences. Furthermore, we can identify the exact interrupt routine or system call made.

This hardware monitor has been used to collect traces from machines running UNIX SysV R3.2, UNIX SysV R4, Mach 2.6, and Mach 3.0. It can be used to trace any application, since neither source nor object files are required. This allows tracing of many commercial application for which source code is not available.

Unlike previously published monitors it can be used to trace a variety of hardware platforms. It consists of 3 major parts: a trace buffer capable of holding 500,000 references, a module which downloads the trace buffer contents to mass storage, and a processor-specific probe module to capture the desired signals. As a result, it is relatively simple to adapt to different CPUs. To date, it has been used on i486 and MC68030 based workstations and a SPARC interface is currently being constructed. For this study, traces collected on the i486 running Mach 2.6 were used.

SINGLE PROCESS WORKLOAD RESULTS

The programs traced were taken from a variety of applications areas: (i) system utilities such as grep, sort, sed, and nroff, (ii) numerical codes including Gaussian elimination, fft, integration, and determinant, (iii) CAD tools such as esim, eqntott, and cnet, and (iv) a LISP routine solving the 4 queens problem and a simple jigsaw puzzle benchmark.

Each application was traced without other user processes running. This resulted in 13 traces containing single applications and operating system references. Each application was traced from the beginning of its execution until its normal termination.

The characteristics of these traces are presented in Table 1. From the table it can be seen that the percentage of operating system references in the traces varies from 12% to 28%. Note that the last three columns of the table do not total 100% — CPU bus cycles such as interrupt and other special bus cycles make up the remaining references.

Operating System References and Cache Miss Rates

To demonstrate the impact of operating system references on system behavior, a simulator was used to evaluate cache miss rates in a number of ways.

We first observe that many studies have been based on traces lacking operating system references. To compare our complete traces with these, we created user-only traces similar to those used in [BKW90, Inc88] by

removing the operating system references from our *nroff* trace. These were then used in a set of cache simulations. The resulting miss rates were compared to those obtained using the complete trace and the error computed as

$$EF = \frac{MR_{actual}}{MR_{subtrace}} \quad (1)$$

where EF is the error factor, MR_{actual} is the miss rate predicted using the full trace and $MR_{subtrace}$ is the miss rate predicted using the subtrace containing only user references. The results for a number of cache configurations are presented in Table 2. For small caches, the miss rates are high enough that the cache interference due to operating system references is comparable to the interference from user references within the same process. For large caches, more of the process and system code fit into the cache and the miss rate improvement due to increased associativity diminishes for the user-only trace, reducing the error factor. For mid-sized caches, the benefits of associativity are overemphasized for the user-only trace, producing optimistic user-only miss rates.

To demonstrate the contribution of the various operating system components to the overall miss rate, we then created Figure 1 by first applying a user-only reference trace to a 64 KByte direct-mapped unified cache with 16 byte lines. The remaining trace components (system calls, exceptions, and interrupts) were then incrementally added to the reference stream for each subsequent cache simulation in order to determine the contribution of each trace component. Note that for most of the trace, the major contributor to the miss rate is the operating system. In particular, the contributions made by the exception handlers and interrupt routines are significant.

Finally, we completed similar simulation runs for a number of test programs. For these runs, however, we used eight subtraces, representing combinations of user and system code components:

<i>u</i>	User references only
<i>ui</i>	User references and interrupt routines
<i>ue</i>	User references and exception handlers
<i>us</i>	User references and system calls
<i>uie</i>	User references, interrupts, and exceptions
<i>uis</i>	User references, interrupts, and system calls
<i>ues</i>	User references, exceptions, and system calls
<i>eis</i>	System references only

Additionally, we traced the same applications using an instruction modification or inlining technique which yields user-only traces similar to [BKW90, Inc88]. The difference between the *inline* and *u* traces are that the *u*

Name	References	% OS	% User	% Ifetches	% Reads	% Writes
cnet	5,989,097	14.62%	85.38%	73.97%	15.45%	10.39%
grep	12,557,619	20.34%	79.66%	81.78%	11.77%	6.25%
nroff	14,613,781	17.73%	82.27%	80.85%	12.44%	6.51%
sed	13,628,111	15.12%	84.88%	81.20%	14.31%	4.24%
jigsaw	15,214,820	14.93%	85.07%	76.14%	13.62%	10.14%
det	8,142,566	14.25%	85.75%	69.35%	20.06%	10.51%
fft	12,091,984	12.30%	87.70%	66.95%	18.56%	14.41%
gauss	6,834,097	13.41%	86.59%	75.12%	15.35%	9.45%
int	6,080,158	16.13%	83.87%	58.68%	25.78%	15.44%
sort	5,682,538	22.28%	77.72%	75.65%	18.27%	5.84%
eqntott	7,299,706	26.30%	73.70%	76.44%	14.52%	8.95%
queen	19,376,617	13.19%	86.81%	74.70%	15.72%	9.51%
esim	13,239,488	28.00%	72.00%	74.76%	17.07%	7.99%

Table 1: Single process trace characteristics

traces also contain references generated by the instruction prefetch unit, the *inline* traces do not.

All 117 subtraces were used in a simulation study of a 64 Kbyte direct-mapped unified data and instruction cache with 16 byte lines. The miss rates were applied to Equation (1) with the resulting error factors presented in Figure 2. Note that given two subtraces, one is not consistently better in predicting miss rate and that all of the subtraces gave optimistic miss rate estimates. Subtrace *eis*, the system references alone, produced very pessimistic results (4 times the error factor of other subtraces) and is not shown in Figure 2. In summary, the data shows that neither operating system nor user references alone are accurate predictors of miss rate.

MULTIPROGRAM WORKLOADS

It is generally accepted that in order to accurately model system behavior, traces containing multiprogram workloads are required [OMB91, WHK91, BKW90, MB91]. Due to the lack of (generally available) long and contiguous multiprogram traces that include both user and system references, researchers have proposed several methods of synthetically generating such workloads. In this section we compare these with our multiprogram traces by evaluating how well they predict cache miss rates.

Our Multiprogram Traces

The multiprogram traces we collected were created by running combinations of the programs described above. Table 3 gives the characteristics of these workloads.

Synthetic Workloads

We created the following synthetic workloads for this study:

CAT: Each single-process trace was stripped of all operating system references to approximate trace data such as in [Inc88]. These traces were then concatenated together.

ICAT: Similar to CAT but used the *inline traces*.

RR50, IRR50, IRR500: RR50 was implemented by interleaving traces at fixed 50,000 reference intervals in a round-robin fashion (the original trace was first stripped of operating system references). IRR50 and IRR500 used the inline traces and were switched at 50,000 and 500,000 reference intervals, respectively.

RR500S: RR500S is a variation of RR50 method in which the time slice was increased to 500,000 references and a context switch was made on every system call. This corresponds to the method described in Olukotun et al. [OMB91].

RR500SOS: This is identical to RR500S except that it includes all operating system references. It is created by switching between single process traces (containing all user and system references) at 500,000 reference intervals and at system calls.

MPUSER: Finally, MPUSER, was produced by stripping an actual multiprogram trace of all system references, producing a user-only multipro-

Size=	4K	8K	16K	32K	64K	128K	256K
Assoc=1	6.89%	3.95%	2.31%	1.28%	0.69%	0.44%	0.28%
Assoc=2	4.21%	2.48%	1.53%	0.88%	0.48%	0.26%	0.16%
Assoc=4	3.43%	2.00%	1.29%	0.74%	0.37%	0.19%	0.13%
Assoc=8	3.17%	1.91%	1.19%	0.69%	0.29%	0.15%	0.13%
Assoc=16	3.07%	1.89%	1.17%	0.69%	0.25%	0.15%	0.13%

a) Complete Trace Miss Rates

Size=	4K	8K	16K	32K	64K	128K	256K
Assoc=1	6.11%	2.84%	1.31%	0.52%	0.12%	0.05%	0.04%
Assoc=2	3.19%	1.45%	0.56%	0.17%	0.07%	0.04%	0.04%
Assoc=4	2.32%	0.95%	0.35%	0.08%	0.04%	0.04%	0.04%
Assoc=8	2.08%	0.88%	0.25%	0.06%	0.04%	0.04%	0.04%
Assoc=16	2.02%	0.88%	0.23%	0.05%	0.04%	0.04%	0.04%

b) User-Only Trace Miss Rates

Size=	4K	8K	16K	32K	64K	128K	256K
Assoc=1	1.1	1.4	1.8	2.5	6.0	9.1	6.7
Assoc=2	1.3	1.7	2.7	5.1	6.7	6.5	4.0
Assoc=4	1.5	2.1	3.7	9.2	8.6	5.0	3.5
Assoc=8	1.5	2.2	4.8	10.8	7.2	3.9	3.3
Assoc=16	1.5	2.2	5.1	13.1	6.3	3.8	3.3

c) Error Factors ($EF = \frac{MR_{actual}}{MR_{subtrace}}$)

Table 2: Complete and User-Only *nroff* Trace Results

grammed trace, similar to those used by Wood et al. [WHK91].

When these synthetic workloads were generated synthetic process identifiers were concatenated to the actual addresses to eliminate the problem of duplicate addresses. Also, note that for all synthetic workload generation techniques, the additional O/S references due to multiprogram overhead are absent.

These traces were used in the simulation of a 64 Kbyte direct-mapped cache with 16 byte lines. The resulting miss rates are shown in Table 4. Also shown at the bottom of Table 4 is the average error factor for each workload generation technique as computed using Equation (1). The average error factors at the bottom of Table 4 can be used to make several interesting observations. The best synthetic workload generation technique is RR500SOS which requires long traces containing all user and operating system references for each application. Obtaining this type of trace data is difficult and was the major driving force in the creation of our hard-

ware monitor. As with our tracing mechanism, if this type of trace can be produced it is also easy to generate the real multiprogram trace. In this case, creating a synthetic workload would be pointless.

The MPUSER trace was generated by removing the supervisor references from the multiprogram trace which left all context switch points intact. As Figure 1 demonstrate, removing the supervisor references from a trace reduces the associated cache miss rate, accounting for the MPUSER error factor.

RR50 is one of the simplest methods, lacking realistic context switch intervals and system references. Yet, it performed better than more sophisticated schemes. This is not due to a better choice of context switch times, additional references, or more realistic process ordering. Rather, the short run of each process (50,000 references) reduces the locality of the trace data. This increases the cache miss rate, reducing the overall error factor. Further reducing the run length results in even smaller error factors, but this decrease in time slice has

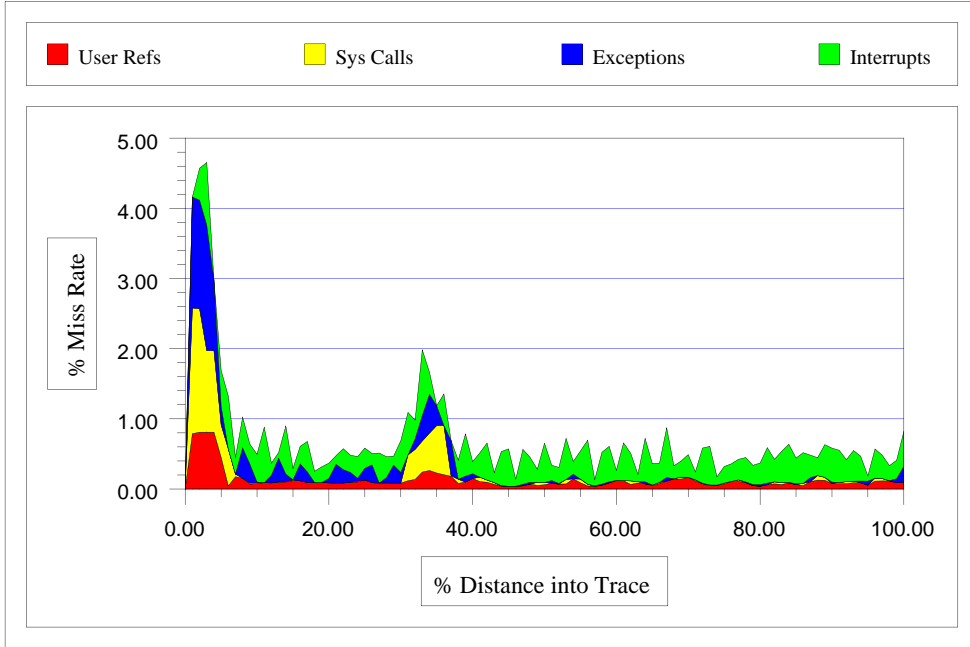


Figure 1: Miss rate contributors and variation in the *nroff* trace using a 64 K direct-mapped cache

no relationship to real system behavior.

Finally, all of the techniques based on inlining gave poor results. We attribute this to their lack of operating system references and the large number of fetches made by the instruction prefetch unit which they cannot capture.

EFFECTS ON SYSTEM PERFORMANCE

Comparing ratios of miss rates as in Tables 2 and 4 may be misleading without examining the actual miss rates and their effect on system performance. Specifically, for cache configurations where the miss rates are already very small, accurately predicting those same miss rates may have little impact on the accuracy of the resulting performance measure.

One measure of system performance is effective memory access time which may be calculated from:

$$t_{eff}(T) = t_{cache} + m(T) \times t_{miss} \quad (2)$$

where t_{cache} , $m(T)$, and t_{miss} are the cache hit time, cache miss rate for a trace T , and cache miss time respectively. Note that for a given configuration the effective access time varies only with the miss rate.

It is interesting to examine what impact the incorrect miss rate predictions shown in Table 4 would have on

effective access time. Consider the case of a memory hierarchy with a 64 Kbyte direct-mapped cache where $t_{miss} = 30$ and $t_{cache} = 1$. Using the *csn* trace generated from the MPUSER synthetic workload model, the estimated effective memory access time is:

$$\begin{aligned} t_{eff}(CSN_{MPUSER}) &= 1.0 + m(CSN_{MPUSER}) \times 30 \\ &= 1.0 + 0.2\% \times 30 \\ &= 1.06 \end{aligned}$$

If the real multiprogram *csn* trace were used in this analysis the actual effective access time would be:

$$\begin{aligned} t_{eff}(CSN_{actual}) &= 1.0 + m(CSN_{actual}) \times 30 \\ &= 1.0 + 3.3\% \times 30 \\ &= 1.99 \end{aligned}$$

Thus the estimate of average memory access time is off by a factor of 1.88 for this illustration; similar errors can be computed for different cache miss times. Since memory access time is often the limiting factor in the performance of a system, an effective doubling of memory access time would lead to a comparable degradation in system performance.

CONCLUSIONS

We have illustrated the sensitivity of trace driven sim-

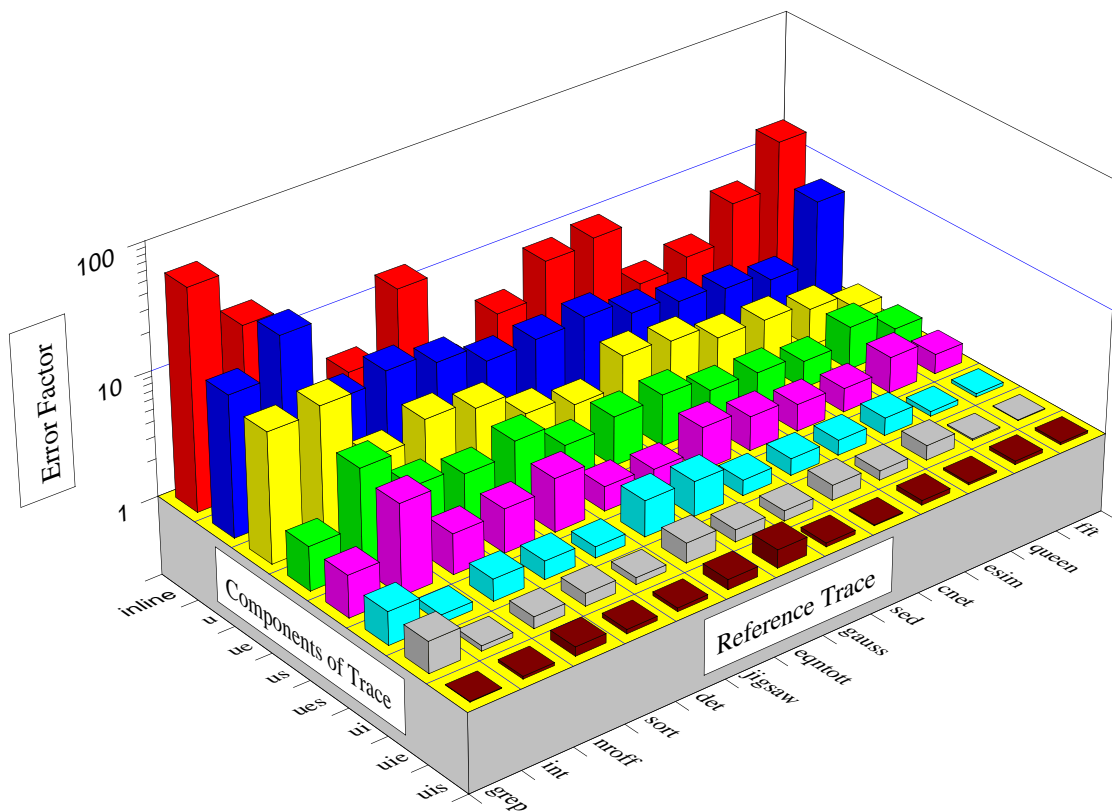


Figure 2: Contribution to simulation error of each trace component

ulation results to the accuracy of the input trace data. This was done by comparing cache simulation results obtained using traces similar to many found in the literature with those obtained using a hardware monitor which we have developed.

Using traces of single processes we have shown that the major contributors to cache miss rate are the interrupt and exception portions of the operating system. Traces without these two components result in cache miss rate estimates in error by as much as a factor of 50.

Synthetic workload generation schemes similar to those proposed in the literature were also evaluated and the corresponding cache simulation errors were quantified. These errors range from a factor of 0.9 to more than 110.

Finally, we have shown that errors of this magnitude in miss rate prediction can result in performance prediction errors in the range of 12% to 36% for systems with cache miss times between 5 and 15 cycles.

These results indicate that operating system references as well as realistic multiprogram workloads are

essential to cache simulation studies. Since perturbations to the input trace data have a significant impact on simulation results, the error introduced by incomplete traces should be quantified for the results to be meaningful. In the future we plan to extend this work to a variety of hardware platforms and operating systems. We are particularly interested in the applicability of the results to UNIX SysV R4 and Mach 3.0 on both RISC and CISC machines.

REFERENCES

- [ASH86] 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.
- [BKW90] 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.

Name = Components	References	% OS	% User	% Ifetches	% Reads	% Writes
dfgiq = det fft gauss int queen	54,952,010	16.93%	83.07%	70.12%	18.26%	11.54%
cgnss = cnet grep nroff sort sed	51,652,388	17.90%	82.10%	79.49%	13.82%	6.57%
sifgc = sort int fft gauss cnet	37,640,917	16.84%	83.16%	69.32%	18.69%	11.88%
ecif = eqtott cnet int fft	32,225,152	18.63%	81.37%	68.82%	18.50%	12.60%
gisn = gauss int sort nroff	32,408,898	18.89%	81.11%	74.04%	16.73%	9.08%
csn = cnet sort nroff	18,242,205	20.62%	79.38%	76.53%	15.38%	7.90%
ggn = grep gauss nroff	30,197,703	17.72%	82.28%	79.66%	12.90%	7.31%
ge = grep esim	20,778,604	20.46%	79.54%	78.94%	13.67%	7.27%

Table 3: Multiprogram trace characteristics

Name	ACTUAL	RR500SOS	RR50	MPUSER	RR500S	CAT	ICAT	IRR50	IRR500
dfgiq	0.72%	0.79%	0.38%	0.26%	0.32%	0.28%	0.06%	0.06%	0.03%
cgnss	1.14%	0.91%	0.27%	0.49%	0.20%	0.14%	0.15%	0.07%	0.03%
ecif	2.69%	1.00%	0.43%	0.18%	0.38%	0.35%	0.13%	0.05%	0.03%
sifgc	2.45%	0.80%	0.34%	0.18%	0.30%	0.27%	0.08%	0.05%	0.03%
csn	3.30%	0.81%	0.25%	0.21%	0.18%	0.14%	0.24%	0.07%	0.04%
ggn	2.58%	0.87%	0.16%	0.20%	0.13%	0.10%	0.14%	0.06%	0.03%
ge	3.08%	1.26%	0.32%	0.38%	0.28%	0.25%	0.21%	0.04%	0.03%
gisn	2.99%	0.71%	0.16%	0.31%	0.12%	0.09%	0.15%	0.05%	0.03%
AVG.	2.37%	0.88%	0.24%	0.24%	0.20%	0.16%	0.14%	0.05	0.03%
AVG. EF	1.0	2.7	9.7	10.0	12.1	15.2	17.1	46.0	79.7

Table 4: Miss rates and average error factors for various workload generation techniques, 64K byte direct-mapped cache

- [Gai91] Blaine Gaither. Empty Empiricism. *Computer Architecture News*, 19(4):4–5, June 1991. Ratios. In *Proc. of 1991 ACM Sigmetrics*, pages 79–89. ACM, 1991.
- [Inc88] MIPS Computer Systems, Inc. *RISCompiler Languages Programmer's Guide*. 1988.
- [MB91] J. C. Mogul and A. Borg. The Effect of Context Switches on Cache Performance. In *Proc. of 4th Int. Conf. on Arch. Support for Prog. Lang. and Operating Systems*, pages 75–84. ACM, 1991.
- [OMB91] O. A. Olukotun, T. N. Mudge, and R. B. Brown. Implementing a Cache for a High-Performance GaAs Microprocessor. In *Proc. of 18th Int. Symp. on Computer Architecture*, pages 138–147. ACM, 1991.
- [WHK91] D. A. Wood, M. D. Hill, and R. E. Kessler. A Model for Estimating Trace-Sample Miss